

---

# **PyNLPI Documentation**

***Release 1.2.8***

**Maarten van Gompel**

**Nov 12, 2018**



<b>1</b>	<b>Common Functions</b>	<b>3</b>
<b>2</b>	<b>Data Types</b>	<b>5</b>
<b>3</b>	<b>Evaluation &amp; Experiments</b>	<b>9</b>
<b>4</b>	<b>FoLiA library</b>	<b>13</b>
4.1	Reading FoLiA . . . . .	13
4.1.1	Loading a document . . . . .	13
4.1.2	Printing text . . . . .	21
4.1.3	Index . . . . .	22
4.1.4	Elements . . . . .	22
4.1.5	Obtaining list of elements . . . . .	94
4.1.6	Select method . . . . .	95
4.1.7	Selection Shortcuts . . . . .	95
4.1.8	Navigating a document . . . . .	96
4.1.9	Structure Annotation Types . . . . .	97
4.1.10	Common attributes . . . . .	425
4.1.11	Annotations . . . . .	425
4.2	Editing FoLiA . . . . .	797
4.2.1	Creating a new document . . . . .	797
4.2.2	Declarations . . . . .	797
4.2.3	Adding structure . . . . .	797
4.2.4	Adding annotations . . . . .	798
4.2.5	Adding span annotation . . . . .	799
4.2.6	Deleting annotations . . . . .	800
4.2.7	Copying annotations . . . . .	800
4.3	Searching in a FoLiA document . . . . .	801
4.3.1	Corpus Query Language (CQL) . . . . .	801
4.3.2	FoLiA Query Language (FQL) . . . . .	802
4.3.3	Streaming Reader . . . . .	804
4.4	Higher-Order Annotations . . . . .	806
4.4.1	Text Markup . . . . .	806
4.4.2	Features . . . . .	859
4.4.3	Alternatives . . . . .	915
4.4.4	Corrections . . . . .	940
4.4.5	Alignments . . . . .	1006

4.4.6	Descriptions, Metrics . . . . .	1029
4.5	Metadata . . . . .	1051
<b>5</b>	<b>Formats</b>	<b>1053</b>
5.1	Corpus Gesproken Nederlands . . . . .	1053
5.2	FoLiA . . . . .	1053
5.3	GIZA++ . . . . .	1053
5.4	Moses . . . . .	1054
5.5	SoNaR . . . . .	1054
5.6	Taggerdata . . . . .	1055
5.7	TiMBL . . . . .	1055
<b>6</b>	<b>Language Models</b>	<b>1057</b>
<b>7</b>	<b>Search Algorithms</b>	<b>1059</b>
<b>8</b>	<b>Statistics and Information Theory</b>	<b>1061</b>
8.1	Generic functions . . . . .	1061
8.2	Frequency Lists and Distributions . . . . .	1061
8.3	API Reference . . . . .	1062
<b>9</b>	<b>Text Processors</b>	<b>1065</b>
9.1	Tokenisation . . . . .	1065
9.2	N-gram extraction . . . . .	1065
<b>10</b>	<b>Indices and tables</b>	<b>1069</b>
	<b>Python Module Index</b>	<b>1071</b>

PyNLPI, pronounced as ‘pineapple’, is a Python library for Natural Language Processing. It contains various modules useful for common, and less common, NLP tasks. PyNLPI can be used for basic tasks such as the extraction of n-grams and frequency lists, and to build simple language model. There are also more complex data types and algorithms. Moreover, there are parsers for file formats common in NLP (e.g. FoLiA/Giza/Moses/ARPA/Timbl/CQL). There are also clients to interface with various NLP specific servers. PyNLPI most notably features a very extensive library for working with FoLiA XML (Format for Linguistic Annotation).

The library is divided into several packages and modules. It works on Python 2.7, as well as Python 3.

The following modules are available:

- `pynlpl.datatypes` - Extra datatypes (priority queues, patterns, tries)
- `pynlpl.evaluation` - Evaluation & experiment classes (parameter search, wrapped progressive sampling, class evaluation (precision/recall/f-score/auc), sampler, confusion matrix, multithreaded experiment pool)
- `pynlpl.formats.cgn` - Module for parsing CGN (Corpus Gesproken Nederlands) part-of-speech tags
- `pynlpl.formats.folia` - Extensive library for reading and manipulating the documents in [FoLiA](#) format (Format for Linguistic Annotation).
- `pynlpl.formats.fql` - Extensive library for the FoLiA Query Language (FQL), built on top of `pynlpl.formats.folia`. FQL is currently documented [here](#).
- `pynlpl.formats.cql` - Parser for the Corpus Query Language (CQL), as also used by Corpus Workbench and Sketch Engine. Contains a convertor to FQL.
- `pynlpl.formats.giza` - Module for reading GIZA++ word alignment data
- `pynlpl.formats.moses` - Module for reading Moses phrase-translation tables.
- `pynlpl.formats.sonar` - Largely obsolete module for pre-releases of the SoNaR corpus, use `pynlpl.formats.folia` instead.
- `pynlpl.formats.timbl` - Module for reading Timbl output (consider using [python-timbl](#) instead though)
- `pynlpl.lm.lm` - Module for simple language model and reader for ARPA language model data as well (used by SRILM).
- `pynlpl.search` - Various search algorithms (Breadth-first, depth-first, beam-search, hill climbing, A star, various variants of each)
- `pynlpl.statistics` - Frequency lists, Levenshtein, common statistics and information theory functions
- `pynlpl.textprocessors` - Simple tokeniser, n-gram extraction

Contents:



---

## Common Functions

---

`pynlpl.common.Enum(*names)`

`pynlpl.common.b(s)`

`pynlpl.common.isstring(s)`

`pynlpl.common.log(msg, **kwargs)`

Generic log method. Will prepend timestamp.

### Keyword Arguments

- - Name of the system/module (*system*) -
- - Integer denoting the desired level of indentation (*indent*) -
- - List of streams to output to (*streams*) -
- - Stream to output to (*stream*) -

`pynlpl.common.u(s, encoding='utf-8', errors='strict')`





## CHAPTER 2

---

### Data Types

---

This library contains various extra data types, based to a certain extent on MIT-licensed code from Peter Norvig, AI: A Modern Approach : <http://aima.cs.berkeley.edu/python/utils.html>

```
class pynlpl.datatypes.FIFOQueue (data=[])
    A First-In-First-Out Queue

    append (item)

    extend (items)
        Append all elements from items to the queue

    pop ()
        Retrieve the next element in line, this will remove it from the queue

class pynlpl.datatypes.Pattern (data, classdecoder=None)

    static fromstring (s, classencoder)

    iterbytes (begin=0, end=0)

class pynlpl.datatypes.PatternMap (default=None)

    items ()

class pynlpl.datatypes.PatternSet

    add (pattern)

    remove (pattern)

class pynlpl.datatypes.PriorityQueue (data=[], f=<function PriorityQueue.<lambda>>, minimize=False, length=0, blockworse=False, blockequal=False, duplicates=True)
    A queue in which the maximum (or minimum) element is returned first, as determined by either an external score function f (by default calling the objects score() method). If minimize=True, the item with minimum f(x) is returned first; otherwise is the item with maximum f(x) or x.score().
```

length can be set to an integer  $> 0$ . Items will only be added to the queue if they're better or equal to the worst scoring item. If set to zero, length is unbounded. blockworse can be set to true if you want to prohibit adding worse-scoring items to the queue. Only items scoring better than the *BEST* one are added. bloquequal can be set to false if you also want to prohibit adding equally-scoring items to the queue. (Both parameters default to False)

**append** (*item*)

Adds an item to the priority queue (in the right place), returns True if successfull, False if the item was blocked (because of a bad score)

**pop** ()

Retrieve the next element in line, this will remove it from the queue

**prune** (*n*)

prune all but the first (=best) n items

**prunebyscore** (*score*, *retainequalscore=False*)

Deletes all items below/above a certain score from the queue, depending on whether minimize is True or False. Note: It is recommended (more efficient) to use blockworse=True / bloquequal=True instead! Preventing the addition of 'worse' items.

**randomprune** (*n*)

prune down to n items at random, disregarding their score

**score** (*i*)

Return the score for item x (cheap lookup), Item 0 is always the best item

**stochasticprune** (*n*)

prune down to n items, chance of an item being pruned is reverse proportional to its score

**class** pynlpl.datatypes.**Queue**

**Queue is an abstract class/interface. There are three types:** Python List: A Last In First Out Queue (no Queue object necessary). FIFOQueue(): A First In First Out Queue. PriorityQueue(lt): Queue where items are sorted by lt, (default  $<$ ).

**Each type supports the following methods and functions:** q.append(item) – add an item to the queue q.extend(items) – equivalent to: for item in items: q.append(item) q.pop() – return the top item from the queue len(q) – number of items in q (also q.\_\_len()).

**extend** (*items*)

Append all elements from items to the queue

**class** pynlpl.datatypes.**Tree** (*value=None*, *children=None*)

Simple tree structure. Nodes are themselves trees.

**append** (*item*)

Add an item to the Tree

**leaf** ()

Is this a leaf node or not?

**class** pynlpl.datatypes.**Trie** (*sequence=None*)

Simple trie structure. Nodes are themselves tries, values are stored on the edges, not the nodes.

**append** (*sequence*)

**depth** ()

Returns the depth of the current node

**find** (*sequence*)

**items** ()

**leaf()**

Is this a leaf node or not?

**path()**

Returns the path to the current node

**root()**

Returns True if this is the root of the Trie

**sequence()**

**size()**

Size is number of nodes under the trie, including the current node

**walk** (*leavesonly=True, maxdepth=None, \_depth=0*)

Depth-first search, walking through trie, returning all encountered nodes (by default only leaves)



---

Evaluation & Experiments

---

```
class pynlpl.evaluation.AbstractExperiment (inputdata=None, **parameters)

    defaultparameters ()
    delete ()
    done (warn=True)
        Is the subprocess done?
    duration ()
    run ()
    sample (size)
        Return a sample of the input data
    score ()
    start ()
        Start as a detached subprocess, immediately returning execution to caller.
    startcommand (command, cwd, stdout, stderr, *arguments, **parameters)
    wait ()

class pynlpl.evaluation.ClassEvaluation (goals=[], observations=[], missing={},
                                         encoding='utf-8')

    accuracy (cls=None)
    append (goal, observation)
    auc (cls=None, macro=False)
    compute ()
    confusionmatrix (casesensitive=True)
    fp_rate (cls=None, macro=False)
```

```
fscore (cls=None, beta=1, macro=False)
outputmetrics ()
precision (cls=None, macro=False)
recall (cls=None, macro=False)
specificity (cls=None, macro=False)
tp_rate (cls=None, macro=False)
class pynlpl.evaluation.ConfusionMatrix (tokens=None,    casesensitive=True,    dovalida-
                                          tion=True)
    Confusion Matrix
class pynlpl.evaluation.ExperimentPool (size)

    append (experiment)
    poll (haltonerror=True)
    run (haltonerror=True)
    start (experiment)
class pynlpl.evaluation.OrdinalEvaluation (goals=[],    observations=[],    missing={},
                                          encoding='utf-8')

    compute ()
    mae (cls=None)
    rmse (cls=None)
class pynlpl.evaluation.ParamSearch (experimentclass, inputdata, parameterscope, poolsize=1,
                                          constraintfunc=None, delete=True)
    A simpler version of ParamSearch without Wrapped Progressive Sampling
exception pynlpl.evaluation.ProcessFailed
class pynlpl.evaluation.WPSParamSearch (experimentclass, inputdata, size, parameterscope,
                                          poolsize=1, sizefunc=None, prunefunc=None, con-
                                          straintfunc=None, delete=True)
    ParamSearch with support for Wrapped Progressive Sampling
    searchbest ()
    test (i=None)
pynlpl.evaluation.auc (x, y, reorder=False)
    Compute Area Under the Curve (AUC) using the trapezoidal rule

    This is a general fuction, given points on a curve.  For computing the area under the ROC-curve, see
    auc_score().
```

#### Parameters

- **x** (array, shape = [n]) – x coordinates.
- **y** (array, shape = [n]) – y coordinates.
- **reorder** (boolean, optional (default=False)) – If True, assume that the curve is ascending in the case of ties, as for an ROC curve. If the curve is non-ascending, the result will be wrong.

#### Returns auc

**Return type** float

## Examples

```
>>> import numpy as np
>>> from sklearn import metrics
>>> y = np.array([1, 1, 2, 2])
>>> pred = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = metrics.roc_curve(y, pred, pos_label=2)
>>> metrics.auc(fpr, tpr)
0.75
```

## See also:

**auc\_score()** Computes the area under the ROC curve

`pynlpl.evaluation.filesampler` (*files*, *testsetsize=0.1*, *devsetsize=0*, *trainsetsize=0*, *outputdir=""*,  
*encoding='utf-8'*)

Extract a training set, test set and optionally a development set from one file, or multiple *interdependent* files (such as a parallel corpus). It is assumed each line contains one instance (such as a word or sentence for example).

`pynlpl.evaluation.mae` (*absolute\_error\_values*)

`pynlpl.evaluation.rmse` (*squared\_error\_values*)





# CHAPTER 4

---

## FoLiA library

---

This tutorial will introduce the **FoLiA Python library**, part of PyNLPL. The FoLiA library provides an Application Programming Interface for the reading, creation and manipulation of FoLiA XML documents. The library works under Python 2.7 as well as Python 3, which is the recommended version. The samples in this documentation follow Python 3 conventions.

Prior to reading this document, it is recommended to first read the FoLiA documentation itself and familiarise yourself with the format and underlying paradigm. The FoLiA documentation can be found on the [FoLiA website](#). It is especially important to understand the way FoLiA handles sets/classes, declarations, common attributes such as annotator/annotortype and the distinction between various kinds of annotation categories such as token annotation and span annotation.

This Python library is also the foundation of the [FoLiA Tools](#) collection, which consists of various command line utilities to perform common tasks on FoLiA documents. If you're merely interested in performing a certain common task, such as a single query or conversion, you might want to check there if it contains is a tool that does what you want already.

## 4.1 Reading FoLiA

### 4.1.1 Loading a document

Any script that uses FoLiA starts with the import:

```
from pynlpl.formats import folia
```

At the basis of any FoLiA processing lies the following class:

---

*Document*

---

This is the FoLiA Document and holds all its data in memory.

---

## pynlpl.formats.folia.Document

**class** pynlpl.formats.folia.Document (\*args, \*\*kwargs)

Bases: object

This is the FoLiA Document and holds all its data in memory.

All FoLiA elements have to be associated with a FoLiA document. Besides holding elements, the document may hold metadata including declarations, and an index of all IDs.

### Method Summary

<code>__init__(*args, **kwargs)</code>	Start/load a FoLiA document:
<code>add(text)</code>	Alias for <code>Document.append()</code>
<code>alias(annotationtype, set[, fallback])</code>	Return the alias for a set (if applicable, returns the unaltered set otherwise iff fallback is enabled)
<code>append(text)</code>	Add a text (or speech) to the document:
<code>count(Class[, set, recursive, ignore])</code>	See <code>AbstractElement.count()</code>
<code>create(Class, *args, **kwargs)</code>	Create an element associated with this Document.
<code>date([value])</code>	Get or set the document's date from/in the metadata.
<code>declare(annotationtype, set, **kwargs)</code>	Declare a new annotation type to be used in the document.
<code>declared(annotationtype, set)</code>	Checks if the annotation type is present (i.e.
<code>defaultannotator(annotationtype[, set])</code>	Obtain the default annotator for the specified annotation type and set.
<code>defaultannotatortype(annotationtype[, set])</code>	Obtain the default annotator type for the specified annotation type and set.
<code>defaultdatetime(annotationtype[, set])</code>	Obtain the default datetime for the specified annotation type and set.
<code>defaultset(annotationtype)</code>	Obtain the default set for the specified annotation type.
<code>findwords(*args, **kwargs)</code>	
<code>items()</code>	Returns a depth-first flat list of all items in the document
<code>json()</code>	Serialise the document to a dict ready for serialisation to JSON.
<code>jsondeclarations()</code>	Return all declarations in a form ready to be serialised to JSON.
<code>language([value])</code>	No arguments: Get the document's language (ISO-639-3) from metadata Argument: Set the document's language (ISO-639-3) in metadata
<code>license([value])</code>	No arguments: Get the document's license from metadata Argument: Set the document's license in metadata
<code>load(filename)</code>	Load a FoLiA XML file.
<code>paragraphs([index])</code>	Return a generator of all paragraphs found in the document.
<code>parsemetadata(node)</code>	Internal method to parse metadata
<code>parsesubmetadata(node)</code>	
<code>parsexml(node[, ParentClass])</code>	Internal method.
<code>parsexmldeclarations(node)</code>	Internal method to parse XML declarations

Continued on next page

Table 2 – continued from previous page

<code>pendingvalidation([warnonly])</code>	Perform any pending validations
<code>publisher([value])</code>	No arguments: Get the document's publisher from metadata Argument: Set the document's publisher in metadata
<code>save([filename])</code>	Save the document to file.
<code>select(Class[, set, recursive, ignore])</code>	See <code>AbstractElement.select()</code>
<code>sentences([index])</code>	Return a generator of all sentence found in the document.
<code>setimdi(node)</code>	OBSOLETE
<code>text([cls, retaintokenisation])</code>	Returns the text of the entire document (returns a unicode instance)
<code>title([value])</code>	Get or set the document's title from/in the metadata
<code>unalias(annotationtype, alias)</code>	Return the set for an alias (if applicable, raises an exception otherwise)
<code>words([index])</code>	Return a generator of all active words found in the document.
<code>xml()</code>	Serialise the document to XML.
<code>xmldeclarations()</code>	Internal method to generate XML nodes for all declarations
<code>xmlmetadata()</code>	Internal method to serialize metadata to XML
<code>xmlstring()</code>	Return the XML representation of the document as a string.
<code>xpath(query)</code>	Run Xpath expression and parse the resulting elements.

## Attributes

---

IDSEPARATOR

---

## Method Details

`__init__ (*args, **kwargs)`

Start/load a FoLiA document:

There are four sources of input for loading a FoLiA document:

1. Create a new document by specifying an *ID*:

```
doc = folia.Document(id='test')
```

2. Load a document from FoLiA or D-Coi XML file:

```
doc = folia.Document(file='/path/to/doc.xml')
```

3. Load a document from an XML string:

```
doc = folia.Document(string='<FoLiA>...</FoLiA>')
```

4. Load a document by passing a parse xml tree (lxml.etree):

```
doc = folia.Document(tree=xmldata)
```

Additionally, there are three modes that can be set with the `mode=` keyword argument:

- `folia.Mode.MEMORY` - The entire FoLiA Document will be loaded into memory. This is the default mode and the only mode in which documents can be manipulated and saved again.
- `folia.Mode.XPATH` - The full XML tree will still be loaded into memory, but conversion to FoLiA classes occurs only when queried. This mode can be used when the full power of XPath is required.

### Keyword Arguments

- **setdefinition** (*dict*) – A dictionary of set definitions, the key corresponds to the set name, the value is a `SetDefinition` instance
- **loadsetdefinitions** (*bool*) – download and load set definitions (default: `False`)
- **deepvalidation** (*bool*) – Do deep validation of the document (default: `False`), implies `loadsetdefinitions`
- **textvalidation** (*bool*) – Do validation of text consistency (default: `False`)
- **preparsexmlcallback** (*function*) – Callback for a function taking one argument (*node*, an `lxml` node). Will be called whenever an XML element is parsed into FoLiA. The function should return an instance inherited from `folia.AbstractElement`, or `None` to abort parsing this element (and all its children)
- **parsexmlcallback** (*function*) – Callback for a function taking one argument (*element*, a FoLiA element). Will be called whenever an XML element is parsed into FoLiA. The function should return an instance inherited from `folia.AbstractElement`, or `None` to abort adding this element (and all its children)
- **debug** (*bool*) – Boolean to enable/disable debug

`__init__` (\*args, \*\*kwargs)

Start/load a FoLiA document:

There are four sources of input for loading a FoLiA document:

1. Create a new document by specifying an *ID*:

```
doc = folia.Document(id='test')
```

2. Load a document from FoLiA or D-Coi XML file:

```
doc = folia.Document(file='/path/to/doc.xml')
```

3. Load a document from an XML string:

```
doc = folia.Document(string='<FoLiA>...</FoLiA>')
```

4. Load a document by passing a parse xml tree (`lxml.etree`):

```
doc = folia.Document(tree=xmldata)
```

Additionally, there are three modes that can be set with the `mode=` keyword argument:

- `folia.Mode.MEMORY` - The entire FoLiA Document will be loaded into memory. This is the default mode and the only mode in which documents can be manipulated and saved again.
- `folia.Mode.XPATH` - The full XML tree will still be loaded into memory, but conversion to FoLiA classes occurs only when queried. This mode can be used when the full power of XPath is required.

### Keyword Arguments

- **setdefinition** (*dict*) – A dictionary of set definitions, the key corresponds to the set name, the value is a `SetDefinition` instance
- **loadsetdefinitions** (*bool*) – download and load set definitions (default: `False`)
- **deepvalidation** (*bool*) – Do deep validation of the document (default: `False`), implies `loadsetdefinitions`
- **textvalidation** (*bool*) – Do validation of text consistency (default: `False`)
- **preparsexmlcallback** (*function*) – Callback for a function taking one argument (*node*, an `lxml` node). Will be called whenever an XML element is parsed into FoLiA. The function should return an instance inherited from `folia.AbstractElement`, or `None` to abort parsing this element (and all its children)
- **parsexmlcallback** (*function*) – Callback for a function taking one argument (*element*, a FoLiA element). Will be called whenever an XML element is parsed into FoLiA. The function should return an instance inherited from `folia.AbstractElement`, or `None` to abort adding this element (and all its children)
- **debug** (*bool*) – Boolean to enable/disable debug

**add** (*text*)

Alias for `Document.append()`

**alias** (*annotationtype*, *set*, *fallback=False*)

Return the alias for a set (if applicable, returns the unaltered set otherwise iff `fallback` is enabled)

**append** (*text*)

Add a text (or speech) to the document:

Example 1:

```
doc.append(folia.Text)
```

**Example 2::** `doc.append( folia.Text(doc, id='example.text') )`

Example 3:

```
doc.append(folia.Speech)
```

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*)

See `AbstractElement.count()`

**create** (*Class*, *\*args*, *\*\*kwargs*)

Create an element associated with this Document. This method may be obsolete and removed later.

**date** (*value=None*)

Get or set the document's date from/in the metadata.

No arguments: Get the document's date from metadata Argument: Set the document's date in metadata

**declare** (*annotationtype*, *set*, *\*\*kwargs*)

Declare a new annotation type to be used in the document.

Keyword arguments can be used to set defaults for any annotation of this type and set.

#### Parameters

- **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.

- **set** (*str*) – the set, should formally be a URL pointing to the set definition

#### Keyword Arguments

- **annotator** (*str*) – Sets a default annotator
- **annotatortype** – Should be either `AnnotatorType.MANUAL` or `AnnotatorType.AUTO`, indicating whether the annotation was performed manually or by an automated process.
- **datetime** (*datetime.datetime*) – Sets the default datetime
- **alias** (*str*) – Defines alias that may be used in set attribute of elements instead of the full set name

Example:

```
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', annotator=
↪ "mytagger", annotatortype=folia.AnnotatorType.AUTO)
```

**declared** (*annotationtype, set*)

Checks if the annotation type is present (i.e. declared) in the document.

#### Parameters

- **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.
- **set** (*str*) – the set, should formally be a URL pointing to the set definition (aliases are also supported)

Example:

```
if doc.declared(folia.PosAnnotation, 'http://some/path/brown-tag-set') :
    ..
```

**Returns** bool

**defaultannotator** (*annotationtype, set=None*)

Obtain the default annotator for the specified annotation type and set.

#### Parameters

- **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.
- **set** (*str*) – the set, should formally be a URL pointing to the set definition

**Returns** the set (*str*)

**Raises** `NoDefaultError` if the annotation type does not exist or if there is ambiguity (multiple sets for the same type)

**defaultannotatortype** (*annotationtype, set=None*)

Obtain the default annotator type for the specified annotation type and set.

#### Parameters

- **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.

- **set** (*str*) – the set, should formally be a URL pointing to the set definition

**Returns** `AnnotatorType.AUTO` or `AnnotatorType.MANUAL`

**Raises** `NoDefaultError` if the annotation type does not exist or if there is ambiguity (multiple sets for the same type)

**defaultdatetime** (*annotationtype*, *set=None*)

Obtain the default datetime for the specified annotation type and set.

**Parameters**

- **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.
- **set** (*str*) – the set, should formally be a URL pointing to the set definition

**Returns** the set (*str*)

**Raises** `NoDefaultError` if the annotation type does not exist or if there is ambiguity (multiple sets for the same type)

**defaultset** (*annotationtype*)

Obtain the default set for the specified annotation type.

**Parameters** **annotationtype** – The type of annotation, this is conveyed by passing the corresponding annotation class (such as `PosAnnotation` for example), or a member of `AnnotationType`, such as `AnnotationType.POS`.

**Returns** the set (*str*)

**Raises** `NoDefaultError` if the annotation type does not exist or if there is ambiguity (multiple sets for the same type)

**findwords** (*\*args*, *\*\*kwargs*)

**items** ()

Returns a depth-first flat list of all items in the document

**json** ()

Serialise the document to a `dict` ready for serialisation to JSON.

Example:

```
import json
jsondoc = json.dumps(doc.json())
```

**jsondeclarations** ()

Return all declarations in a form ready to be serialised to JSON.

**Returns** list of dict

**language** (*value=None*)

No arguments: Get the document's language (ISO-639-3) from metadata Argument: Set the document's language (ISO-639-3) in metadata

**license** (*value=None*)

No arguments: Get the document's license from metadata Argument: Set the document's license in metadata

**load** (*filename*)

Load a FoLiA XML file.

**Argument:** *filename* (*str*): The file to load

**paragraphs** (*index=None*)

Return a generator of all paragraphs found in the document.

If an index is specified, return the n'th paragraph only (starting at 0)

**parsemetadata** (*node*)

Internal method to parse metadata

**parsesubmetadata** (*node*)

**parsexml** (*node, ParentClass=None*)

Internal method.

This is the main XML parser, will invoke class-specific XML parsers.

**parsexmldeclarations** (*node*)

Internal method to parse XML declarations

**pendingvalidation** (*warnonly=None*)

Perform any pending validations

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**publisher** (*value=None*)

No arguments: Get the document's publisher from metadata Argument: Set the document's publisher in metadata

**save** (*filename=None*)

Save the document to file.

**Parameters** **filename** (\*) – The filename to save to. If not set (None, default), saves to the same file as loaded from.

**select** (*Class, set=None, recursive=True, ignore=True*)

See [AbstractElement.select\(\)](#)

**sentences** (*index=None*)

Return a generator of all sentence found in the document. Except for sentences in quotes.

If an index is specified, return the n'th sentence only (starting at 0)

**setimdi** (*node*)

OBSOLETE

**text** (*cls='current', retaintokenisation=False*)

Returns the text of the entire document (returns a unicode instance)

**See also:**

[AbstractElement.text\(\)](#)

**title** (*value=None*)

Get or set the document's title from/in the metadata

No arguments: Get the document's title from metadata Argument: Set the document's title in metadata

**unalias** (*annotationtype, alias*)

Return the set for an alias (if applicable, raises an exception otherwise)



**words** (*index=None*)

Return a generator of all active words found in the document. Does not descend into annotation layers, alternatives, originals, suggestions.

If an index is specified, return the n'th word only (starting at 0)

**xml** ()

Serialise the document to XML.

**Returns** lxml.etree.Element

**See also:**

*Document.xmlstring()*

**xmldeclarations** ()

Internal method to generate XML nodes for all declarations

**xmlmetadata** ()

Internal method to serialize metadata to XML

**xmlstring** ()

Return the XML representation of the document as a string.

**xpath** (*query*)

Run Xpath expression and parse the resulting elements. Don't forget to use the FoLiA namespace in your expressions, using folia: or the short form f:

To read a document from file, instantiate a document as follows:

```
doc = folia.Document(file="/path/to/document.xml")
```

This returned *Document* instance holds the entire document in memory. Note that for large FoLiA documents this may consume quite some memory! If you happened to already have the document content in a string, you can load as follows:

```
doc = folia.Document(string("<FoLiA ..."))
```

Once you have loaded a document, all data is available for you to read and manipulate as you see fit. We will first illustrate some simple use cases:

To save a document back to the file it was loaded from, we do:

```
doc.save()
```

Or we can specify a specific filename:

```
doc.save("/tmp/document.xml")
```

**Note:** Any content that is in a different XML namespace than the FoLiA namespaces or other supported namespaces (XML, Xlink), will be ignored upon loading and lost when saving.

## 4.1.2 Printing text

You may want to simply print all (plain) text contained in the document, which is as easy as:

```
print(doc)
```

Obtaining the text as a string is done by invoking the document's `Document.text()` method:

```
text = doc.text()
```

Or alternatively as follows:

```
text = str(doc)
```

For any subelement of the document, you can obtain its text in the same fashion as well, by calling its `AbstractElement.text()` method or by using `str()`, the only difference is that the former allows for extensive fine tuning using various extra parameters (See `AbstractElement.text()`).

---

**Note:** In Python 2, both `str()` as well as `unicode()` return a unicode instance. You may need to append `.encode('utf-8')` for proper output.

---

### 4.1.3 Index

A document instance has an **index** which you can use to grab any of its elements by ID. Querying using the index proceeds similar to using a python dictionary:

```
word = doc['example.p.3.s.5.w.1']
print(word)
```

---

**Note:** Python 2 users will have to do `print word.text().encode('utf-8')` instead, to ensure non-ascii characters are printed properly.

---

IDs are unique in the entire document, and preferably even beyond.

### 4.1.4 Elements

All FoLiA elements are derived from `AbstractElement` and offer an identical interface. To quickly check whether you are dealing with a FoLiA element you can therefore always do the following:

```
isinstance(word, folia.AbstractElement)
```

This abstract base element is never instantiated directly. The FoLiA paradigm derives several more abstract base classes which may implement some additional methods or overload some of the original ones:

<code>AbstractElement</code>	Abstract base class from which all FoLiA elements are derived.
<code>AbstractStructureElement</code>	Abstract element, all structure elements inherit from this class.
<code>AllowTokenAnnotation</code>	Elements that allow token annotation (including extended annotation) must inherit from this class
<code>AbstractSpanAnnotation</code>	Abstract element, all span annotation elements are derived from this class
<code>AbstractTokenAnnotation</code>	Abstract element, all token annotation elements are derived from this class

Continued on next page

Table 4 – continued from previous page

<i>AbstractAnnotationLayer</i>	Annotation layers for Span Annotation are derived from this abstract base class
<i>AbstractTextMarkup</i>	Abstract class for text markup elements, elements that appear with the <i>TextContent</i> (t) element.

## pynlpl.formats.folia.AbstractElement

**class** pynlpl.formats.folia.**AbstractElement** (*doc*, \*args, \*\*kwargs)

Bases: object

Abstract base class from which all FoLiA elements are derived.

This class implements many generic methods that are available on all FoLiA elements.

To see if an element is a FoLiA element, as opposed to any other python object, do:

```
isinstance(x, AbstractElement)
```

Generic FoLiA attributes can be accessed on all instances derived from this class:

- `element.id` (str) - The unique identifier of the element
- `element.set` (str) - The set the element pertains to.
- `element.cls` (str) - The assigned class, i.e. the actual value of the annotation, defined in the set. Classes correspond with tagsets in this case of many annotation types. Note that since *class* is already a reserved keyword in python, the library consistently uses *cls* everywhere.
- `element.annotator` (str) - The name or ID of the annotator who added/modified this element
- `element.annotatortype` - The type of annotator, can be either `folia.AnnotatorType.MANUAL` or `folia.AnnotatorType.AUTO`
- `element.confidence` (float) - A confidence value expressing
- `element.datetime` (datetime.datetime) - The date and time when the element was added/modified.
- `element.n` (str) - An ordinal label, used for instance in enumerated list contexts, numbered sections, etc..

The following generic attributes are specific to a speech context:

- `element.src` (str) - A URL or filename referring the an audio or video file containing the speech. Access this attribute using the `element.speaker_src()` method, as it is inheritable from ancestors.
- `element.speaker` (str) - The name or ID of the speaker. Access this attribute using the `element.speech_speaker()` method, as it is inheritable from ancestors.
- `element.begintime` (4-tuple) - The time in the above source fragment when the phonetic content of this element starts, this is a (hours, minutes, seconds, milliseconds) tuple.
- `element.endtime` (4-tuple) - The time in the above source fragment when the phonetic content of this element ends, this is a (hours, minutes, seconds, milliseconds) tuple.

Not all attributes are allowed, unset or unavailable attributes will always default to `None`.

---

**Note:** This class should never be instantiated directly, as it is abstract!

---

See also:

*AbstractElement.\_\_init\_\_()*

## Method Summary

<i>__init__(doc, *args, **kwargs)</i>	Initialize self.
<i>accepts(Class[, raiseexceptions, parentinstance])</i>	
<i>add(child, *args, **kwargs)</i>	
<i>addable(parent[, set, raiseexceptions])</i>	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix(idsuffix[, recursive])</i>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex([norecurse])</i>	Makes sure this element (and all subelements), are properly added to the index.
<i>ancestor(*Classes)</i>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors([Class])</i>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>append(child, *args, **kwargs)</i>	
<i>context(size[, placeholder, scope])</i>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy([newdoc, idsuffix])</i>	Make a deep copy of this element and all its children.
<i>copychildren([newdoc, idsuffix])</i>	Generator creating a deep copy of the children of this element.
<i>count(Class[, set, recursive, ignore, node])</i>	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<i>deepvalidation()</i>	Perform deep validation of this element.
<i>description()</i>	Obtain the description associated with the element.
<i>feat(subset)</i>	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling(cls)</i>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables(parent[, set])</i>	Internal method to find replaceable elements.
<i>getindex(child[, recursive, ignore])</i>	Get the index at which an element occurs, recursive by default!
<i>getmetadata([key])</i>	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter([retaintokenisation])</i>	Return the text delimiter for this class.
<i>hasphon([cls, strict, correctionhandling])</i>	Does this element have phonetic content (of the specified class)
<i>hastext([cls, strict, correctionhandling])</i>	Does this element have text (of the specified class)
<i>incorrection()</i>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert(index, child, *args, **kwargs)</i>	
<i>items([founditems])</i>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json([attribs, recurse, ignorelist])</i>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext(size[, placeholder, scope])</i>	Returns the left context for an element, as a list.

Continued on next page

Table 5 – continued from previous page

<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

## Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None
```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** `ValueError`

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (`AbstractElement` or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (`AbstractElement` or subclasses). Not instances!

**Yields** elements (instances derived from `AbstractElement`)

**append** (*child*, \**args*, \*\**kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hasstext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters



- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – `XML Element` (\*) –
- **doc** – `Document` (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for *text()* with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.AbstractStructureElement**

**class** `pynlpl.formats.folia.AbstractStructureElement` (*doc, \*args, \*\*kwargs*)

**Bases:** `pynlpl.formats.folia.AbstractElement`, `pynlpl.formats.folia.AllowTokenAnnotation`, `pynlpl.formats.folia.AllowGenerateID`

Abstract element, all structure elements inherit from this class. Never instantiated directly.

### **Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)

Continued on next page

Table 6 – continued from previous page

<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)

Continued on next page



Table 6 – continued from previous page

<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = None

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hasstext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attrs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.



- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.AllowTokenAnnotation

**class** `pynlpl.formats.folia.AllowTokenAnnotation`

Bases: `pynlpl.formats.folia.AllowCorrections`

Elements that allow token annotation (including extended annotation) must inherit from this class

## Method Summary

<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>__str__</code>	Return str(self).

## Method Details

### `__init__()`

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

### `alternatives (Class=None, set=None)`

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

### `annotation (type, set=None)`

Obtain a single annotation element.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** `NoSuchAnnotation` if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

`AbstractElement.select()`

**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations()` for a description of the parameters.

**\_\_str\_\_** ()

Return `str(self)`.

## `pynlpl.formats.folia.AbstractSpanAnnotation`

**class** `pynlpl.formats.folia.AbstractSpanAnnotation` (*doc*, *\*args*, *\*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`, `pynlpl.formats.folia`

`AllowGenerateID`, `pynlpl.formats.folia.AllowCorrections`

Abstract element, all span annotation elements are derived from this class

### Method Summary

<code>__init__</code> ( <i>doc</i> , <i>*args</i> , <i>**kwargs</i> )	Initialize self.
<code>accepts</code> ( <i>Class</i> [], <i>raiseexceptions</i> , <i>parentinstance</i> )	
<code>add</code> ( <i>child</i> , <i>*args</i> , <i>**kwargs</i> )	

Continued on next page

Table 8 – continued from previous page

<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.

Continued on next page

Table 8 – continued from previous page

<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 8 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
add(child, *args, **kwargs)

classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.
```

#### Parameters



- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.

- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If `idsuffix` is a string, if set to `True`, a random `idsuffix` will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan(\*args)**

Sets the span of the span element anew, erases all data inside.

**Parameters** *\*args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iff you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be `Words` but also `Morphemes` or `Phonemes`.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.



**Returns** a string with XML representation for this element and all its children

**Return type** str

`__iter__()`

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.AbstractTokenAnnotation

**class** pynlpl.formats.folia.**AbstractTokenAnnotation** (*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractElement`, `pynlpl.formats.folia.AllowGenerateID`

Abstract element, all token annotation elements are derived from this class

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.

Continued on next page

Table 9 – continued from previous page

<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>generate_id</i> (cls)	
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.

Continued on next page

Table 9 – continued from previous page

<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False

```

**XMLTAG = None**

## Method Details

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)

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

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)

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

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** `bool`

**Raises** `ValueError`

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter=""*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`,

which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.



**classmethod** `relaxng` (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** `doc` (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retain\_tokenisation*=False, *previous\_delimiter*="", *strict*=False, *correction\_handling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retain\_tokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previous\_delimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correction\_handling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text* ()

## pynlpl.formats.folia.AbstractAnnotationLayer

**class** pynlpl.formats.folia.**AbstractAnnotationLayer** (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractElement*, *pynlpl.formats.folia.*

*AllowGenerateID*, *pynlpl.formats.folia.AllowCorrections*

Annotation layers for Span Annotation are derived from this abstract base class

### Method Summary

<i>__init__</i> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<i>accepts</i> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<i>add</i> ( <i>child, *args, **kwargs</i> )	
<i>addable</i> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> ( <i>idsuffix[, recursive]</i> )	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ( <i>[norecurse]</i> )	Makes sure this element (and all subelements), are properly added to the index.
<i>alternatives</i> ( <i>[Class, set]</i> )	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<i>ancestor</i> ( <i>*Classes</i> )	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ( <i>[Class]</i> )	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>annotation</i> ( <i>type[, set]</i> )	Will return a <b>single</b> annotation (even if there are multiple).
<i>annotations</i> ( <i>Class[, set]</i> )	Obtain annotations.
<i>append</i> ( <i>child, *args, **kwargs</i> )	See <i>AbstractElement.append()</i>
<i>context</i> ( <i>size[, placeholder, scope]</i> )	Returns this word in context, {size} words to the left, the current word, and {size} words to the right

Continued on next page

Table 10 – continued from previous page

<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.

Continued on next page

Table 10 – continued from previous page

<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)

```

```

PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \*Classes – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (Class=None)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \*Class – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (type, set=None)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (Class, set=None)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (child, \*args, \*\*kwargs)

See *AbstractElement.append()*

**context** (size, placeholder=None, scope=None)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (newdoc=None, idsuffix="")

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (newdoc=None, idsuffix="")

Generator creating a deep copy of the children of this element.



Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** DeepValidationError

**description** ()

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchText – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** NoSuchText if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.AbstractTextMarkup

**class** pynlpl.formats.folia.**AbstractTextMarkup**(*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractElement`

Abstract class for text markup elements, elements that appear with the `TextContent` (t) element.

Markup elements pertain primarily to styling, but also have other roles.

Iterating over the element of a `TextContent` element will first and foremost produce strings, but also uncover these markup elements when present.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	See <code>AbstractElement.__init__()</code> , text is passed as a string in <code>*args</code> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

Continued on next page



Table 11 – continued from previous page

<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	See <i>AbstractElement.json()</i>
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolve</i> ()	
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text)	Sets the text content of the markup element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page

Table 11 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = True
TEXTDELIMITER = ''
XLINK = True
XMLTAG = None
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

See `AbstractElement.__init__()`, text is passed as a string in \**args*.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

See `AbstractElement.__init__()`, text is passed as a string in \**args*.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, \**args*, \*\**kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

See `AbstractElement.json()`

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling*=*CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolve()**

**resolveword**(*id*)

**rightcontext**(*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select**(*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc**(*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument**(*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents**()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext**(*text*)

Sets the text content of the markup element.

**Parameters** **text** (*str*) –

**speech\_speaker**()

Retrieves the speaker of the audio or video file associated with the element.



The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iff you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

### 4.1.5 Obtaining list of elements

The aforementioned index is useful only if you know the ID of the element. This is often not the case, and you will want to iterate through the hierarchy of elements through different means.

If you want to iterate over all of the child elements of a certain element, regardless of what type they are, you can simply do so as follows:

```

for subelement in element:
    if isinstance(subelement, folia.Sentence):
        print("this is a sentence")
    else:
        print("this is something else")

```

If applied recursively this allows you to traverse the entire element tree, there are however specialised methods available that do this for you.

### 4.1.6 Select method

There is a generic method `AbstractElement.select()` available on all elements to select child elements of any desired class. This method is by default applied recursively for most element types:

```

sentence = doc['example.p.3.s.5.w.1']
words = sentence.select(folia.Word)
for word in words:
    print(word)

```

The `AbstractElement.select()` method has a sibling `AbstractElement.count()`, invoked with the same arguments, which simply counts how many items it finds, without actually returning them:

```

word = sentence.count(folia.Word)

```

**Note:** The `select()` method and similar high-level methods derived from it, are generators. This implies that the results of the selection are returned one by one in the iteration, as opposed to all stored in memory. This also implies that you can only iterate over it once, we can not do another iteration over the `words` variable in the above example, unless we reinvoke the `select()` method to get a new generator. Likewise, we can not do `len(words)`, but have to use the `count()` method instead.

If you want to have all results in memory in a list, you can simply do the following:

```

words = list(sentence.select(folia.Word))

```

The `select` method is by default recursive, set the third argument to `False` to make it non-recursive. The second argument can be used for restricting matches to a specific set, a tuple of classes. The recursion will not go into any *non-authoritative* elements such as alternatives, originals of corrections.

### 4.1.7 Selection Shortcuts

There are various shortcut methods for `select()`.

For example, you can iterate over all words in the document using `Document.words()`, or all words under any structural element using `AbstractStructureElement.words()`:

```

for word in doc.words():
    print(word)

```

That however gives you one big iteration of words without boundaries. You may more likely want to seek words within sentences, provided the document distinguishes sentences. So we first iterate over all sentences using `Document.sentences()` and then over the words therein using `AbstractStructureElement.words()`:

```
for sentence in doc.sentences():
    for word in sentence.words():
        print(word)
```

Or including paragraphs, assuming the document has them:

```
for paragraph in doc.paragraphs():
    for sentence in paragraph.sentences():
        for word in sentence.words():
            print(word)
```

**Warning:** Do be aware that such constructions make presumptions about the structure of the FoLiA document that may not always apply!

All of these shortcut methods also take an `index` parameter to quickly select a specific item in the sequence:

```
word = sentence.words(3) #retrieves the fourth word
```

### 4.1.8 Navigating a document

The `AbstractElement.select()` method is your main tool for descending downwards in the document tree. There are occasions, however, when you want go upwards or sideways. The `AbstractElement.next()` and `AbstractElement.previous()` methods can be used for sideways navigation, they return the next or previous element, respectively:

```
nextelement = element.next()
previouselement = element.previous()
```

You can explicitly filter by passing an element type:

```
nextword = word.next(folia.Word)
```

By default, the search is constrained not to cross certain boundaries, such as sentences and paragraphs. You can do so explicitly as well by passing a list of constraints:

```
nextword = word.next(folia.Word, [folia.Sentence])
```

If you do not want any constraints, pass `None`:

```
nextword = word.next(folia.Word, None)
```

These methods will return `None` if no next/previous element was found (of the specified type).

Each element has a `parent` attribute that links it to its parent:

```
sentence = word.parent
```

Only for the top-level element (`Text` or `Speech`), the parent is `None`. There is also the method `AbstractElement.ancestors()` to iterate over all ancestors, ordered from most immediate to most distant ancestor:

```
for ancestor in element.ancestors():
    print(type(ancestor))
```

If you are looking for ancestors of a specific type, you can pass it as an argument:

```
for ancestor in element.ancestors(folia.Division):
    print(type(ancestor))
```

If only a single ancestor is desired, use the `AbstractElement.ancestor()` method instead, unlike the generator version `AbstractElement.ancestors()`, it will raise a `NoSuchAnnotation` exception if the ancestor was not found:

```
paragraph = word.ancestor(folia.Paragraph)
```

## 4.1.9 Structure Annotation Types

The FoLiA library discerns various Python classes for structure annotation, all are subclasses of `AbstractStructureElement`, which in turn is a subclass of `AbstractElement`. We list the classes for structure annotation along with the FoLiA XML tag. Sets and classes can be associated with most of these elements to make them more specific, these are never prescribed by FoLiA. The list of classes is as follows:

<i>Cell</i>	A cell in a <i>Row</i> in a <i>Table</i>
<i>Definition</i>	Element used in <i>Entry</i> for the portion that provides a definition for the entry.
<i>Division</i>	Structure element representing some kind of division.
<i>Entry</i>	Represents an entry in a glossary/lexicon/dictionary.
<i>Event</i>	Structural element representing events, often used in new media contexts for things such as tweets, chat messages and forum posts.
<i>Example</i>	Element that provides an example.
<i>Figure</i>	Element for the representation of a graphical figure.
<i>Gap</i>	Gap element, represents skipped portions of the text.
<i>Head</i>	Head element; a structure element that acts as the header/title of a <i>Division</i> .
<i>Linebreak</i>	Line break element, signals a line break.
<i>List</i>	Element for enumeration/itemisation.
<i>ListItem</i>	Single element in a List.
<i>Note</i>	Element used for notes, such as footnotes or warnings or notice blocks.
<i>Paragraph</i>	Paragraph element.
<i>Part</i>	Generic structure element used to mark a part inside another block.
<i>Quote</i>	Quote: a structure element.
<i>Reference</i>	A structural element that denotes a reference, internal or external.
<i>Row</i>	A row in a <i>Table</i>
<i>Sentence</i>	Sentence element.
<i>Table</i>	A table consisting of <i>Row</i> elements that in turn consist of <i>Cell</i> elements
<i>Term</i>	A term, often used in context of <i>Entry</i>
<i>TableHead</i>	Encapsulated the header of a table, contains <i>Cell</i> elements
<i>Text</i>	A full text.
<i>Whitespace</i>	Whitespace element, signals a vertical whitespace

Continued on next page

Table 12 – continued from previous page

<i>Word</i>	Word (aka token) element.
<b>pynlpl.formats.folia.Cell</b>	
<b>class</b> pynlpl.formats.folia. <b>Cell</b> ( <i>doc</i> , *args, **kwargs)	
Bases: <i>pynlpl.formats.folia.AbstractStructureElement</i>	
A cell in a <i>Row</i> in a <i>Table</i>	
<b>Method Summary</b>	
<i>__init__</i> (doc, *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> (idsuffix[, recursive])	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<i>addtoindex</i> ([norecurse])	Makes sure this element (and all subelements), are properly added to the index.
<i>alternatives</i> ([Class, set])	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<i>ancestor</i> (*Classes)	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ([Class])	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>annotation</i> (type[, set])	Obtain a single annotation element.
<i>annotations</i> (Class[, set])	Obtain child elements (annotations) of the specified class.
<i>append</i> (child, *args, **kwargs)	See <i>AbstractElement.append()</i>
<i>context</i> (size[, placeholder, scope])	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy</i> ([newdoc, idsuffix])	Make a deep copy of this element and all its children.
<i>copychildren</i> ([newdoc, idsuffix])	Generator creating a deep copy of the children of this element.
<i>correct</i> (**kwargs)	Apply a correction (TODO: documentation to be written still)
<i>count</i> (Class[, set, recursive, ignore, node])	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<i>deepvalidation</i> ()	Perform deep validation of this element.
<i>description</i> ()	Obtain the description associated with the element.
<i>feat</i> (subset)	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>generate_id</i> (cls)	

Continued on next page

Table 13 – continued from previous page

<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.

Continued on next page

Table 13 – continued from previous page

<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Cell'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False

```



```

SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' | '
XLINK = False
XMLTAG = 'cell'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to `None` to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \*Classes – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (Class=None)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \*Class – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (type, set=None)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (class) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (str) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (Class, set=None)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (class) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (str) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**`AbstractElement.select()`**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)See `AbstractElement.append()`**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.**Returns** int**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.

- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype*=None, *set*=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (*index*=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int* or *None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod** **parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if

you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found



**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if

you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

*text()* *phoncontent()* *phon()*

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for *text()* with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words** (*index=None*)

Returns a generator of `Word` elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

*AbstractElement.xmlstring()* - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text()*

**pynlpl.formats.folia.Definition**

**class** pynlpl.formats.folia.**Definition** (*doc, \*args, \*\*kwargs*)  
 Bases: *pynlpl.formats.folia.AbstractStructureElement*

Element used in *Entry* for the portion that provides a definition for the entry.

**Method Summary**

<i>__init__</i> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<i>accepts</i> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<i>add</i> ( <i>child, *args, **kwargs</i> )	
<i>addable</i> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> ( <i>idsuffix[, recursive]</i> )	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ( <i>[norecurse]</i> )	Makes sure this element (and all subelements), are properly added to the index.
<i>alternatives</i> ( <i>[Class, set]</i> )	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<i>ancestor</i> ( <i>*Classes</i> )	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ( <i>[Class]</i> )	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>annotation</i> ( <i>type[, set]</i> )	Obtain a single annotation element.
<i>annotations</i> ( <i>Class[, set]</i> )	Obtain child elements (annotations) of the specified class.
<i>append</i> ( <i>child, *args, **kwargs</i> )	See <i>AbstractElement.append()</i>
<i>context</i> ( <i>size[, placeholder, scope]</i> )	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy</i> ( <i>[newdoc, idsuffix]</i> )	Make a deep copy of this element and all its children.
<i>copychildren</i> ( <i>[newdoc, idsuffix]</i> )	Generator creating a deep copy of the children of this element.
<i>correct</i> ( <i>**kwargs</i> )	Apply a correction (TODO: documentation to be written still)
<i>count</i> ( <i>Class[, set, recursive, ignore, node]</i> )	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<i>deepvalidation</i> ()	Perform deep validation of this element.
<i>description</i> ()	Obtain the description associated with the element.
<i>feat</i> ( <i>subset</i> )	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> ( <i>cls</i> )	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> ( <i>parent[, set]</i> )	Internal method to find replaceable elements.
<i>generate_id</i> ( <i>cls</i> )	
<i>getindex</i> ( <i>child[, recursive, ignore]</i> )	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ( <i>[key]</i> )	Get the metadata that applies to this element, automatically inherited from parent elements

Continued on next page

Table 14 – continued from previous page

<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.

Continued on next page

Table 14 – continued from previous page

<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 39
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Definition'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False

```

```
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'def'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)  
Makes sure this element (and all subelements), are properly added to the index.  
Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)  
Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. *PosAnnotation*). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused



**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you

want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –

- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Division

**class** `pynlpl.formats.folia.Division` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Structure element representing some kind of division. Divisions may be nested at will, and may include almost all kinds of other structure elements.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page



Table 15 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>head</i> ()	
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 15 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 2
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Division'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n\n'
```

```
XLINK = False
XMLTAG = 'div'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects regardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*

- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**head()**

**incorrection()**

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert(index, child, \*args, \*\*kwargs)**

**items(founditems=[])**

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json(attrs=None, recurse=True, ignorelist=False)**

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers(annotationtype=None, set=None)**

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext(size, placeholder=None, scope=None)**

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next(Class=True, scope=True, reverse=False)**

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class (\*)** – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope (\*)** – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext(cls='original')**

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs(index=None)**

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters index (int or None)** – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml(node, doc, \*\*kwargs)**

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node - XML Element (\*)** –
- **doc - Document (\*)** –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element



See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence, Paragraph, Division, Event, ListItem, Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Entry**

**class** `pynlpl.formats.folia.Entry` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Represents an entry in a glossary/lexicon/dictionary.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page

Table 16 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page

Table 16 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 37
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Entry'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'entry'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```



**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** `int`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** `str` or `list`

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype*=None, *set*=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (*index*=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int* or *None*) – If set to an integer, will retrieve and return the *n*’th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element



See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

See also:

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Event

**class** `pynlpl.formats.folia.Event` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Structural element representing events, often used in new media contexts for things such as tweets, chat messages and forum posts.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 17 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 17 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 21
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Event'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
```

```
XLINK = False
XMLTAG = 'event'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*

- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such an annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool



**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence, Paragraph, Division, Event, ListItem, Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Example**

**class** `pynlpl.formats.folia.Example` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Element that provides an example. Used for instance in the context of *Entry*

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page

Table 18 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page



Table 18 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 40
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Example'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'ex'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.



- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Figure

**class** `pynlpl.formats.folia.Figure` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Element for the representation of a graphical figure. Structure element.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>caption()</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 19 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 19 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 5
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Figure'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
```

```
XLINK = False
XMLTAG = 'figure'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*



- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**caption** ()

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you

want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype*=None, *set*=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (*index*=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int* or *None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod** **parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –

- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Gap**

**class** `pynlpl.formats.folia.Gap` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`



Gap element, represents skipped portions of the text.

Usually contains `Content` and possibly also a `Description` element

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>content()</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)

Continued on next page

Table 20 – continued from previous page

<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 20 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 24
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Gap'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 5, 8, 6, 7, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'gap'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.

```

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool***Raises** *ValueError***addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)**content** ()**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to

`CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*



**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the

corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Head

**class** pynlpl.formats.folia.**Head**(doc, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractStructureElement*

Head element; a structure element that acts as the header/title of a *Division*.

There may be only one per division. Often contains sentences (*Sentence*) or Words (*Word*).

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!

Continued on next page

Table 21 – continued from previous page

<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.

Continued on next page

Table 21 – continued from previous page

<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Head'
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True

```

```

SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'head'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)  
Makes sure this element (and all subelements), are properly added to the index.  
Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)  
Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\*Classes)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters \*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters \*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):  
    ..
```

**See also:**

*AbstractElement.select()*



**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you

want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –

- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Linebreak**

**class** `pynlpl.formats.folia.Linebreak` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`, `pynlpl.formats.folia.`



*AbstractTextMarkup*

Line break element, signals a line break.

This element acts both as a structure element as well as a text markup element.

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	See <code>AbstractElement.__init__()</code> , text is passed as a string in <code>*args</code> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements

Continued on next page

Table 22 – continued from previous page

<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	See <code>AbstractElement.json()</code>
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve()</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text)</code>	Sets the text content of the markup element.

Continued on next page

Table 22 – continued from previous page

<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 7
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Linebreak'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = True

```

```
TEXTDELIMITER = ''  
XLINK = True  
XMLTAG = 'br'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
See [AbstractElement.\\_\\_init\\_\\_\(\)](#), text is passed as a string in \**args*.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
See [AbstractElement.\\_\\_init\\_\\_\(\)](#), text is passed as a string in \**args*.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** ([AbstractElement](#)) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by [copy\(\)](#)

**addtoindex** (*norecurse=[]*)  
Makes sure this element (and all subelements), are properly added to the index.  
Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)  
Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by *set*.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or *set* to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of *set*)

**Yields** [Alternative](#) elements

**ancestor** (\**Classes*)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes ([AbstractElement](#) or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you

want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

See `AbstractElement.json()`

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (cls='current', previousdelimiter=",", strict=False, correctionhandling=1)

Get the phonetic representation associated with this element (of the specified class)



The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolve()****resolveword** (*id*)**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.

- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*)

Sets the text content of the markup element.

**Parameters** **text** (*str*) –

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**text** (*cls='current', retaintokenisation=False, previousdelimiter=",", strict=False, correctionhandling=None, normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.List

**class** `pynlpl.formats.folia.List` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Element for enumeration/itemisation. Structure element. Contains *ListItem* elements.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)

Continued on next page

Table 23 – continued from previous page

<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)

Continued on next page

Table 23 – continued from previous page

<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 4
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'List'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'list'
```



## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.



**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

See also:

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.ListItem**

**class** `pynlpl.formats.folia.ListItem` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Single element in a List. Structure element. Contained within *List* element.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page

Table 24 – continued from previous page

<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page

Table 24 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'List Item'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n'
XLINK = False
XMLTAG = 'item'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)



**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Note

**class** `pynlpl.formats.folia.Note` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Element used for notes, such as footnotes or warnings or notice blocks.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page



Table 25 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page

Table 25 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 25
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Note'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'note'
```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element



See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Paragraph**

**class** `pynlpl.formats.folia.Paragraph` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Paragraph element. A structure element. Represents a paragraph and holds all its sentences (and possibly other structure Whitespace and Quotes).

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 26 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 26 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 3
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Paragraph'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'

```

**XLINK = False**

**XMLTAG = 'p'**

## Method Details

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)

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

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)

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

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to `None` to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:



```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*

- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element



**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Part

**class** `pynlpl.formats.folia.Part` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Generic structure element used to mark a part inside another block.

Do **not** use this for morphology, use `Morpheme` instead.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 27 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 27 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 35
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Part'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
```

```
XLINK = False
XMLTAG = 'part'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*

- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool



**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia Quote**

**class** `pynlpl.formats.folia.Quote` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Quote: a structure element. For quotes/citations. May hold *Word*, *Sentence* or *Paragraph* data.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page

Table 28 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page



Table 28 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Quote'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'quote'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.



- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

See also:

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Reference

**class** `pynlpl.formats.folia.Reference` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

A structural element that denotes a reference, internal or external. Examples are references to footnotes, bibliographies, hyperlinks.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 29 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolve</i> ()	
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 29 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Reference'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None

```

```
XLINK = False
XMLTAG = 'ref'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*



- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such an annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolve()**

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Row**

**class** `pynlpl.formats.folia.Row` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`



A row in a *Table*

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?

Continued on next page

Table 30 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>sentences</i> ([index])	Returns a generator of Sentence elements found (recursively) under this element
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.

Continued on next page

Table 30 – continued from previous page

<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Table Row'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n'
XLINK = False
XMLTAG = 'row'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (annotationtype=None, set=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (index=None)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (int or None) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.



**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Sentence

**class** `pynlpl.formats.folia.Sentence` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Sentence element. A structure element. Represents a sentence and holds all its words (*Word*), and possibly other structure such as *LineBreak*, *Whitespace* and *Quote*

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Example.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>corrections()</code>	Are there corrections in this sentence?
<code>correctwords(originalwords, newwords, **kwargs)</code>	Generic correction method for words.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>deleteword(word, **kwargs)</code>	TODO: Write documentation
<code>description()</code>	Obtain the description associated with the element.
<code>division()</code>	Obtain the division this sentence is a part of (None otherwise).
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	

Continued on next page

Table 31 – continued from previous page

<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>insertword(newword, prevword, **kwargs)</code>	Inserts a word <b>as a correction</b> after an existing word.
<code>insertwordleft(newword, nextword, **kwargs)</code>	Inserts a word <b>as a correction</b> before an existing word.
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>mergewords(newword, *originalwords, **kwargs)</code>	TODO: Write documentation
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraph()</code>	Obtain the paragraph this sentence is a part of (None otherwise).
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element

Continued on next page

Table 31 – continued from previous page

<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>splitword(originalword, *newwords, **kwargs)</code>	TODO: Write documentation
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 8
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Sentence'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0

```



```

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' '
XLINK = False
XMLTAG = 's'

```

## Method Details

`__init__(doc, *args, **kwargs)`

Example:

```

sentence = paragraph.append( folia.Sentence)

sentence.append( folia.Word, 'This')
sentence.append( folia.Word, 'is')
sentence.append( folia.Word, 'a')
sentence.append( folia.Word, 'test', space=False)
sentence.append( folia.Word, '.')

```

Example:

```

sentence = folia.Sentence( doc, folia.Word(doc, 'This'), folia.Word(doc, 'is
↪'), folia.Word(doc, 'a'), folia.Word(doc, 'test', space=False), folia.
↪Word(doc, '.') )
paragraph.append(sentence)

```

See also:

`AbstractElement.__init__()`

`__init__(doc, *args, **kwargs)`

Example:

```

sentence = paragraph.append( folia.Sentence)

sentence.append( folia.Word, 'This')
sentence.append( folia.Word, 'is')
sentence.append( folia.Word, 'a')
sentence.append( folia.Word, 'test', space=False)
sentence.append( folia.Word, '.')

```

Example:

```
sentence = folia.Sentence( doc, folia.Word(doc, 'This'), folia.Word(doc, 'is  
↪'), folia.Word(doc, 'a'), folia.Word(doc, 'test', space=False), folia.  
↪Word(doc, '.') )  
paragraph.append(sentence)
```

**See also:**

`AbstractElement.__init__()`

**classmethod** `accepts` (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, *\*args*, *\*\*kwargs*)

**classmethod** `addable` (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. *PosAnnotation*). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**corrections** ()

Are there corrections in this sentence?

**Returns** bool

**correctwords** (*originalwords*, *newwords*, \*\*kwargs)

Generic correction method for words. You most likely want to use the helper functions `Sentence.splitword()`, `Sentence.mergewords()`, `deleteword()`, `insertword()` instead

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**deleteword** (*word*, \*\*kwargs)

TODO: Write documentation

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**division** ()

Obtain the division this sentence is a part of (None otherwise). Shortcut for `AbstractElement.ancestor()`

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)**insertword** (*newword*, *prevword*, *\*\*kwargs*)

Inserts a word **as a correction** after an existing word.

This method automatically computes the index of insertion and calls `AbstractElement.insert()`

**Parameters**

- **newword** (*Word*) – The new word to insert
- **prevword** (*Word*) – The word to insert after

**Keyword Arguments** **suggest** (*bool*) – Do a suggestion for correction rather than the default authoritative correction

**See also:**

`AbstractElement.insert()` and `AbstractElement.getindex()` If you do not want to do corrections

**insertwordleft** (*newword*, *nextword*, *\*\*kwargs*)

Inserts a word **as a correction** before an existing word.

Reverse of `Sentence.insertword()`.

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None*, *recurse=True*, *ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`**layers** (*annotationtype=None*, *set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**mergewords** (*newword*, *\*originalwords*, *\*\*kwargs*)

TODO: Write documentation

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraph** ()

Obtain the paragraph this sentence is a part of (None otherwise). Shortcut for `AbstractElement.ancestor()`

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.

- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all



- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod** `relaxng` (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**splitword** (*originalword, \*newwords, \*\*kwargs*)

TODO: Write documentation

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**text** (*cls='current', retaintokenisation=False, previousdelimiter=", strict=False, correctionhandling=1, normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

## **pynlpl.formats.folia.Table**

**class** `pynlpl.formats.folia.Table` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

A table consisting of `Row` elements that in turn consist of `Cell` elements

### **Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.

Continued on next page

Table 32 – continued from previous page

<code>addtoindex([norecure])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Continued on next page

Table 32 – continued from previous page

<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 32 – continued from previous page

<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 33
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Table'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'table'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts (Class, raiseexceptions=True, parentinstance=None)

```

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.



**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.

- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype=None, set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*’th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```

for sense in text.select(folia.Sense, 'corretto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..

```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**text** (*cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`



**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words (*index=None*)**

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml (*attrs=None, elements=None, skipchildren=False*)**

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring (*pretty\_print=False*)**

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.Term**

**class** `pynlpl.formats.folia.Term`(*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

A term, often used in context of `Entry`

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.

Continued on next page

Table 33 – continued from previous page

<code>addtoindex([norecure])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Continued on next page

Table 33 – continued from previous page

<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 33 – continued from previous page

<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 38
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Term'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'term'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts (Class, raiseexceptions=True, parentinstance=None)
```

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to `None` to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):  
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.

- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:



```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype=None, set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*’th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=`True`, *scope*=`True`)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**text** (*cls='current'*, *retaintokenisation=False*, *previousdelimiter=""*, *strict=False*, *correctionhandling=1*, *normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

## pynlpl.formats.folia.TableHead

**class** `pynlpl.formats.folia.TableHead` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Encapsulated the header of a table, contains `Cell` elements

### Method Summary

<code>__init__</code> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<code>accepts</code> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<code>add</code> ( <i>child, *args, **kwargs</i> )	
<code>addable</code> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix</code> ( <i>idsuffix[, recursive]</i> )	Appends a suffix to this element's ID, and optionally to all child IDs as well.

Continued on next page

Table 34 – continued from previous page

<code>addtoindex([norecure])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Continued on next page

Table 34 – continued from previous page

<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page



Table 34 – continued from previous page

<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Table Header'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
XLINK = False
XMLTAG = 'tablehead'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts (Class, raiseexceptions=True, parentinstance=None)

```

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.

- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype=None, set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*’th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=`True`, *scope*=`True`)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:



```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls='current'*)

Alias for *text()* with *strict=True*

**text** (*cls='current'*, *retaintokenisation=False*, *previousdelimiter=""*, *strict=False*, *correctionhandling=1*, *normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words** (*index=None*)

Returns a generator of `Word` elements found (recursively) under this element.

**Parameters** *index* (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

**pynlpl.formats.folia.Text**

**class** `pynlpl.formats.folia.Text` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

A full text. This is a high-level element (not to be confused with `TextContent`!). This element may contain `Division`; `class:Paragraph`, `class:Sentence`, etc..

**Method Summary**

<code>__init__</code> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<code>accepts</code> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<code>add</code> ( <i>child, *args, **kwargs</i> )	
<code>addable</code> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.

Continued on next page

Table 35 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )

Continued on next page

Table 35 – continued from previous page

<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>

Continued on next page

Table 35 – continued from previous page

<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Text Body'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n\n'
XLINK = False
XMLTAG = 'text'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.
```

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to `None` to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.



**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retain\_tokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such an annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype=None, set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrected text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=",", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls='current'*)

Alias for *text()* with *strict=True*

**text** (*cls='current', retaintokenisation=False, previousdelimiter=", strict=False, correctionhandling=1, normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls*='current')

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Whitespace

**class** pynlpl.formats.folia.**Whitespace** (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`

Whitespace element, signals a vertical whitespace

### Method Summary

<code>__init__</code> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<code>accepts</code> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<code>add</code> ( <i>child, *args, **kwargs</i> )	

Continued on next page



Table 36 – continued from previous page

<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasannotationlayer([annotationtype, set])</code>	Does the specified annotation layer exist?
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	

Continued on next page

Table 36 – continued from previous page

<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>layers([annotationtype, set])</code>	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>paragraphs([index])</code>	Returns a generator of Paragraph elements found (recursively) under this element.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.

Continued on next page

Table 36 – continued from previous page

<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retain_tokenisation=True</code>
<code>update_text()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 6
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Whitespace'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ''
XLINK = False
XMLTAG = 'whitespace'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

```

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to `None` to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

**See also:**

*AbstractElement.select()*

#### Raises

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retain\_tokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such an annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasannotationlayer** (*annotationtype=None, set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype=None, set=None*)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrected text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n<sup>th</sup> element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=",", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.



- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls='current'*)

Alias for *text()* with *strict=True*

**text** (*cls='current', retaintokenisation=False, previousdelimiter=",", strict=False, correctionhandling=None, normalize\_spaces=False*)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Word

**class** pynlpl.formats.folia.**Word** (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractStructureElement`, `pynlpl.formats.folia.AllowCorrections`

Word (aka token) element. Holds a word/token and all its related token annotations.

## Method Summary

<code>__init__</code> ( <i>doc, *args, **kwargs</i> )	Constructor for words.
<code>accepts</code> ( <i>Class[, raiseexceptions, parentinstance]</i> )	

Continued on next page

Table 37 – continued from previous page

<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>division()</code>	Obtain the deepest division this word is a part of, otherwise return None
<code>domain([set])</code>	Shortcut: returns the FoLiA class of the domain annotation (will return only one if there are multiple!)
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspans(type[, set])</code>	Yields span annotation elements of the specified type that include this word.
<code>generate_id(cls)</code>	
<code>getcorrection([set, cls])</code>	
<code>getcorrections([set, cls])</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Returns the text delimiter
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.

Continued on next page

Table 37 – continued from previous page

<i>hasannotationlayer</i> ([annotationtype, set])	Does the specified annotation layer exist?
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>layers</i> ([annotationtype, set])	Returns a list of annotation layers found <i>directly</i> under this element, does not include alternative layers
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>lemma</i> ([set])	Shortcut: returns the FoLiA class of the lemma annotation (will return only one if there are multiple!)
<i>morpheme</i> (index[, set])	Returns a specific morpheme, the n'th morpheme (given the particular set if specified).
<i>morphemes</i> ([set])	Generator yielding all morphemes (in a particular set if specified).
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>paragraph</i> ()	Obtain the paragraph this word is a part of, otherwise return None
<i>paragraphs</i> ([index])	Returns a generator of Paragraph elements found (recursively) under this element.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>phoneme</i> (index[, set])	Returns a specific phoneme, the n'th morpheme (given the particular set if specified).
<i>phonemes</i> ([set])	Generator yielding all phonemes (in a particular set if specified).
<i>pos</i> ([set])	Shortcut: returns the FoLiA class of the PoS annotation (will return only one if there are multiple!)
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element

Continued on next page

Table 37 – continued from previous page

<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>sense([set])</code>	Shortcut: returns the FoLiA class of the sense annotation (will return only one if there are multiple!)
<code>sentence()</code>	Obtain the sentence this word is a part of, otherwise return None
<code>sentences([index])</code>	Returns a generator of Sentence elements found (recursively) under this element
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>split(*newwords, **kwargs)</code>	
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>words([index])</code>	Returns a generator of Word elements found (recursively) under this element.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = 1
AUTH = True
AUTO_GENERATE_ID = True
LABEL = 'Word/Token'

```



```

OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' '
XLINK = False
XMLTAG = 'w'

```

## Method Details

`__init__(doc, *args, **kwargs)`

Constructor for words.

See [AbstractElement.\\_\\_init\\_\\_](#) for all inherited keyword arguments and parameters.

Keyword arguments:

- `space` (bool): Indicates whether this token is followed by a space (defaults to True)

Example:

```

sentence.append( folia.Word, 'This')
sentence.append( folia.Word, 'is')
sentence.append( folia.Word, 'a')
sentence.append( folia.Word, 'test', space=False)
sentence.append( folia.Word, '.')

```

**See also:**

[AbstractElement.\\_\\_init\\_\\_](#)

`__init__(doc, *args, **kwargs)`

Constructor for words.

See [AbstractElement.\\_\\_init\\_\\_](#) for all inherited keyword arguments and parameters.

Keyword arguments:

- `space` (bool): Indicates whether this token is followed by a space (defaults to True)

Example:

```
sentence.append( folia.Word, 'This')
sentence.append( folia.Word, 'is')
sentence.append( folia.Word, 'a')
sentence.append( folia.Word, 'test', space=False)
sentence.append( folia.Word, '.')
```

See also:

`AbstractElement.__init__`

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, *\*args*, *\*\*kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

#### Parameters

- **Class** (*class*) – The python Class you want to retrieve (e.g. `PosAnnotation`). Or set to *None* to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to *None*, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/corretto').cls
```

**See also:**

*AllowTokenAnnotation.annotations()* *AbstractElement.select()*

**Raises** *NoSuchAnnotation* if no such annotation exists

**annotations** (*Class, set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/corretto'):
    ..
```

**See also:**

*AbstractElement.select()*

**Raises**

- *AllowTokenAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**division** ()

Obtain the deepest division this word is a part of, otherwise return `None`

**domain** (*set=None*)

Shortcut: returns the FoLiA class of the domain annotation (will return only one if there are multiple!)

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**findspans** (*type*, *set=None*)

Yields span annotation elements of the specified type that include this word.

#### Parameters

- **type** – The annotation type, can be passed as using any of the *AnnotationType* member, or by passing the relevant *AbstractSpanAnnotation* or *AbstractAnnotationLayer* class.
- **set** (*str* or *None*) – Constrain by set

Example:

```
for chunk in word.findspans(folia.Chunk):
    print(" Chunk class=", chunk.cls, " words=")
    for word2 in chunk.wrefs(): #print all words in the chunk (of which the
        ↪word is a part)
        print(word2, end=" ")
    print()
```

**Yields** Matching span annotation instances (derived from *AbstractSpanAnnotation*)

**generate\_id** (*cls*)

**getcorrection** (*set=None*, *cls=None*)

**getcorrections** (*set=None*, *cls=None*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Returns the text delimiter

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See *AllowTokenAnnotation.annotations`()* for a description of the parameters.

**hasannotationlayer** (*annotationtype=None*, *set=None*)

Does the specified annotation layer exist?

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.

- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**layers** (*annotationtype*=None, *set*=None)

Returns a list of annotation layers found *directly* under this element, does not include alternative layers

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**lemma** (*set*=None)

Shortcut: returns the FoLiA class of the lemma annotation (will return only one if there are multiple!)

**morpheme** (*index*, *set*=None)

Returns a specific morpheme, the *n*'th morpheme (given the particular set if specified).

**morphemes** (*set=None*)

Generator yielding all morphemes (in a particular set if specified). For retrieving one specific morpheme by index, use `morpheme()` instead

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**paragraph** ()

Obtain the paragraph this word is a part of, otherwise return `None`

**paragraphs** (*index=None*)

Returns a generator of Paragraph elements found (recursively) under this element.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the generator of all

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=",", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.

- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**phoneme** (*index*, *set*=None)

Returns a specific phoneme, the *n*'th morpheme (given the particular set if specified).

**phonemes** (*set*=None)

Generator yielding all phonemes (in a particular set if specified). For retrieving one specific morpheme by index, use `morpheme()` instead

**pos** (*set*=None)

Shortcut: returns the FoLiA class of the PoS annotation (will return only one if there are multiple!)

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.



**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**sense** (*set=None*)

Shortcut: returns the FoLiA class of the sense annotation (will return only one if there are multiple!)

**sentence** ()

Obtain the sentence this word is a part of, otherwise return None

**sentences** (*index=None*)

Returns a generator of Sentence elements found (recursively) under this element

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning a generator of all

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**split** (*\*newwords, \*\*kwargs*)

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**words** (*index=None*)

Returns a generator of Word elements found (recursively) under this element.

**Parameters** **index** (\*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

The [FoLiA documentation](#) explains the exact semantics and use of these in detail. Make sure to consult it to familiarize yourself with how the elements should be used.

FoLiA and this library enforce explicit rules about what elements are allowed in what others. Exceptions will be raised when this is about to be violated.

### 4.1.10 Common attributes

The FoLiA paradigm features *sets* and *classes* as primary means to represent the actual value (class) of an annotation. A set often corresponds to a tagset, such as a set of part-of-speech tags, and a class is one selected value in such a set.

The paradigm furthermore introduces other common attributes to set on annotation elements, such as an identifier, information on the annotator, and more. A full list is provided below:

- `element.id` (str) - The unique identifier of the element
- `element.set` (str) - The set the element pertains to.
- `element.cls` (str) - The assigned class, i.e. the actual value of the annotation, defined in the set. Classes correspond with tagsets in this case of many annotation types. Note that since *class* is already a reserved keyword in python, the library consistently uses `cls` everywhere.
- `element.annotator` (str) - The name or ID of the annotator who added/modified this element
- `element.annotatortype` - The type of annotator, can be either `folia.AnnotatorType.MANUAL` or `folia.AnnotatorType.AUTO`
- `element.confidence` (float) - A confidence value expressing
- `element.datetime` (datetime.datetime) - The date and time when the element was added/modified.
- `element.n` (str) - An ordinal label, used for instance in enumerated list contexts, numbered sections, etc..

The following attributes are specific to a speech context:

- `element.src` (str) - A URL or filename referring the an audio or video file containing the speech. Access this attribute using the `element.speaker_src()` method, as it is inheritable from ancestors.
- `element.speaker` (str) - The name or ID of the speaker. Access this attribute using the `element.speech_speaker()` method, as it is inheritable from ancestors.
- `element.begintime` (4-tuple) - The time in the above source fragment when the phonetic content of this element starts, this is a (hours, minutes, seconds, milliseconds) tuple.
- `element.endtime` (4-tuple) - The time in the above source fragment when the phonetic content of this element ends, this is a (hours, minutes, seconds, milliseconds) tuple.

Attributes that are not available for certain elements, or not set, default to `None`.

### 4.1.11 Annotations

As FoLiA is a format for linguistic annotation, accessing annotation is one of the primary functions of this library. This can be done using the methods `AllowTokenAnnotation.annotations()` or `AllowTokenAnnotation.annotation()` that are available on many FoLiA elements. These methods are similar to the `AbstractElement.select()` method except they will raise a `NoSuchAnnotation` exception when no such annotation is found. The difference between `annotation()` and `annotations()` is that the former will grab only one and raise an exception if there are more between which it can't disambiguate, whereas the second is a generator, but will still raise an exception if none is found:

```
for word in doc.words():
    try:
        pos = word.annotation(folia.PosAnnotation, 'http://somewhere/CGN')
        lemma = word.annotation(folia.LemmaAnnotation)
        print("Word: ", word)
        print("ID: ", word.id)
        print("PoS-tag: " , pos.cls)
        print("PoS Annotator: ", pos.annotator)
```

(continues on next page)

(continued from previous page)

```
print("Lemma-tag: " , lemma.cls)
except folia.NoSuchAnnotation:
    print("No PoS or Lemma annotation")
```

Note that the second argument of `AllowTokenAnnotation.annotation()`, `AllowTokenAnnotation.annotations()` or `AbstractElement.select()` can be used to restrict your selection to a certain set. In the above example we restrict ourselves to Part-of-Speech tags in the CGN set.

## Token Annotation Types

The following token annotation elements are available in FoLiA, they are embedded under a structural element (not necessarily a token, despite the name).

<i>DomainAnnotation</i>	Domain annotation: an extended token annotation element
<i>PosAnnotation</i>	Part-of-Speech annotation: a token annotation element
<i>LangAnnotation</i>	Language annotation: an extended token annotation element
<i>LemmaAnnotation</i>	Lemma annotation: a token annotation element
<i>SenseAnnotation</i>	Sense annotation: a token annotation element
<i>SubjectivityAnnotation</i>	Subjectivity annotation/Sentiment analysis: a token annotation element

## pynlpl.formats.folia.DomainAnnotation

**class** `pynlpl.formats.folia.DomainAnnotation` (*doc*, \*args, \*\*kwargs)  
 Bases: `pynlpl.formats.folia.AbstractExtendedTokenAnnotation`  
 Domain annotation: an extended token annotation element

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right

Continued on next page

Table 39 – continued from previous page

<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element

Continued on next page

Table 39 – continued from previous page

<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.LiA'>)
ANNOTATIONTYPE = 11
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Domain'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False

```



```

REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'domain'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (cls='current', previousdelimiter=",", strict=False, correctionhandling=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (str) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`



**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.PosAnnotation**

**class** `pynlpl.formats.folia.PosAnnotation` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractTokenAnnotation`

Part-of-Speech annotation: a token annotation element

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 40 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 40 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 9
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Part-of-Speech'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False

```

```
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'pos'
```

### Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs=None*, *recurse=True*, *ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.



**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an lxml.etree.Element

**See also:**

*AbstractElement.xmlstring()* - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for *text()*

## pynlpl.formats.folia.LangAnnotation

**class** pynlpl.formats.folia.LangAnnotation (*doc, \*args, \*\*kwargs*)

Bases: pynlpl.formats.folia.AbstractExtendedTokenAnnotation

Language annotation: an extended token annotation element

### Method Summary

<i>__init__</i> (doc, *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> (idsuffix[, recursive])	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ([norecurse])	Makes sure this element (and all subelements), are properly added to the index.
<i>ancestor</i> (*Classes)	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 41 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 41 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 31
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Language'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False

```

```

PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'lang'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)  
Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list



**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.LemmaAnnotation**

**class** `pynlpl.formats.folia.LemmaAnnotation` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractTokenAnnotation`

Lemma annotation: a token annotation element

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 42 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page



Table 42 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 10
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Lemma'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False

```

```
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'lemma'
```

## Method Details

`__init__(doc, *args, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

`__init__(doc, *args, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

`classmethod accepts(Class, raiseexceptions=True, parentinstance=None)`

`add(child, *args, **kwargs)`

`classmethod addable(parent, set=None, raiseexceptions=True)`  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

`addidsuffix(idsuffix, recursive=True)`

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

`addtoindex(norecurse=[])`

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

`ancestor(*Classes)`

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.



- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

## pynlpl.formats.folia.SenseAnnotation

**class** `pynlpl.formats.folia.SenseAnnotation` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractTokenAnnotation`

Sense annotation: a token annotation element

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 43 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 43 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 12
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Semantic Sense'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False

```

```

PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'sense'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (cls='current', previousdelimiter=",", strict=False, correctionhandling=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (str) – The class of the phonetic content to obtain, defaults to `current`.



- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an lxml.etree.Element

**See also:**

*AbstractElement.xmlstring()* - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for *text()*

**pynlpl.formats.folia.SubjectivityAnnotation**

**class** pynlpl.formats.folia.**SubjectivityAnnotation** (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractTokenAnnotation*

Subjectivity annotation/Sentiment analysis: a token annotation element

**Method Summary**

<i>__init__</i> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<i>accepts</i> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<i>add</i> ( <i>child, *args, **kwargs</i> )	
<i>addable</i> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> ( <i>idsuffix[, recursive]</i> )	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ( <i>[norecurse]</i> )	Makes sure this element (and all subelements), are properly added to the index.
<i>ancestor</i> ( <i>*Classes</i> )	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 44 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 44 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 19
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Subjectivity/Sentiment'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False

```

```
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'subjectivity'
```

## Method Details

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!



Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

## Text and phonetic annotation

The actual text of an element, or a phonetic textual representation, are also considered annotations themselves.

<i>TextContent</i>	Text content element (t), holds text to be associated with whatever element the text content element is a child of.
<i>PhonContent</i>	Phonetic content element (ph), holds a phonetic representation to be associated with whatever element the phonetic content element is a child of.

### pynlpl.formats.folia.TextContent

**class** `pynlpl.formats.folia.TextContent` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`

Text content element (t), holds text to be associated with whatever element the text content element is a child of.

Text content elements on structure elements like *Paragraph* and *Sentence* are by definition untokenised. Only on `Word` level and deeper they are by definition tokenised.

Text content elements can specify offset that refer to text at a higher parent level. Use the following keyword



arguments:

- `ref=`: The instance to point to, this points to the element holding the text content element, not the text content element itself.
- `offset=`: The offset where this text is found, offsets start at 0

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Example.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>finddefaultreference()</code>	Find the default reference for text offsets: The parent of the current textcontent's parent (counting only Structure Elements and Subtoken Annotation Elements)
<code>findreplaceables(parent, set, **kwargs)</code>	(Method for internal usage, see <code>AbstractElement</code> )
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>getreference([validate])</code>	Returns and validates the Text Content's reference.
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)

Continued on next page

Table 46 – continued from previous page

<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	See <code>AbstractElement.json()</code>
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	(Method for internal usage, see AbstractElement)
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text)</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([normalize_spaces])</code>	Obtain the text (unicode instance)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>

Continued on next page

Table 46 – continued from previous page

<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = 0
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Text'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (1, 2, 3, 5, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = True
TEXTDELIMITER = None
XLINK = True
XMLTAG = 't'

```

### Method Details

`__init__(doc, *args, **kwargs)`

Example:

```

text = folia.TextContent(doc, 'test')
text = folia.TextContent(doc, 'test', cls='original')

```

`__init__(doc, *args, **kwargs)`

Example:

```
text = folia.TextContent(doc, 'test')
text = folia.TextContent(doc, 'test', cls='original')
```

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, *\*args*, *\*\*kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, *\*args*, *\*\*kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.

- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**finddefaultreference** ()

Find the default reference for text offsets: The parent of the current textcontent's parent (counting only Structure Elements and Subtoken Annotation Elements)

Note: This returns not a `TextContent` element, but its parent. Whether the textcontent actually exists is checked later/elsewhere

**classmethod findreplaceables** (*parent, set, \*\*kwargs*)

(Method for internal usage, see `AbstractElement`)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**getreference** (*validate=True*)

Returns and validates the Text Content's reference. Raises `UnresolvableTextContent` when invalid

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

See `AbstractElement.json()`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrected text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

(Method for internal usage, see *AbstractElement*)

**phon** (*cls='current', previousdelimiter=" , strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

**See also:**

*phoncontent()*: Retrieves the phonetic content as an element rather than a string *text()*  
*textcontent()*

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike [phon\(\)](#), this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content ([PhonContent](#))

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

[phon\(\)](#) [textcontent\(\)](#) [text\(\)](#)

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use [AbstractElement.append\(\)](#) if you want the added element
- **be an alternative.** (*to*) –

See [AbstractElement.append\(\)](#) for more information and all parameters.



**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↪folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker ()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**speech\_src ()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext (cls='current')**

Alias for `text ()` with `strict=True`

**text (normalize\_spaces=False)**

Obtain the text (unicode instance)

**textcontent (cls='current', correctionhandling=1)**

Get the text content explicitly associated with this element (of the specified class).

Unlike `text ()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text ()` `phoncontent ()` `phon ()`

**textvalidation (warnonly=None)**

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext (cls='current')**

Alias for `text ()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml (attribs=None, elements=None, skipchildren=False)**

See `AbstractElement.xml ()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text* ()

## pynlpl.formats.folia.PhonContent

**class** pynlpl.formats.folia.**PhonContent** (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractElement*

Phonetic content element (ph), holds a phonetic representation to be associated with whatever element the phonetic content element is a child of.

Phonetic content elements behave much like text content elements.

Phonetic content elements can specify offset that refer to phonetic content at a higher parent level. Use the following keyword arguments:

- *ref*=: The instance to point to, this points to the element holding the text content element, not the text content element itself.
- *offset*=: The offset where this text is found, offsets start at 0

## Method Summary

<i>__init__</i> ( <i>doc, *args, **kwargs</i> )	Example.
<i>accepts</i> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<i>add</i> ( <i>child, *args, **kwargs</i> )	
<i>addable</i> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> ( <i>idsuffix[, recursive]</i> )	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ( <i>[norecurse]</i> )	Makes sure this element (and all subelements), are properly added to the index.
<i>ancestor</i> ( <i>*Classes</i> )	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ( <i>[Class]</i> )	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>append</i> ( <i>child, *args, **kwargs</i> )	

Continued on next page

Table 47 – continued from previous page

<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>finddefaultreference()</code>	Find the default reference for text offsets: The parent of the current textcontent's parent (counting only Structure Elements and Subtoken Annotation Elements)
<code>findreplaceables(parent, set, **kwargs)</code>	(Method for internal usage, see AbstractElement)
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>getreference([validate])</code>	Return and validate the Phonetic Content's reference.
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	(Method for internal usage, see AbstractElement)
<code>phon()</code>	Obtain the actual phonetic representation (unicode/str instance)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 47 – continued from previous page

<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setphon(phon)</code>	Set the representation for the phonetic content (unicode instance), called whenever <code>phon=</code> is passed as a keyword argument to an element constructor
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 18
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Phonetic Content'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0

```

```
OPTIONAL_ATTRIBS = (1, 2, 3, 5, 11)
PHONCONTAINER = True
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'ph'
```

## Method Details

`__init__(doc, *args, **kwargs)`

Example:

```
phon = folia.PhonContent(doc, 'hl')
phon = folia.PhonContent(doc, 'hl', cls="original")
```

`__init__(doc, *args, **kwargs)`

Example:

```
phon = folia.PhonContent(doc, 'hl')
phon = folia.PhonContent(doc, 'hl', cls="original")
```

**classmethod** `accepts` (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod** `addable` (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** `bool`

**Raises** `ValueError`

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecuse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**finddefaultreference** ()

Find the default reference for text offsets: The parent of the current textcontent's parent (counting only Structure Elements and Subtoken Annotation Elements)

Note: This returns not a TextContent element, but its parent. Whether the textcontent actually exists is checked later/elsewhere

**classmethod findreplaceables** (*parent, set, \*\*kwargs*)

(Method for internal usage, see AbstractElement)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**getreference** (*validate=True*)

Return and validate the Phonetic Content's reference. Raises UnresolvableTextContent when invalid

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hasstext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.



**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

(Method for internal usage, see `AbstractElement`)

**phon** ()

Obtain the actual phonetic representation (unicode/str instance)

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike [phon\(\)](#), this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content ([PhonContent](#))

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

[phon\(\)](#) [textcontent\(\)](#) [text\(\)](#)

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use [AbstractElement.append\(\)](#) if you want the added element
- **be an alternative.** (*to*) –

See [AbstractElement.append\(\)](#) for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setphon** (*phon*)

Set the representation for the phonetic content (unicode instance), called whenever phon= is passed as a keyword argument to an element constructor

**settext** (*text, cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text

- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

Text is retrieved as string using `AbstractElement.text()`, or as element using Phonetic content is retrieved as string using `AbstractElement.text()`, or as element using `AbstractElement.textcontent()`.

---

**Note:** These are the only elements for which FoLiA prescribes a default set and a default class (`current`). This will only be relevant if you work with multiple text layers (current text vs OCR'd text for instance) or with corrections of orthography or phonetics.

---

## Span Annotation

FoLiA distinguishes token annotation and span annotation, token annotation is embedded in-line within a structural element, and the annotation therefore pertains to that structural element, whereas span annotation is stored in a stand-off annotation layer outside the element and refers back to it. Span annotation elements typically *span* over multiple structural elements, they are all subclasses of `AbstractSpanAnnotation`.

We will discuss three ways of accessing span annotation. As stated, span annotation is contained within an annotation layer (a subclass of `AbstractAnnotationLayer`) of a certain structure element, often a sentence. In the first way of accessing span annotation, we do everything explicitly: We first obtain the layer, then iterate over the span annotation elements within that layer, and finally iterate over the words to which the span applies. Assume we have a sentence and we want to print all the named entities in it, assuming the entities layer is embedded at sentence level as is conventional:

```
for layer in sentence.select(folia.EntitiesLayer):
    for entity in layer.select(folia.Entity):
        print(" Entity class=", entity.cls, " words=")
        for word in entity.wrefs():
            print(word, end="") #print without newline
        print() #print newline
```

The `AbstractSpanAnnotation.wrefs()` method, available on all span annotation elements, will return a list of all words (as well as morphemes and phonemes) over which a span annotation element spans.

This first way is rather verbose. The second way of accessing span annotation takes another approach, using the `Word.findspans()` method available on `Word` instances. Here we start from a word and seek span annotations in which that word occurs. Assume we have a word and want to find chunks it occurs in:

```
for chunk in word.findspans(folia.Chunk):
    print(" Chunk class=", chunk.cls, " words=")
    for word2 in chunk.wrefs(): #print all words in the chunk (of which the word is a
    ↪part)
        print(word2, end="")
    print()
```

The `Word.findspans()` method can be called with either the class of a Span Annotation Element, such as `Chunk`, or with the class of the layer, such as `ChunkingLayer`.

The third way allows us to look for span elements given an annotation layer and words. In other words, it checks if one or more words form a span. This is an exact match and not a sub-part match as in the previously described method. To do this, we use the `AbstractAnnotationLayer.findspan` method, available on all annotation layers:

```
for span in annotationlayer.findspan(word1, word2):
    print("Class: ", span.cls)
    print("Text: ", span.text()) #same for every span here
```

## Span Annotation Types

This section lists the available Span annotation elements, the layer that contains them is explicitly mentioned as well.

Some of the span annotation elements are complex and take span role elements as children, these are normal span annotation elements that occur on a within another span annotation (of a particular type) and can not be used standalone.

FoLiA distinguishes the following span annotation elements:

<i>Chunk</i>	Chunk element, span annotation element to be used in <i>ChunkingLayer</i>
<i>CoreferenceChain</i>	Coreference chain.
<i>Dependency</i>	Span annotation element to encode dependency relations
<i>Entity</i>	Entity element, for entities such as named entities, multi-word expressions, temporal entities.
<i>Observation</i>	Observation.
<i>Predicate</i>	Predicate, used within <i>SemanticRolesLayer</i> , takes <i>SemanticRole</i> annotations as children, but has its own annotation type and separate declaration
<i>Sentiment</i>	Sentiment.
<i>Statement</i>	Statement.
<i>SyntacticUnit</i>	Syntactic Unit, span annotation element to be used in <i>SyntaxLayer</i>
<i>SemanticRole</i>	Semantic Role
<i>TimeSegment</i>	A time segment

### pynlpl.formats.folia.Chunk

**class** pynlpl.formats.folia.Chunk (*doc*, \*args, \*\*kwargs)  
 Bases: *pynlpl.formats.folia.AbstractSpanAnnotation*  
 Chunk element, span annotation element to be used in *ChunkingLayer*

#### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.

Continued on next page

Table 49 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.

Continued on next page



Table 49 – continued from previous page

<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 49 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 14
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Chunk'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'chunk'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.
```

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool**Raises** ValueError**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If `idsuffix` is a string, if set to `True`, a random `idsuffix` will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** `int`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** `str` or `list`

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** `int`

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.



**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan(\*args)**

Sets the span of the span element anew, erases all data inside.

**Parameters** *\*args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iff you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be `Words` but also `Morphemes` or `Phonemes`.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

`__iter__()`

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.CoreferenceChain

**class** pynlpl.formats.folia.CoreferenceChain(*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Coreference chain. Holds `CoreferenceLink` instances.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)

Continued on next page

Table 50 – continued from previous page

<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.

Continued on next page

Table 50 – continued from previous page

<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 28
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Coreference Chain'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True

```

```
REQUIRED_ATTRIBS = None
REQUIRED_DATA = (<class 'pynlpl.formats.folia.CoreferenceLink'>,)
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'coreferencechain'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to *True*, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** *int*

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.featsynset('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)



By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

#### **incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

#### **insert** (*index, child, \*args, \*\*kwargs*)

#### **items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

#### **json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

#### Returns dict

#### **leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

#### **next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence, Paragraph, Division, Event, ListItem, Caption`), set to `None` to not constrain at all.

#### **originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

#### **classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the `Class`.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if

you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or None if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (cls='current')

Alias for `text()` with `strict=True`

**text** (cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (str) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (bool) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (str) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (bool) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (bool) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (cls='current', correctionhandling=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (str) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if

you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

*text()* *phoncontent()* *phon()*

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for *text()* with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See *AbstractElement.xml()*

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text()*

## pynlpl.formats.folia.Dependency

**class** `pynlpl.formats.folia.Dependency` (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractSpanAnnotation*

Span annotation element to encode dependency relations

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>dependent()</code>	Returns the dependent of the dependency relation.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)

Continued on next page

Table 51 – continued from previous page

<code>head()</code>	Returns the head of the dependency relation.
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.

Continued on next page



Table 51 – continued from previous page

<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retain_tokenisation=True</code>
<code>update_text()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 22
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Dependency'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = (<class 'pynlpl.formats.folia.DependencyDependent'>, <class 'pynlpl.fo
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'dependency'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

```

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)

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

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

#### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** `int`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**dependent** ()

Returns the dependent of the dependency relation. Instance of `DependencyDependent`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** `str` or `list`

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**head()**

Returns the head of the dependency relation. Instance of *Headspan*

**incorrection()**

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert(index, child, \*args, \*\*kwargs)**

**items(founditems=[])**

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json(attrs=None, recurse=True, ignorelist=False)**

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext(size, placeholder=None, scope=None)**

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next(Class=True, scope=True, reverse=False)**

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext(cls='original')**

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml(node, doc, \*\*kwargs)**

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon(cls='current', previousdelimiter="", strict=False, correctionhandling=1)**

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative, AlternativeLayer, Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.

- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.



**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls*='current')

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** *index* (*int* or *None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Entity

**class** pynlpl.formats.folia.**Entity** (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Entity element, for entities such as named entities, multi-word expressions, temporal entities. This is a span annotation element to be used in `EntitiesLayer`

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.

Continued on next page

Table 52 – continued from previous page

<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.

Continued on next page

Table 52 – continued from previous page

<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

## Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 15
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Entity'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'entity'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (`AbstractElement` or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (`AbstractElement` or subclasses). Not instances!

**Yields** elements (instances derived from `AbstractElement`)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (`Document`) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to *True*, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** `int`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** `str` or `list`

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** `int`

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

#### Returns dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`



**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter="", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to True, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchText – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** NoSuchText if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.Observation

**class** pynlpl.formats.folia.Observation(*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Observation.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.

Continued on next page

Table 53 – continued from previous page

<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>generate_id</i> (cls)	
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.

Continued on next page

Table 53 – continued from previous page

<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 43
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Observation'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False

```



```

SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'observation'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to *True*, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such an annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – `XML Element` (\*) –

- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `'AbstractElement'`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*=*'current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element



See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Predicate

**class** `pynlpl.formats.folia.Predicate` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Predicate, used within `SemanticRolesLayer`, takes `SemanticRole` annotations as children, but has its own annotation type and separate declaration

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	

Continued on next page

Table 54 – continued from previous page

<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 54 – continued from previous page

<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 42
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Predicate'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'predicate'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
```

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

#### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter="", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.



- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\*args)

Sets the span of the span element anew, erases all data inside.

**Parameters** \*args – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml ()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text ()`

**pynlpl.formats.folia.Sentiment**

**class** `pynlpl.formats.folia.Sentiment` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Sentiment. Takes span roles `Headspan`, `Source` and `Target` as children

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 55 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)

Continued on next page

Table 55 – continued from previous page

<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>setspan</i> (*args)	Sets the span of the span element anew, erases all data inside.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <code>text()</code> with <code>strict=True</code>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<i>updatetext</i> ()	Recompute textual value based on the text content of the children.
<i>wrefs</i> ([index, recurse])	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<i>xml</i> ([attribs, elements, skipchildren])	See <code>AbstractElement.xml()</code>
<i>xmlstring</i> ([pretty_print])	Serialises this FoLiA element and all its contents to XML.
<code>__iter__</code> ()	Iterate over all children of this element.
<code>__len__</code> ()	Returns the number of child elements under the current element.
<code>__str__</code> ()	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 44

```

```
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Sentiment'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'sentiment'
```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*



**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (`AbstractElement` or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (`AbstractElement` or subclasses). Not instances!

**Yields** elements (instances derived from `AbstractElement`)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See `AbstractElement.append()`

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (`Document`) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If `idsuffix` is a string, if set to `True`, a random `idsuffix` will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling*=*CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retain\_tokenisation*=False, *previous\_delimiter*="", *strict*=False, *correction\_handling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retain\_tokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previous\_delimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correction\_handling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls*='current')

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index*=None, *recurse*=True)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int* or *None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attrs*=None, *elements*=None, *skipchildren*=False)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print*=False)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```



`__len__()`  
Returns the number of child elements under the current element.

`__str__()`  
Alias for `text()`

## pynlpl.formats.folia.Statement

**class** pynlpl.formats.folia.Statement(*doc*, \**args*, \*\**kwargs*)  
Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`  
Statement. Takes span roles *Headspan*, *Source* and *Relation* as children

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	

Continued on next page

Table 56 – continued from previous page

<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>setspan</i> (*args)	Sets the span of the span element anew, erases all data inside.
<i>settext</i> (text[, cls])	Set the text for this element.

Continued on next page

Table 56 – continued from previous page

<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 45
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Statement'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False

```

```
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'statement'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=None*)  
Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)  
Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

#### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, *\*args*, *\*\*kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – `XML Element` (\*) –
- **doc** – `Document` (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element



See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*=*'current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.SyntacticUnit**

**class** `pynlpl.formats.folia.SyntacticUnit` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Syntactic Unit, span annotation element to be used in `SyntaxLayer`

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	

Continued on next page

Table 57 – continued from previous page

<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 57 – continued from previous page

<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 13
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Syntactic Unit'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'su'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts (Class, raiseexceptions=True, parentinstance=None)

```

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \**args*, \*\**kwargs*)

See *AbstractElement.append()*



**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter="", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.

- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml ()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text ()`

## pynlpl.formats.folia.SemanticRole

**class** pynlpl.formats.folia.SemanticRole (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`

Semantic Role

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page



Table 58 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)

Continued on next page

Table 58 – continued from previous page

<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 29
```

```

AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Semantic Role'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'semrole'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (`AbstractElement` or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (`AbstractElement` or subclasses). Not instances!

**Yields** elements (instances derived from `AbstractElement`)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See `AbstractElement.append()`

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (`Document`) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If `idsuffix` is a string, if set to `True`, a random `idsuffix` will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling*=*CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element



- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iff you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`  
Returns the number of child elements under the current element.

`__str__()`  
Alias for `text()`

## `pynlpl.formats.folia.TimeSegment`

**class** `pynlpl.formats.folia.TimeSegment` (*doc*, \**args*, \*\**kwargs*)  
Bases: `pynlpl.formats.folia.AbstractSpanAnnotation`  
A time segment

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	

Continued on next page

Table 59 – continued from previous page

<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>setspan</i> (*args)	Sets the span of the span element anew, erases all data inside.
<i>settext</i> (text[, cls])	Set the text for this element.

Continued on next page

Table 59 – continued from previous page

<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.ActorFeature'>, <class 'pynlpl.formats.f
ANNOTATIONTYPE = 23
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Time Segment'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False

```

```
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'timesegment'
```

## Method Details

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)  
Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=None*)  
Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters \*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)  
Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters \*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, *\*args*, *\*\*kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:



```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – `XML Element` (\*) –
- **doc** – `Document` (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

These are placed in the following annotation layers:

<i>ChunkingLayer</i>	Chunking Layer: Annotation layer for <i>Chunk</i> span annotation elements
<i>CoreferenceLayer</i>	Syntax Layer: Annotation layer for <i>SyntacticUnit</i> span annotation elements
<i>DependenciesLayer</i>	Dependencies Layer: Annotation layer for <i>Dependency</i> span annotation elements.
<i>EntitiesLayer</i>	Entities Layer: Annotation layer for <i>Entity</i> span annotation elements.

Continued on next page

Table 60 – continued from previous page

<i>ObservationLayer</i>	Observation Layer: Annotation layer for <i>Observation</i> span annotation elements.
<i>SentimentLayer</i>	Sentiment Layer: Annotation layer for <i>Sentiment</i> span annotation elements, used for sentiment analysis.
<i>StatementLayer</i>	Statement Layer: Annotation layer for <i>Statement</i> span annotation elements, used for attribution annotation.
<i>SyntaxLayer</i>	Syntax Layer: Annotation layer for <i>SyntacticUnit</i> span annotation elements
<i>SemanticRolesLayer</i>	Syntax Layer: Annotation layer for <i>SemanticRole</i> span annotation elements
<i>TimingLayer</i>	Timing layer: Annotation layer for <i>TimeSegment</i> span annotation elements.

### pynlpl.formats.folia.ChunkingLayer

**class** pynlpl.formats.folia.**ChunkingLayer** (*doc*, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractAnnotationLayer*

Chunking Layer: Annotation layer for *Chunk* span annotation elements

#### Method Summary

<i>__init__</i> ( <i>doc</i> , *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> (idsuffix[, recursive])	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ([norecurse])	Makes sure this element (and all subelements), are properly added to the index.
<i>alternatives</i> ([Class, set])	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<i>ancestor</i> (*Classes)	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ([Class])	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>annotation</i> (type[, set])	Will return a <b>single</b> annotation (even if there are multiple).
<i>annotations</i> (Class[, set])	Obtain annotations.
<i>append</i> (child, *args, **kwargs)	See <i>AbstractElement.append()</i>
<i>context</i> (size[, placeholder, scope])	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy</i> ([newdoc, idsuffix])	Make a deep copy of this element and all its children.
<i>copychildren</i> ([newdoc, idsuffix])	Generator creating a deep copy of the children of this element.

Continued on next page



Table 61 – continued from previous page

<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

Continued on next page

Table 61 – continued from previous page

<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Chunk'>, <class 'pynlpl.formats.folia.Co
ANNOTATIONTYPE = 14
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None

```

```

REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'chunking'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \*Classes – The possible classes ([AbstractElement](#) or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (Class=None)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \*Class – The class or classes ([AbstractElement](#) or subclasses). Not instances!

**Yields** elements (instances derived from [AbstractElement](#))

**annotation** (type, set=None)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (Class, set=None)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (child, \*args, \*\*kwargs)

See [AbstractElement.append\(\)](#)

**context** (size, placeholder=None, scope=None)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (newdoc=None, idsuffix="")

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** ([Document](#)) – The document the copy should be associated with.
- **idsuffix** (str or bool) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (newdoc=None, idsuffix="")

Generator creating a deep copy of the children of this element.

Invokes [copy\(\)](#) on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (Class, set=None, recursive=True, ignore=True, node=None)

Like [AbstractElement.select\(\)](#), but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation()**

Perform deep validation of this element.

**Raises** DeepValidationError

**description()**

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**findspan(\*words)**

Returns the span element which spans over the specified words or morphemes.

**See also:**

*Word.findspans()*

**generate\_id(cls)**

**getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation(Class, set=None)**

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined *scope*. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattrs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments



- **alternative** (*bool*) – If set to True, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

**pynlpl.formats.folia.CoreferenceLayer**

**class** pynlpl.formats.folia.CoreferenceLayer (*doc*, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractAnnotationLayer*

Syntax Layer: Annotation layer for *SyntacticUnit* span annotation elements

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <i>AbstractElement.append()</i>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!

Continued on next page

Table 62 – continued from previous page

<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <code>text()</code> with <code>strict=True</code>

Continued on next page

Table 62 – continued from previous page

<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.  
ANNOTATIONTYPE = 28  
AUTH = True  
AUTO_GENERATE_ID = False  
OCCURRENCES = 0  
OCCURRENCES_PER_SET = 0  
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)  
PHONCONTAINER = False  
PRIMARYELEMENT = False  
PRINTABLE = False  
REQUIRED_ATTRIBS = None  
REQUIRED_DATA = None  
SETONLY = True  
SPEAKABLE = False  
SUBSET = None  
TEXTCONTAINER = False  
TEXTDELIMITER = None  
XLINK = False  
XMLTAG = 'coreferences'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child*, *\*args*, *\*\*kwargs*)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.



**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.featsynset('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

*Word.findspans()*

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to

`CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattrs=None*, *extraelements=None*, *orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the

corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

*text()* *phoncontent()* *phon()*

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for *text()* with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See *AbstractElement.xml()*

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text()*

## pynlpl.formats.folia.DependenciesLayer

**class** `pynlpl.formats.folia.DependenciesLayer` (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractAnnotationLayer*

Dependencies Layer: Annotation layer for *Dependency* span annotation elements. For dependency entities.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)

Continued on next page



Table 63 – continued from previous page

<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <i>text()</i> with <i>retaintokenisation=True</i>

Continued on next page

Table 63 – continued from previous page

<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.  
ANNOTATIONTYPE = 22  
AUTH = True  
AUTO_GENERATE_ID = False  
OCCURRENCES = 0  
OCCURRENCES_PER_SET = 0  
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)  
PHONCONTAINER = False  
PRIMARYELEMENT = False  
PRINTABLE = False  
REQUIRED_ATTRIBS = None  
REQUIRED_DATA = None  
SETONLY = True  
SPEAKABLE = False  
SUBSET = None  
TEXTCONTAINER = False  
TEXTDELIMITER = None  
XLINK = False  
XMLTAG = 'dependencies'
```

### Method Details

```
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
classmethod accepts (Class, raiseexceptions=True, parentinstance=None)  
add (child, *args, **kwargs)
```

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

*Word.findspans()*

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.

- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.



- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for *text()* with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.EntitiesLayer

**class** `pynlpl.formats.folia.EntitiesLayer` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Entities Layer: Annotation layer for *Entity* span annotation elements. For named entities.

### Method Summary

<code>__init__</code> ( <i>doc, *args, **kwargs</i> )	Initialize self.
<code>accepts</code> ( <i>Class[, raiseexceptions, parentinstance]</i> )	
<code>add</code> ( <i>child, *args, **kwargs</i> )	
<code>addable</code> ( <i>parent[, set, raiseexceptions]</i> )	Tests whether a new element of this class can be added to the parent.

Continued on next page

Table 64 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)

Continued on next page

Table 64 – continued from previous page

<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 64 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.  
ANNOTATIONTYPE = 15  
AUTH = True  
AUTO_GENERATE_ID = False  
OCCURRENCES = 0  
OCCURRENCES_PER_SET = 0  
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)  
PHONCONTAINER = False  
PRIMARYELEMENT = False  
PRINTABLE = False  
REQUIRED_ATTRIBS = None  
REQUIRED_DATA = None  
SETONLY = True  
SPEAKABLE = False  
SUBSET = None  
TEXTCONTAINER = False  
TEXTDELIMITER = None  
XLINK = False  
XMLTAG = 'entities'
```

### Method Details

```
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
  
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
  
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)  
add(child, *args, **kwargs)  
  
classmethod addable(parent, set=None, raiseexceptions=True)  
    Tests whether a new element of this class can be added to the parent.  
  
    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden  
    by subclasses for more customised behaviour.
```

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child*, \**args*, \*\**kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {*size*} words to the left, the current word, and {*size*} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\**kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.



**findspan** (\*words)

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id** (cls)

**getindex** (child, recursive=True, ignore=True)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (key=None)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (retaintokenisation=False)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (Class, set=None)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (cls='current', strict=True, correctionhandling=1)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (str) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (bool) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (cls='current', strict=True, correctionhandling=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (str) – The class of the text content to obtain, defaults to `current`.
- **strict** (bool) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml (attrs=None, elements=None, skipchildren=False)**

See `AbstractElement.xml ()`

**xmlstring (pretty\_print=False)**

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text ()`

**pynlpl.formats.folia.ObservationLayer**

**class** pynlpl.formats.folia.ObservationLayer (doc, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Observation Layer: Annotation layer for *Observation* span annotation elements.

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

Continued on next page

Table 65 – continued from previous page

<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)

Continued on next page



Table 65 – continued from previous page

<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <i>text()</i> with <i>retaintokenisation=True</i>
<i>updatetext</i> ()	Recompute textual value based on the text content of the children.
<i>xml</i> ([attribs, elements, skipchildren])	See <i>AbstractElement.xml()</i>
<i>xmlstring</i> ([pretty_print])	Serialises this FoLiA element and all its contents to XML.
<i>__iter__</i> ()	Iterate over all children of this element.
<i>__len__</i> ()	Returns the number of child elements under the current element.
<i>__str__</i> ()	Alias for <i>text()</i>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 43
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0

```

```
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'observations'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.



**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchText – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** NoSuchText if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.SentimentLayer

**class** pynlpl.formats.folia.SentimentLayer(*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Sentiment Layer: Annotation layer for *Sentiment* span annotation elements, used for sentiment analysis.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.

Continued on next page

Table 66 – continued from previous page

<i>feat</i> (subset)	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>findspan</i> (*words)	Returns the span element which spans over the specified words or morphemes.
<i>generate_id</i> (cls)	
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.

Continued on next page

Table 66 – continued from previous page

<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 44
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None

```

```
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'sentiments'
```

## Method Details

`__init__` (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

`__init__` (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod** `accepts` (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod** `addable` (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description()**

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan(\*words)**

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id(cls)****getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation(Class, set=None)**

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.



- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all

- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

#### See also:

`phon()` `textcontent()` `text()`

#### `postappend()`

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

#### `previous (Class=True, scope=True)`

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

#### `classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, orig-class=None)`

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

#### `remove (child)`

Removes the child element

#### `replace (child, *args, **kwargs)`

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

#### `resolveword (id)`

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iff you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

*text()* *phoncontent()* *phon()*

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for *text()* with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See *AbstractElement.xml()*

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text()*

## **pynlpl.formats.folia.StatementLayer**

**class** `pynlpl.formats.folia.StatementLayer` (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractAnnotationLayer*

Statement Layer: Annotation layer for *Statement* span annotation elements, used for attribution annotation.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)

Continued on next page

Table 67 – continued from previous page

<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <i>text()</i> with <i>retaintokenisation=True</i>

Continued on next page



Table 67 – continued from previous page

<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 45
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'statements'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

```

**classmethod** **addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

*Word.findspans()*

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.

- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for *text()* with `strict=True`



**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.SyntaxLayer

**class** pynlpl.formats.folia.**SyntaxLayer** (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Syntax Layer: Annotation layer for *SyntacticUnit* span annotation elements

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.

Continued on next page

Table 68 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)

Continued on next page

Table 68 – continued from previous page

<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 68 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 13
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'syntax'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
add(child, *args, **kwargs)
classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.

```

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (\*\*kwargs)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\*kwargs)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan** (\*words)

Returns the span element which spans over the specified words or morphemes.

**See also:**

*Word.findspans()*

**generate\_id** (cls)

**getindex** (child, recursive=True, ignore=True)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (key=None)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (retaintokenisation=False)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (Class, set=None)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (cls='current', strict=True, correctionhandling=1)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (str) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (bool) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (cls='current', strict=True, correctionhandling=1)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (str) – The class of the text content to obtain, defaults to *current*.
- **strict** (bool) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.



**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=",", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_()**

Returns the number of child elements under the current element.

**\_\_str\_\_()**

Alias for `text()`

## **pynlpl.formats.folia.SemanticRolesLayer**

**class** `pynlpl.formats.folia.SemanticRolesLayer` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Syntax Layer: Annotation layer for *SemanticRole* span annotation elements

### **Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

Continued on next page

Table 69 – continued from previous page

<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>findspan(*words)</code>	Returns the span element which spans over the specified words or morphemes.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)

Continued on next page

Table 69 – continued from previous page

<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <i>text()</i> with <i>retaintokenisation=True</i>
<i>updatetext</i> ()	Recompute textual value based on the text content of the children.
<i>xml</i> ([attribs, elements, skipchildren])	See <i>AbstractElement.xml()</i>
<i>xmlstring</i> ([pretty_print])	Serialises this FoLiA element and all its contents to XML.
<i>__iter__</i> ()	Iterate over all children of this element.
<i>__len__</i> ()	Returns the number of child elements under the current element.
<i>__str__</i> ()	Alias for <i>text()</i>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.
ANNOTATIONTYPE = 29
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0

```



```

OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'semroles'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan** (*\*words*)

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class*, *set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None*, *recurse=True*, *ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:



```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchText – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** NoSuchText if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls*='current')

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs*=None, *elements*=None, *skipchildren*=False)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print*=False)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.TimingLayer

**class** pynlpl.formats.folia.**TimingLayer**(*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractAnnotationLayer`

Timing layer: Annotation layer for *TimeSegment* span annotation elements.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.

Continued on next page

Table 70 – continued from previous page

<i>feat</i> (subset)	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>findspan</i> (*words)	Returns the span element which spans over the specified words or morphemes.
<i>generate_id</i> (cls)	
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.

Continued on next page

Table 70 – continued from previous page

<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.  
ANNOTATIONTYPE = 23  
AUTH = True  
AUTO_GENERATE_ID = False  
OCCURRENCES = 0  
OCCURRENCES_PER_SET = 0  
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 10, 11)  
PHONCONTAINER = False  
PRIMARYELEMENT = False  
PRINTABLE = False  
REQUIRED_ATTRIBS = None  
REQUIRED_DATA = None  
SETONLY = True  
SPEAKABLE = False  
SUBSET = None
```

```

TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'timing'

```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None*, *set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Returns** Generator over Alternative elements

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a *NoSuchAnnotation* exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to *select()* but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** *NoSuchAnnotation* if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description()**

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**findspan(\*words)**

Returns the span element which spans over the specified words or morphemes.

**See also:**

`Word.findspans()`

**generate\_id(cls)****getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation(Class, set=None)**

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.

- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all



- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=",", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

#### See also:

`phon()` `textcontent()` `text()`

#### `postappend()`

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

#### `previous (Class=True, scope=True)`

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

#### `classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, originalclass=None)`

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

#### `remove (child)`

Removes the child element

#### `replace (child, *args, **kwargs)`

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

#### `resolveword (id)`

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

Some span annotation elements take *span roles*, depending on their type:

<i>CoreferenceLink</i>	Coreference link.
<i>DependencyDependent</i>	Span role element that marks the dependent in a dependency relation.

Continued on next page

Table 71 – continued from previous page

<i>Headspan</i>	The headspan role is used to mark the head of a span annotation.
-----------------	--

**pynlpl.formats.folia.CoreferenceLink**

**class** pynlpl.formats.folia.**CoreferenceLink** (*doc*, \*args, \*\*kwargs)

Bases: pynlpl.formats.folia.AbstractSpanRole

Coreference link. Used in *CoreferenceChain*

**Method Summary**

<i>__init__</i> (doc, *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> (idsuffix[, recursive])	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ([norecurse])	Makes sure this element (and all subelements), are properly added to the index
<i>ancestor</i> (*Classes)	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ([Class])	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>annotation</i> (type[, set])	Will return a <b>single</b> annotation (even if there are multiple).
<i>annotations</i> (Class[, set])	Obtain annotations.
<i>append</i> (child, *args, **kwargs)	See <i>AbstractElement.append()</i>
<i>context</i> (size[, placeholder, scope])	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy</i> ([newdoc, idsuffix])	Make a deep copy of this element and all its children.
<i>copychildren</i> ([newdoc, idsuffix])	Generator creating a deep copy of the children of this element.
<i>correct</i> (**kwargs)	Apply a correction (TODO: documentation to be written still)
<i>count</i> (Class[, set, recursive, ignore, node])	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<i>deepvalidation</i> ()	Perform deep validation of this element.
<i>description</i> ()	Obtain the description associated with the element.
<i>feat</i> (subset)	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>generate_id</i> (cls)	
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!

Continued on next page

Table 72 – continued from previous page

<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasannotation</i> (Class[, set])	Returns an integer indicating whether such as annotation exists, and if so, how many.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attrs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattrs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>setspan</i> (*args)	Sets the span of the span element anew, erases all data inside.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.

Continued on next page

Table 72 – continued from previous page

<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 28
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Coreference Link'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 4, 5)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False

```



**XMLTAG** = 'coreferencelink'

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, *\*args*, *\*\*kwargs*)

See `AbstractElement.append()`

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** `int`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** `str` or `list`

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See *annotations()* for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you

want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attribs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=",", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

**Keyword Arguments**

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.

- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authoritative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext** (*text*, *cls*=`'current'`)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retain\_tokenisation*=False, *previous\_delimiter*="", *strict*=False, *correction\_handling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retain\_tokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previous\_delimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correction\_handling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correction\_handling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correction\_handling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element



See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** *bool*

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n’t element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** *str*

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.DependencyDependent

**class** `pynlpl.formats.folia.DependencyDependent` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractSpanRole`

Span role element that marks the dependent in a dependency relation. Used in `Dependency`.

`Headspan` in turn is used to mark the head of a dependency relation.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	

Continued on next page

Table 73 – continued from previous page

<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 73 – continued from previous page

<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Dependent'
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 4, 5)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'dep'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts (Class, raiseexceptions=True, parentinstance=None)
```

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type*, *set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class*, *set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

#### Parameters

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child*, \*args, \*\*kwargs)

See *AbstractElement.append()*

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element. If *idsuffix* is a string, if set to `True`, a random *idsuffix* will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter="", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.



- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of *Word*, *Morpheme* or *Phoneme*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** *index* (*int or None*) – If set to an integer, will retrieve and return the *n*'th element (starting at 0) instead of returning the list of all

**xml** (*attribs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_()**

Returns the number of child elements under the current element.

**\_\_str\_\_()**

Alias for `text()`

**pynlpl.formats.folia.Headspan**

**class** pynlpl.formats.folia.**Headspan** (*doc, \*args, \*\*kwargs*)

Bases: pynlpl.formats.folia.AbstractSpanRole

The headspan role is used to mark the head of a span annotation.

It can be used in various contexts, for instance to mark the head of a *Dependency*. It is allowed by most span annotations.

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index

Continued on next page

Table 74 – continued from previous page

<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Will return a <b>single</b> annotation (even if there are multiple).
<code>annotations(Class[, set])</code>	Obtain annotations.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.

Continued on next page

Table 74 – continued from previous page

<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>setspan(*args)</code>	Sets the span of the span element anew, erases all data inside.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>wrefs([index, recurse])</code>	Returns a list of word references, these can be Words but also Morphemes or Phonemes.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
```

```
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Head'
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 4, 5)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'hd'
```

## Method Details

`__init__(doc, *args, **kwargs)`

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

`__init__(doc, *args, **kwargs)`

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

**classmethod** `accepts` (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod** `addable` (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** `bool`

**Raises** `ValueError`



**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=None*)

Makes sure this element (and all subelements), are properly added to the index

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (`AbstractElement` or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (`AbstractElement` or subclasses). Not instances!

**Yields** elements (instances derived from `AbstractElement`)

**annotation** (*type, set=None*)

Will return a **single** annotation (even if there are multiple). Raises a `NoSuchAnnotation` exception if none was found

**annotations** (*Class, set=None*)

Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

- **Class** – The Class you want to retrieve (\*) –
- **set** – The set you want to retrieve (\*) –

**Yields** elements

**Raises** `NoSuchAnnotation` if the specified annotation does not exist.

**append** (*child, \*args, \*\*kwargs*)

See `AbstractElement.append()`

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (`Document`) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element. If `idsuffix` is a string, if set to `True`, a random `idsuffix` will be generated including a random 32-bit hash

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many. See `annotations()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the `Correction` element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling*=*CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**setspan** (\**args*)

Sets the span of the span element anew, erases all data inside.

**Parameters** \**args* – Instances of `Word`, `Morpheme` or `Phoneme`

**settext** (*text, cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this iif you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**wrefs** (*index=None, recurse=True*)

Returns a list of word references, these can be Words but also Morphemes or Phonemes.

**Parameters** **index** (*int or None*) – If set to an integer, will retrieve and return the n'th element (starting at 0) instead of returning the list of all

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```



```
__len__()
    Returns the number of child elements under the current element.

__str__()
    Alias for text()
```

## 4.2 Editing FoLiA

### 4.2.1 Creating a new document

Creating a new FoLiA document, rather than loading an existing one from file, is done by explicitly providing the ID for the new document in the *Document* constructor:

```
doc = folia.Document(id='example')
```

### 4.2.2 Declarations

Whenever you add a new **type** of annotation, or a different set, to a FoLiA document, you have to first declare it. This is done using the *Document.declare()* method. It takes as arguments the annotation type, the set, and you can optionally pass keyword arguments to *annotator=* and *annotatortype=* to set defaults.

An example for Part-of-Speech annotation:

```
doc.declare(folia.PosAnnotation, 'http://somewhere/brown-tag-set')
```

An example with a default annotator:

```
doc.declare(folia.PosAnnotation, 'http://somewhere/brown-tag-set', annotator='proycon
↪', annotatortype=folia.AnnotatorType.MANUAL)
```

Any additional sets for Part-of-Speech would have to be explicitly declared as well. To check if a particular annotation type and set is declared, use the *Document.declared()* method.

### 4.2.3 Adding structure

Assuming we begin with an empty document, we should first add a Text element. Then we can add paragraphs, sentences, or other structural elements. The *AbstractElement.add()* method adds new children to an element:

```
text = doc.add(folia.Text)
paragraph = text.add(folia.Paragraph)
sentence = paragraph.add(folia.Sentence)
sentence.add(folia.Word, 'This')
sentence.add(folia.Word, 'is')
sentence.add(folia.Word, 'a')
sentence.add(folia.Word, 'test')
sentence.add(folia.Word, '.')
```

**Note:** The *AbstractElement.add()* method is actually a wrapper around *AbstractElement.append()*, which takes the exact same arguments. It performs extra checks and works for both span annotation as well as token annotation. Using *append()* will be faster though.

## 4.2.4 Adding annotations

Adding annotations, or any elements for that matter, is done using the `AbstractElement.add()` method on the intended parent element. We assume that the annotations we add have already been properly declared, otherwise an exception will be raised as soon as `add()` is called. Let's build on the previous example:

```
#First we grab the fourth word, 'test', from the sentence
word = sentence.words(3)

#Add Part-of-Speech tag
word.add(folia.PosAnnotation, set='brown-tagset', cls='n')

#Add lemma
lemma.add(folia.LemmaAnnotation, cls='test')
```

Note that in the above examples, the `add()` method takes a class as first argument, and subsequently takes keyword arguments that will be passed to the classes' constructor.

A second way of using `AbstractElement.add()` is by simply passing a fully instantiated child element, thus constructing it prior to adding. The following is equivalent to the above example, as the previous method is merely a shortcut for convenience:

```
#First we grab the fourth word, 'test', from the sentence
word = sentence.words(3)

#Add Part-of-Speech tag
word.add( folia.PosAnnotation(doc, set='brown-tagset', cls='n') )

#Add lemma
lemma.add( folia.LemmaAnnotation(doc, cls='test') )
```

The `AbstractElement.add()` method always returns that which was added, allowing it to be chained.

In the above example we first explicitly instantiate a `PosAnnotation` and a `LemmaAnnotation`. Instantiation of any FoLiA element (always Python class subclassed off `AbstractElement`) follows the following pattern:

```
Class(document, *children, **kwargs)
```

---

**Note:** See `AbstractElement.__init__()` for all details on construction

---

Note that the document has to be passed explicitly as first argument to the constructor.

The common attributes are set using equally named keyword arguments:

- `id=`
- `cls=`
- `set=`
- `annotator=`
- `annotatortype=`
- `confidence=`
- `src=`
- `speaker=`

- `begin_time=`
- `end_time=`

Not all attributes are allowed for all elements, and certain attributes are required for certain elements. `ValueError` exceptions will be raised when these constraints are not met.

Instead of setting `id`, you can also set the keyword argument `generate_id_in` and pass it another element, an ID will be automatically generated, based on the ID of the element passed. When you use the first method of adding elements, instantiation with `generate_id_in` will take place automatically behind the scenes when applicable and when `id` is not explicitly set.

Any extra non-keyword arguments should be FoLiA elements and will be appended as the contents of the element, i.e. the children or subelements. Instead of using non-keyword arguments, you can also use the keyword argument `content` and pass a list. This is a shortcut made merely for convenience, as Python obliges all non-keyword arguments to come before the keyword-arguments, which is often aesthetically unpleasing for our purposes. Example of this use case will be shown in the next section.

## 4.2.5 Adding span annotation

Adding span annotation is easy with the FoLiA library. As you know, span annotation uses a stand-off annotation embedded in annotation layers. These layers are in turn embedded in structural elements such as sentences. However, the `AbstractElement.add()` method abstracts over this. Consider the following example of a named entity:

```
doc.declare(folia.Entity, "https://raw.githubusercontent.com/proycon/folia/master/
↳setdefinitions/namedentities.foliaset.xml")

sentence = text.add(folia.Sentence)
sentence.add(folia.Word, 'I', id='example.s.1.w.1')
sentence.add(folia.Word, 'saw', id='example.s.1.w.2')
sentence.add(folia.Word, 'the', id='example.s.1.w.3')
word = sentence.add(folia.Word, 'Dalai', id='example.s.1.w.4')
word2 = sentence.add(folia.Word, 'Lama', id='example.s.1.w.5')
sentence.add(folia.Word, '.', id='example.s.1.w.6')

word.add(folia.Entity, word, word2, cls="per")
```

To make references to the words, we simply pass the word instances and use the document's index to obtain them. Note also that passing a list using the keyword argument `contents` is wholly equivalent to passing the non-keyword arguments separately:

```
word.add(folia.Entity, cls="per", contents=[word, word2])
```

In the next example we do things more explicitly. We first create a sentence and then add a syntax parse, consisting of nested elements:

```
doc.declare(folia.SyntaxLayer, 'some-syntax-set')

sentence = text.add(folia.Sentence)
sentence.add(folia.Word, 'The', id='example.s.1.w.1')
sentence.add(folia.Word, 'boy', id='example.s.1.w.2')
sentence.add(folia.Word, 'pets', id='example.s.1.w.3')
sentence.add(folia.Word, 'the', id='example.s.1.w.4')
sentence.add(folia.Word, 'cat', id='example.s.1.w.5')
sentence.add(folia.Word, '.', id='example.s.1.w.6')

#Adding Syntax Layer
```

(continues on next page)

(continued from previous page)

```

layer = sentence.add(folia.SyntaxLayer)

#Adding Syntactic Units
layer.add(
    folia.SyntacticUnit(self.doc, cls='s', contents=[
        folia.SyntacticUnit(self.doc, cls='np', contents=[
            folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.1'], cls='det'),
            folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.2'], cls='n'),
        ]),
        folia.SyntacticUnit(self.doc, cls='vp', contents=[
            folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.3'], cls='v')
            folia.SyntacticUnit(self.doc, cls='np', contents=[
                folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.4'], cls=
↪ 'det'),
                folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.5'], cls='n
↪ '),
            ]),
        ]),
        folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.6'], cls='fin')
    ])
)

```

**Note:** The lower-level `AbstractElement.append()` method would have had the same effect in the above syntax tree sample.

## 4.2.6 Deleting annotations

Any element can be deleted by calling the `AbstractElement.remove()` method on its parent. Suppose we want to delete word:

```
word.parent.remove(word)
```

## 4.2.7 Copying annotations

A *deep copy* can be made of any element by calling its `AbstractElement.copy()` method:

```
word2 = word.copy()
```

The copy will be without parent and document. If you intend to associate a copy with a new document, then copy as follows instead:

```
word2 = word.copy(newdoc)
```

If you intend to attach the copy somewhere in the same document, you may want to add a suffix for any identifiers in its scope, since duplicate identifiers are not allowed and would raise an exception. This can be specified as the second argument:

```
word2 = word.copy(doc, ".copy")
```

## 4.3 Searching in a FoLiA document

If you have loaded a FoLiA document into memory, you may want to search for a particular annotations. You can of course loop over all structural and annotation elements using `AbstractElement.select()`, `AllowTokenAnnotation.annotation()` and `AllowTokenAnnotation.annotations()`. Additionally, `Word.findspans()` and `AbstractAnnotationLayer.findspan()` are useful methods of finding span annotations covering particular words, whereas `AbstractSpanAnnotation.wrefs()` does the reverse and finds the words for a given span annotation element. In addition to these main methods of navigation and selection, there is higher-level function available for searching, this uses the **FoLiA Query Language (FQL)** or the **Corpus Query Language (CQL)**.

These two languages are part of separate libraries that need to be imported:

```
from pynlpl.formats import fql, cql
```

### 4.3.1 Corpus Query Language (CQL)

CQL is the easier-language of the two and most suitable for corpus searching. It is, however, less flexible than FQL, which is designed specifically for FoLiA and can not just query, but also manipulate FoLiA documents in great detail.

CQL was developed for the **IMS Corpus Workbench**, at Stuttgart Univeristy, and is implemented in Sketch Engine, who provide good [CQL documentation](#).

CQL has to be converted to FQL first, which is then executed on the given document. This is a simple example querying for the word “house”:

```
doc = folia.Document(file="/path/to/some/document.folia.xml")
query = fql.Query(cql.cql2fql('"house"'))
for word in query(doc):
    print(word) #these will be folia.Word instances (all matching house)
```

Multiple words can be queried:

```
query = fql.Query(cql.cql2fql('"the" "big" "house"'))
for word1, word2, word3 in query(doc):
    print(word1, word2, word3)
```

Queries may contain wildcard expressions to match multiple text patterns. Gaps can be specified using `[]`. The following will match any three word combination starting with the and ending with something that starts with house. It will thus match things like “the big house” or “the small household”:

```
query = fql.Query(cql.cql2fql('"the" [] "house.*"'))
for word1, word2, word3 in query(doc):
    ...
```

We can make the gap optional with a question mark, it can be lengthened with `+` or `*`, like regular expressions:

```
query = fql.Query(cql.cql2fql('"the" []? "house.*"'))
for match in query(doc):
    print("We matched ", len(match), " words")
```

Querying is not limited to text, but all of FoLiA’s annotations can be used. To force our gap consist of one or more adjectives, we do:

```
query = fql.Query(cql.cql2fql('"the" [ pos = "a" ]+ "house.*"'))
for match in query(doc):
    ...
```

The original CQL attribute here is `tag` rather than `pos`, this can be used too. In addition, all FoLiA element types can be used! Just use their FoLiA tagname.

Consult the CQL documentation for more. Do note that CQL is very word/token centered, for searching other types of elements, use FQL instead.

### 4.3.2 FoLiA Query Language (FQL)

FQL is documented [here](#), a full overview is beyond the scope of this documentation. We will just introduce some basic selection queries so you can develop an initial impression of the language's abilities.

All FQL processing is done via the following class, as already seen in the previous section:

---

*Query*

---

This class represents an FQL query.

---

#### `pynlpl.formats.fql.Query`

**class** `pynlpl.formats.fql.Query` (*q*, *context*=<*pynlpl.formats.fql.Context* object>)

Bases: `object`

This class represents an FQL query.

Selecting a word with a particular text is done as follows, `doc` is an instance of *pynlpl.formats.folia.Document*:

```
query = fql.Query('SELECT w WHERE text = "house"')
for word in query(doc):
    print(word)  #this will be an instance of folia.Word
```

Regular expression matching can be done using the `MATCHES` operator:

```
query = fql.Query('SELECT w WHERE text MATCHES "^house.*$"')
for word in query(doc):
    print(word)
```

The classes of other annotation types can be easily queried as follows:

```
query = fql.Query('SELECT w WHERE :pos = "v" AND :lemma = "be"')
for word in query(doc):
    print(word)
```

You can constrain your queries to a particular target selection using the `FOR` keyword:

```
query = fql.Query('SELECT w WHERE text MATCHES "^house.*$" FOR s WHERE text_
↪CONTAINS "sell"')
for word in query(doc):
    print(word)
```

This construction also allows you to select the actual annotations. To select all people (a named entity) for words that are not John:

```
query = fql.Query('SELECT entity WHERE class = "person" FOR w WHERE text != "John"
↪')
for entity in query(doc):
    print(entity) #this will be an instance of folia.Entity
```

**FOR** statement may be chained, and Explicit IDs can be passed using the ID keyword:

```
query = fql.Query('SELECT entity WHERE class = "person" FOR w WHERE text != "John"
↪" FOR div ID "section.21"')
for entity in query(doc):
    print(entity)
```

Sets are specified using the **OF** keyword, it can be omitted if there is only one for the annotation type, but will be required otherwise:

```
query = fql.Query('SELECT su OF "http://some/syntax/set" WHERE class = "np"')
for su in query(doc):
    print(su) #this will be an instance of folia.SyntacticUnit
```

We have just covered just the **SELECT** keyword, FQL has other keywords for manipulating documents, such as **EDIT**, **ADD**, **APPEND** and **PREPEND**.

**Note:** Consult the FQL documentation at <https://github.com/proycon/foliadocserve/blob/master/README.rst> for further documentation on the language.

## Method Summary

<code>__init__(q[, context])</code>	Initialize self.
<code>parse(q[, i])</code>	

## Method Details

`__init__(q, context=<pynlpl.formats.fql.Context object>)`  
Initialize self. See help(type(self)) for accurate signature.

`__init__(q, context=<pynlpl.formats.fql.Context object>)`  
Initialize self. See help(type(self)) for accurate signature.

`parse(q, i=0)`

Selecting a word with a particular text is done as follows:

```
query = fql.Query('SELECT w WHERE text = "house"')
for word in query(doc):
    print(word) #this will be an instance of folia.Word
```

Regular expression matching can be done using the **MATCHES** operator:

```
query = fql.Query('SELECT w WHERE text MATCHES "^house.*$"')
for word in query(doc):
    print(word)
```

The classes of other annotation types can be easily queried as follows:

```
query = fql.Query('SELECT w WHERE :pos = "v" AND :lemma = "be"')
for word in query(doc):
    print(word)
```

You can constrain your queries to a particular target selection using the `FOR` keyword:

```
query = fql.Query('SELECT w WHERE text MATCHES "^house.*$" FOR s WHERE text CONTAINS
↪ "sell"')
for word in query(doc):
    print(word)
```

This construction also allows you to select the actual annotations. To select all people (a named entity) for words that are not John:

```
query = fql.Query('SELECT entity WHERE class = "person" FOR w WHERE text != "John"')
for entity in query(doc):
    print(entity) #this will be an instance of folia.Entity
```

`FOR` statement may be chained, and Explicit IDs can be passed using the `ID` keyword:

```
query = fql.Query('SELECT entity WHERE class = "person" FOR w WHERE text != "John"
↪ FOR div ID "section.21"')
for entity in query(doc):
    print(entity)
```

Sets are specified using the `OF` keyword, it can be omitted if there is only one for the annotation type, but will be required otherwise:

```
query = fql.Query('SELECT su OF "http://some/syntax/set" WHERE class = "np"')
for su in query(doc):
    print(su) #this will be an instance of folia.SyntacticUnit
```

We have just covered the `SELECT` keyword, FQL has other keywords for manipulating documents, such as **EDIT**, **ADD**, **APPEND** and **PREPEND**.

---

**Note:** Consult the FQL documentation at <https://github.com/proycon/foliadocserve/blob/master/README.rst> for further documentation on the language.

---

### 4.3.3 Streaming Reader

Throughout this tutorial you have seen the `Document` class as a means of reading FoLiA documents. This class always loads the entire document in memory, which can be a considerable resource demand. The following class provides an alternative to loading FoLiA documents:

---

<code>Reader</code>	Streaming FoLiA reader.
---------------------	-------------------------

---

#### `pynlpl.formats.folia.Reader`

```
class pynlpl.formats.folia.Reader(filename, target, *args, **kwargs)
    Bases: object
    Streaming FoLiA reader.
```



The reader allows you to read a FoLiA Document without holding the whole tree structure in memory. The document will be read and the elements you seek returned as they are found. If you are querying a corpus of large FoLiA documents for a specific structure, then it is strongly recommend to use the Reader rather than the standard Document!

## Method Summary

<code>__init__(filename, target, *args, **kwargs)</code>	Read a FoLiA document in a streaming fashion.
<code>findwords(*args, **kwargs)</code>	
<code>initdoc()</code>	

## Method Details

`__init__(filename, target, *args, **kwargs)`

Read a FoLiA document in a streaming fashion. You select a specific target element and all occurrences of this element, including all contents (so all elements within), will be returned.

### Parameters

- **filename** (\*) – The filename of the document to read
- **target** (\*) – The FoLiA element(s) you want to read (with everything contained in its scope). Passed as a class. For example: `folia.Sentence`, or a tuple of multiple element classes. Can also be set to `None` to return all elements, but that would load the full tree structure into memory.

`__init__(filename, target, *args, **kwargs)`

Read a FoLiA document in a streaming fashion. You select a specific target element and all occurrences of this element, including all contents (so all elements within), will be returned.

### Parameters

- **filename** (\*) – The filename of the document to read
- **target** (\*) – The FoLiA element(s) you want to read (with everything contained in its scope). Passed as a class. For example: `folia.Sentence`, or a tuple of multiple element classes. Can also be set to `None` to return all elements, but that would load the full tree structure into memory.

`findwords(*args, **kwargs)`

`initdoc()`

It does not load the entire document in memory but merely returns the elements you are interested in. This results in far less memory usage and also provides a speed-up.

A reader is constructed as follows, the second argument is the class of the element you want:

```
reader = folia.Reader("my.folia.xml", folia.Word)
for word in reader:
    print(word.id)
```

## 4.4 Higher-Order Annotations

### 4.4.1 Text Markup

FoLiA has a number of text markup elements, these appear within the *TextContent* (t) element, iterating over the element of a *TextContent* element will first and foremost produce strings, but also uncover these markup elements when present. The following markup types exists:

<i>TextMarkupGap</i>	Markup element to mark gaps in text content ( <i>TextContent</i> )
<i>TextMarkupString</i>	Markup element to mark arbitrary substrings in text content ( <i>TextContent</i> )
<i>TextMarkupStyle</i>	Markup element to style text content ( <i>TextContent</i> ), e.g.
<i>TextMarkupCorrection</i>	Markup element to mark corrections in text content ( <i>TextContent</i> ).
<i>TextMarkupError</i>	Markup element to mark gaps in text content ( <i>TextContent</i> )

#### `pynlpl.formats.folia.TextMarkupGap`

**class** `pynlpl.formats.folia.TextMarkupGap` (*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractTextMarkup`

Markup element to mark gaps in text content (*TextContent*)

Only consider this element for gaps in spans of untokenised text. The use of structural element *Gap* is preferred.

#### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	See <code>AbstractElement.__init__()</code> , text is passed as a string in <i>*args</i> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.

Continued on next page

Table 80 – continued from previous page

<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	See <code>AbstractElement.json()</code>
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve()</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.

Continued on next page

Table 80 – continued from previous page

<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text)</code>	Sets the text content of the markup element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = 24
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None

```

```
TEXTCONTAINER = True
```

```
TEXTDELIMITER = ''
```

```
XLINK = True
```

```
XMLTAG = 't-gap'
```

## Method Details

```
__init__(doc, *args, **kwargs)
```

See [AbstractElement.\\_\\_init\\_\\_\(\)](#), text is passed as a string in *\*args*.

```
__init__(doc, *args, **kwargs)
```

See [AbstractElement.\\_\\_init\\_\\_\(\)](#), text is passed as a string in *\*args*.

```
classmethod accepts (Class, raiseexceptions=True, parentinstance=None)
```

```
add (child, *args, **kwargs)
```

```
classmethod addable (parent, set=None, raiseexceptions=True)
```

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** ([AbstractElement](#)) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** `ValueError`

```
addidsuffix (idsuffix, recursive=True)
```

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by [copy\(\)](#)

```
addtoindex (norecurse=[])
```

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

```
ancestor (*Classes)
```

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes ([AbstractElement](#) or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

```
ancestors (Class=None)
```

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes ([AbstractElement](#) or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, \**args*, \*\**kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

See `AbstractElement.json()`

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:



```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \*args, \*\*kwargs)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolve** ()

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↪folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text)**

Sets the text content of the markup element.

**Parameters** `text` (*str*) –

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls*='current')

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs*=None, *elements*=None, *skipchildren*=False)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print*=False)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

`__str__()`  
Alias for `text()`

## `pynlpl.formats.folia.TextMarkupString`

**class** `pynlpl.formats.folia.TextMarkupString` (*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractTextMarkup`

Markup element to mark arbitrary substrings in text content (*TextContent*)

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	See <code>AbstractElement.__init__()</code> , <code>text</code> is passed as a string in <code>*args</code> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)

Continued on next page

Table 81 – continued from previous page

<i>incorrection()</i>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	See <i>AbstractElement.json()</i>
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolve</i> ()	
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text)	Sets the text content of the markup element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.
<i>toktext</i> ([cls])	Alias for <i>text()</i> with <i>retaintokenisation=True</i>

Continued on next page

Table 81 – continued from previous page

<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = 32
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = True
TEXTDELIMITER = ''
XLINK = True
XMLTAG = 't-str'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    See AbstractElement.__init__(), text is passed as a string in *args.

__init__(doc, *args, **kwargs)
    See AbstractElement.__init__(), text is passed as a string in *args.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

```

**classmethod** **addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, *\*args*, *\*\*kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element



**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** DeepValidationError

**description** ()

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls*='current', *strict*=*True*, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=*None*, *recurse*=*True*, *ignorelist*=*False*)

See `AbstractElement.json()`

**leftcontext** (*size*, *placeholder*=*None*, *scope*=*None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class*=*True*, *scope*=*True*, *reverse*=*False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolve()**

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*)

Sets the text content of the markup element.

**Parameters** **text** (*str*) –

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.TextMarkupStyle

**class** `pynlpl.formats.folia.TextMarkupStyle` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractTextMarkup`

Markup element to style text content (`TextContent`), e.g. make text bold, italics, underlined, coloured, etc..

## Method Summary

---

`__init__` (*doc, \*args, \*\*kwargs*)

See `AbstractElement.__init__()`, text is passed as a string in *\*args*.

---

Continued on next page

Table 82 – continued from previous page

<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attrs, recurse, ignorelist])</code>	See <code>AbstractElement.json()</code>
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.

Continued on next page



Table 82 – continued from previous page

<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve()</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text)</code>	Sets the text content of the markup element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = 34
AUTH = True

```

```
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = True
TEXTDELIMITER = ''
XLINK = True
XMLTAG = 't-style'
```

## Method Details

`__init__(doc, *args, **kwargs)`

See [`AbstractElement.\_\_init\_\_\(\)`](#), text is passed as a string in `*args`.

`__init__(doc, *args, **kwargs)`

See [`AbstractElement.\_\_init\_\_\(\)`](#), text is passed as a string in `*args`.

`classmethod accepts(Class, raiseexceptions=True, parentinstance=None)`

`add(child, *args, **kwargs)`

`classmethod addable(parent, set=None, raiseexceptions=True)`

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** ([`AbstractElement`](#)) – The element that is being added to
- **set** (`str` or `None`) – The set
- **raiseexceptions** (`bool`) – Raise an exception if the element can't be added?

**Returns** `bool`

**Raises** `ValueError`

`addidsuffix(idsuffix, recursive=True)`

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by [`copy\(\)`](#)

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.

- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

See `AbstractElement.json()`

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (cls='current', previousdelimiter=",", strict=False, correctionhandling=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (str) – The class of the phonetic content to obtain, defaults to current.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolve** ()

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.

- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*)

Sets the text content of the markup element.

**Parameters** *text* (*str*) –

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *text()*. Defaults to an empty string.



- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for *text* ()

## pynlpl.formats.folia.TextMarkupCorrection

**class** pynlpl.formats.folia.**TextMarkupCorrection** (*doc, \*args, \*\*kwargs*)

Bases: *pynlpl.formats.folia.AbstractTextMarkup*

Markup element to mark corrections in text content (*TextContent*).

Only consider this element for corrections on untokenised text. The use of *Correction* is preferred.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	See <i>AbstractElement.__init__()</i> , text is passed as a string in <i>*args</i> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.

Continued on next page

Table 83 – continued from previous page

<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	See <code>AbstractElement.json()</code>
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve()</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.

Continued on next page

Table 83 – continued from previous page

<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text)</code>	Sets the text content of the markup element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = 16
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None

```

```

TEXTCONTAINER = True
TEXTDELIMITER = ''
XLINK = True
XMLTAG = 't-correction'

```

## Method Details

`__init__(doc, *args, **kwargs)`

See `AbstractElement.__init__()`, text is passed as a string in `*args`.

`__init__(doc, *args, **kwargs)`

See `AbstractElement.__init__()`, text is passed as a string in `*args`.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, *\*args*, *\*\*kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** `ValueError`

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** *\*Classes* – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** *\*Class* – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, \**args*, \*\**kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, \*\**kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

#### Returns

`bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

See `AbstractElement.json()`

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:



```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolve** ()

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text)**

Sets the text content of the markup element.

**Parameters** `text` (*str*) –

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls*='current')

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs*=None, *elements*=None, *skipchildren*=False)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print*=False)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

`__str__()`  
Alias for `text()`

## `pynlpl.formats.folia.TextMarkupError`

**class** `pynlpl.formats.folia.TextMarkupError` (*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractTextMarkup`

Markup element to mark gaps in text content (*TextContent*)

Only consider this element for gaps in spans of untokenised text. The use of structural element *ErrorDetection* is preferred.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	See <code>AbstractElement.__init__()</code> , <code>text</code> is passed as a string in <code>*args</code> .
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.

Continued on next page

Table 84 – continued from previous page

<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	See <i>AbstractElement.json()</i>
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <i>append()</i> , but replaces any existing child element of the same type and set.
<i>resolve</i> ()	
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text)	Sets the text content of the markup element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <i>text()</i> with <i>strict=True</i>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)
<i>textcontent</i> ([cls, correctionhandling])	Get the text content explicitly associated with this element (of the specified class).
<i>textvalidation</i> ([warnonly])	Run text validation on this element.

Continued on next page

Table 84 – continued from previous page

<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retain_tokenisation=True</code>
<code>update_text()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	See <code>AbstractElement.xml()</code>
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTextMarkup'>, <class 'pynlpl.formats.folia.AbstractTextMarkup'>)
ANNOTATIONTYPE = 17
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = True
TEXTDELIMITER = ''
XLINK = True
XMLTAG = 't-error'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    See AbstractElement.__init__(), text is passed as a string in *args.

__init__(doc, *args, **kwargs)
    See AbstractElement.__init__(), text is passed as a string in *args.

classmethod accepts(Class, raise_exceptions=True, parent_instance=None)

```

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool*

**Raises** *ValueError*

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, \**args*, \*\**kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element



**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls*='current', *strict*=*True*, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=*None*, *recurse*=*True*, *ignorelist*=*False*)

See `AbstractElement.json()`

**leftcontext** (*size*, *placeholder*=*None*, *scope*=*None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class*=*True*, *scope*=*True*, *reverse*=*False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=",", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolve()**

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*)

Sets the text content of the markup element.

**Parameters** **text** (*str*) –

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

See `AbstractElement.xml()`

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## 4.4.2 Features

Features allow a second-order annotation by adding the ability to assign properties and values to any of the existing annotation elements. They follow the set/class paradigm by adding the notion of a subset and class relative to this subset. The `AbstractElement.feats()` method provides a shortcut that can be used on any annotation element to obtain the class of the feature, given a subset. To illustrate the concept, take a look at part of speech annotation with some features:

```
pos = word.annotation(folia.PosAnnotation)
if pos.cls == "n":
    if pos.feats('number') == 'plural':
        print("We have a plural noun!")
    elif pos.feats('number') == 'singular':
        print("We have a singular noun!")
```

The `AbstractElement.feats()` method will return an exception when the feature does not exist. Note that the actual subset and class values are defined by the set and not FoLiA itself! They are therefore fictitious in the above example.

The Python class for features is `Feature`, in the following example we add a feature:

```
pos.add(folia.Feature, subset="gender", cls="f")
```

Although FoLiA does not define any sets nor subsets. Some annotation types do come with some associated subsets, their use is never mandatory. The advantage is that these associated subsets can be directly used as an XML attribute in the FoLiA document. The FoLiA library provides extra classes, all subclassed off `Feature` for these:

<code>Feature</code>	Feature elements can be used to associate subsets and subclasses with almost any annotation element
<code>SynsetFeature</code>	Synset feature, to be used within <code>Sense</code>
<code>ActorFeature</code>	Actor feature, to be used within <code>Event</code>
<code>BegindatetimeFeature</code>	Begindatetime feature, to be used within <code>Event</code>
<code>EnddatetimeFeature</code>	Enddatetime feature, to be used within <code>Event</code>

## pynlpl.formats.folia.Feature

**class** `pynlpl.formats.folia.Feature` (*doc*, \*args, \*\*kwargs)

Bases: `pynlpl.formats.folia.AbstractElement`

Feature elements can be used to associate subsets and subclasses with almost any annotation element

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Constructor.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.

Continued on next page



Table 86 – continued from previous page

<i>description()</i>	Obtain the description associated with the element.
<i>feat(subset)</i>	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling(cls)</i>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables(parent[, set])</i>	Internal method to find replaceable elements.
<i>getindex(child[, recursive, ignore])</i>	Get the index at which an element occurs, recursive by default!
<i>getmetadata([key])</i>	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter([retaintokenisation])</i>	Return the text delimiter for this class.
<i>hasphon([cls, strict, correctionhandling])</i>	Does this element have phonetic content (of the specified class)
<i>hastext([cls, strict, correctionhandling])</i>	Does this element have text (of the specified class)
<i>incorrection()</i>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert(index, child, *args, **kwargs)</i>	
<i>items([founditems])</i>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json([attribs, recurse, ignorelist])</i>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext(size[, placeholder, scope])</i>	Returns the left context for an element, as a list.
<i>next([Class, scope, reverse])</i>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext([cls])</i>	Alias for retrieving the original uncorrect text.
<i>parsexml(node, doc, **kwargs)</i>	Internal class method used for turning an XML element into an instance of the Class.
<i>phon([cls, previousdelimiter, strict, ...])</i>	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent([cls, correctionhandling])</i>	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend()</i>	This method will be called after an element is added to another and does some checks.
<i>previous([Class, scope])</i>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng([includechildren, extraattribs, ...])</i>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove(child)</i>	Removes the child element
<i>replace(child, *args, **kwargs)</i>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword(id)</i>	
<i>rightcontext(size[, placeholder, scope])</i>	Returns the right context for an element, as a list.
<i>select(Class[, set, recursive, ignore, node])</i>	Select child elements of the specified class.
<i>setdoc(newdoc)</i>	Set a different document.
<i>setdocument(doc)</i>	Associate a document with this element.
<i>setparents()</i>	Correct all parent relations for elements within the scop.

Continued on next page

Table 86 – continued from previous page

<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml()</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Feature'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
```

**TEXTDELIMITER = None**

**XLINK = False**

**XMLTAG = 'feat'**

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.



**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml ()**

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring (pretty\_print=False)**

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.SynsetFeature**

**class** `pynlpl.formats.folia.SynsetFeature (doc, *args, **kwargs)`

Bases: `pynlpl.formats.folia.Feature`

Synset feature, to be used within `Sense`

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Constructor.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 87 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 87 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml()</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Feature'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False

```

```
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = 'synset'
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None
```

## Method Details

`__init__(doc, *args, **kwargs)`  
Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

`__init__(doc, *args, **kwargs)`  
Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**classmethod** `accepts` (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod** `addable` (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** `bool`

**Raises** `ValueError`

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feats('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.



- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (index, child, \*args, \*\*kwargs)

**items** (founditems=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (attribs=None, recurse=True, ignorelist=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (size, placeholder=None, scope=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (Class=True, scope=True, reverse=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement', may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (cls='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (node, doc, \*\*kwargs)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

See also:

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.

- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for *text()* with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retain_tokenisation=True`

**update\_text** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** ()

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.ActorFeature**

**class** `pynlpl.formats.folia.ActorFeature` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.Feature`

Actor feature, to be used within *Event*

### **Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Constructor.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.

Continued on next page

Table 88 – continued from previous page

<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml()</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

`ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.fo`



```

ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Feature'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = 'actor'
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None

```

## Method Details

`__init__(doc, *args, **kwargs)`

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

`__init__(doc, *args, **kwargs)`

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool***Raises** *ValueError***addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, *\*args*, *\*\*kwargs*)**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation()**

Perform deep validation of this element.

**Raises** DeepValidationError

**description()**

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod** `parsexml (node, doc, **kwargs)`

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattribs=None*, *extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls='current'*)

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)



**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** ()

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.BegindatetimeFeature**

**class** `pynlpl.formats.folia.BegindatetimeFeature` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.Feature`

Begindatetime feature, to be used within `Event`

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Constructor.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 89 – continued from previous page

<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml()</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.fo
```

```
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Feature'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = 'begindatetime'
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None
```

## Method Details

`__init__(doc, *args, **kwargs)`  
Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

`__init__(doc, *args, **kwargs)`  
Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool**Raises** ValueError**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation()**

Perform deep validation of this element.

**Raises** DeepValidationError

**description()**

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod** `parsexml (node, doc, **kwargs)`

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.



- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** ()

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.EnddatetimeFeature**

**class** `pynlpl.formats.folia.EnddatetimeFeature` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.Feature`

Enddatetime feature, to be used within `Event`

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Constructor.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 90 – continued from previous page

<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml()</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.fo
```

```

ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Feature'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = 'enddatetime'
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = None

```

## Method Details

`__init__(doc, *args, **kwargs)`

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

`__init__(doc, *args, **kwargs)`

Constructor.

### Keyword Arguments

- **subset** (*str*) – the subset
- **cls** (*str*) – the class

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool***Raises** *ValueError***addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.



**Returns** int

**deepvalidation()**

Perform deep validation of this element.

**Raises** DeepValidationError

**description()**

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod** `parsexml (node, doc, **kwargs)`

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*=", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

*phon()* *textcontent()* *text()*

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True*, *scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True*, *extraattribs=None*, *extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document’s FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** ()

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

### 4.4.3 Alternatives

A key feature of FoLiA is its ability to make explicit alternative annotations, for token annotations, the `Alternative` (`alt`) class is used to this end. Alternative annotations are embedded in this structure. This implies the annotation is not authoritative, but is merely an alternative to the actual annotation (if any). Alternatives may typically occur in larger numbers, representing a distribution each with a confidence value (not mandatory). Each alternative is wrapped in its own `Alternative` element, as multiple elements inside a single alternative are considered dependent and part

of the same alternative. Combining multiple annotation in one alternative makes sense for mixed annotation types, where for instance a pos tag alternative is tied to a particular lemma:

```
alt = word.add(folia.Alternative)
alt.add(folia.PosAnnotation, set='brown-tagset', cls='n', confidence=0.5)
alt = word.add(folia.Alternative) #note that we reassign the variable!
alt.add(folia.PosAnnotation, set='brown-tagset', cls='a', confidence=0.3)
alt = word.add(folia.Alternative)
alt.add(folia.PosAnnotation, set='brown-tagset', cls='v', confidence=0.2)
```

Span annotation elements have a different mechanism for alternatives, for those the entire annotation layer is embedded in a *AlternativeLayers* element. This element should be repeated for every type, unless the layers it describes are dependent on it eachother:

```
alt = sentence.add(folia.AlternativeLayers)
layer = alt.add(folia.Entities)
entity = layer.add(folia.Entity, word1, word2, cls="person", confidence=0.3)
```

Because the alternative annotations are **non-authoritative**, normal selection methods such as `select()` and `annotations()` will never yield them, unless explicitly told to do so. For this reason, there is an `alternatives()` method on structure elements, for the first category of alternatives.

In summary, a list of the two relevant classes for alternatives:

<i>Alternative</i>	Element grouping alternative token annotation(s).
<i>AlternativeLayers</i>	Element grouping alternative subtoken annotation(s).

## pynlpl.formats.folia.Alternative

**class** pynlpl.formats.folia.**Alternative** (*doc*, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractElement*, *pynlpl.formats.folia.AllowTokenAnnotation*, *pynlpl.formats.folia.AllowGenerateID*

Element grouping alternative token annotation(s).

Multiple alternative elements may occur, each denoting a different alternative. Elements grouped inside an alternative block are considered dependent.

A key feature of FoLiA is its ability to make explicit alternative annotations, for token annotations, this class is used to this end. Alternative annotations are embedded in this structure. This implies the annotation is *not authoritative*, but is merely an alternative to the actual annotation (if any). Alternatives may typically occur in larger numbers, representing a distribution each with a confidence value (not mandatory). Each alternative is wrapped in its an instance of this class, as multiple elements inside a single alternative are considered dependent and part of the same alternative. Combining multiple annotation in one alternative makes sense for mixed annotation types, where for instance a pos tag alternative is tied to a particular lemma.

## Method Summary

<i>__init__</i> ( <i>doc</i> , *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.

Continued on next page



Table 92 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>alternatives([Class, set])</code>	Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>annotation(type[, set])</code>	Obtain a single annotation element.
<code>annotations(Class[, set])</code>	Obtain child elements (annotations) of the specified class.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	Apply a correction (TODO: documentation to be written still)
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasannotation(Class[, set])</code>	Returns an integer indicating whether such as annotation exists, and if so, how many.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Continued on next page

Table 92 – continued from previous page

<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.

Continued on next page

Table 92 – continued from previous page

<code>__str__()</code>	Alias for <code>text()</code>
------------------------	-------------------------------

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractTokenAnnotation'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = False
AUTO_GENERATE_ID = False
LABEL = 'Alternative'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'alt'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.

```

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**alternatives** (*Class=None, set=None*)

Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

**Parameters**

- **Class** (*class*) – The python Class you want to retrieve (e.g. PosAnnotation). Or set to None to select all alternatives regardless of what type they are.
- **set** (*str*) – The set you want to retrieve (defaults to None, which selects irregardless of set)

**Yields** *Alternative* elements

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**annotation** (*type, set=None*)

Obtain a single annotation element.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.

**Returns** An element (instance derived from *AbstractElement*)

Example:

```
sense = word.annotation(folia.Sense, 'http://some/path/cornetto').cls
```

See also:

`AllowTokenAnnotation.annotations()` `AbstractElement.select()`

**Raises** `NoSuchAnnotation` if no such annotation exists

**annotations** (*Class*, *set=None*)

Obtain child elements (annotations) of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.annotations(folia.Sense, 'http://some/path/cornetto'):
    ..
```

See also:

`AbstractElement.select()`

**Raises**

- `AllowTokenAnnotation.annotations()`
- `NoSuchAnnotation` if no such annotation exists

**append** (*child*, *\*args*, *\*\*kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**correct** (*\*\*kwargs*)

Apply a correction (TODO: documentation to be written still)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasannotation** (*Class, set=None*)

Returns an integer indicating whether such as annotation exists, and if so, how many.

See `AllowTokenAnnotation.annotations`()` for a description of the parameters.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined `scope`. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.



**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `Listitem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattrs*=None, *extraelements*=None, *orig-class*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to True, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** **doc** (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updateetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

## pynlpl.formats.folia.AlternativeLayers

**class** pynlpl.formats.folia.**AlternativeLayers** (*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`

Element grouping alternative subtoken annotation(s). Multiple altlayers elements may occur, each denoting a different alternative. Elements grouped inside an alternative block are considered dependent.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!

Continued on next page

Table 93 – continued from previous page

<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<i>insert</i> (index, child, *args, **kwargs)	
<i>items</i> ([founditems])	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<i>json</i> ([attribs, recurse, ignorelist])	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<i>leftcontext</i> (size[, placeholder, scope])	Returns the left context for an element, as a list.
<i>next</i> ([Class, scope, reverse])	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>originaltext</i> ([cls])	Alias for retrieving the original uncorrect text.
<i>parsexml</i> (node, doc, **kwargs)	Internal class method used for turning an XML element into an instance of the Class.
<i>phon</i> ([cls, previousdelimiter, strict, ...])	Get the phonetic representation associated with this element (of the specified class)
<i>phoncontent</i> ([cls, correctionhandling])	Get the phonetic content explicitly associated with this element (of the specified class).
<i>postappend</i> ()	This method will be called after an element is added to another and does some checks.
<i>previous</i> ([Class, scope])	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<i>relaxng</i> ([includechildren, extraattribs, ...])	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<i>remove</i> (child)	Removes the child element
<i>replace</i> (child, *args, **kwargs)	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<i>resolveword</i> (id)	
<i>rightcontext</i> (size[, placeholder, scope])	Returns the right context for an element, as a list.
<i>select</i> (Class[, set, recursive, ignore, node])	Select child elements of the specified class.
<i>setdoc</i> (newdoc)	Set a different document.
<i>setdocument</i> (doc)	Associate a document with this element.
<i>setparents</i> ()	Correct all parent relations for elements within the scop.
<i>settext</i> (text[, cls])	Set the text for this element.
<i>speech_speaker</i> ()	Retrieves the speaker of the audio or video file associated with the element.
<i>speech_src</i> ()	Retrieves the URL/filename of the audio or video file associated with the element.
<i>stricttext</i> ([cls])	Alias for <code>text()</code> with <code>strict=True</code>
<i>text</i> ([cls, retaintokenisation, ...])	Get the text associated with this element (of the specified class)

Continued on next page

Table 93 – continued from previous page

<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractAnnotationLayer'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = False
AUTO_GENERATE_ID = False
LABEL = 'Alternative Layers'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'altlayers'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)

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

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** \***Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, \*args, \*\*kwargs)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.



**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrected text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter="*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchPhon – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** NoSuchPhon if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off 'AbstractElement'. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, \**args*, \*\**kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on *set*.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents()**

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext(text, cls='current')**

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext(cls='current')**

Alias for `text()` with `strict=True`

**text(cls='current', retaintokenisation=False, previousdelimiter="", strict=False, correctionhandling=1, normalize\_spaces=False)**

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, str in Python 3)

**Raises** NoSuchText – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** NoSuchText if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** bool

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

`__iter__()`  
Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

`__len__()`  
Returns the number of child elements under the current element.

`__str__()`  
Alias for `text()`

#### 4.4.4 Corrections

Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, token annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.

Despite their complexity, the library treats correction transparently. Whenever you query for a particular element, and it is part of a correction, you get the corrected version rather than the original. The original is always *non-authoritative* and normal selection methods will ignore it.

If you want to deal with correction, you have to explicitly handle the `Correction` element. If an element is part of a correction, its `AbstractElement.incorrection()` method will give the correction element, if not, it will return `None`:

```
pos = word.annotation(folia.PosAnnotation)
correction = pos.incorrection()
if correction:
    if correction.hasoriginal():
        originalpos = correction.original(0) #assuming it's the only element as is_
        ↪customary
        #originalpos will be an instance of folia.PosAnnotation
        print("The original pos was", originalpos.cls)
```

Corrections themselves carry a class too, indicating the type of correction (defined by the set used and not by FoLiA).

Besides `Correction.original()`, corrections distinguish three other types, `Correction.new()` (the corrected version), `Correction.current()` (the current uncorrected version) and `Correction.suggestions()` (a suggestion for correction), the former two and latter two usually form pairs, `current()` and `new()` can never be used together. Of `suggestions(index)` there may be multiple, hence the index argument. These return, respectively, instances of `Original`, `folia.New`, `folia.Current` and `folia.Suggestion`.

Adding a correction can be done explicitly:

```
wrongpos = word.annotation(folia.PosAnnotation)
word.add(folia.Correction, folia.New(doc, folia.PosAnnotation(doc, cls="n")) , folia.
        ↪Original(doc, wrongpos), cls="misclassified")
```

Let's settle for a suggestion rather than an actual correction:

```
wrongpos = word.annotation(folia.PosAnnotation)
word.add(folia.Correction, folia.Suggestion(doc, folia.PosAnnotation(doc, cls="n")),
        ↪cls="misclassified")
```



In some instances, when correcting text or structural elements, *New* may be empty, which would correspond to an *deletion*. Similarly, *Original* may be empty, corresponding to an *insertion*.

The use of *Current* is reserved for use with structure elements, such as words, in combination with suggestions. The structure elements then have to be embedded in *Current*. This situation arises for instance when making suggestions for a merge or split.

Here is a list of all relevant classes for corrections:

<i>Correction</i>	Corrections are one of the most complex annotation types in FoLiA.
<i>Current</i>	Used in the context of <i>Correction</i> to encapsulate the currently authoritative annotations.
<i>ErrorDetection</i>	The ErrorDetection element is used to signal the presence of errors in a structural element.
<i>New</i>	
<i>Original</i>	Used in the context of <i>Correction</i> to encapsulate the original annotations <i>prior</i> to correction.
<i>Suggestion</i>	Suggestions are used in the context of <i>Correction</i> , but rather than provide an authoritative correction, it instead offers a suggestion for correction.

## pynlpl.formats.folia.Correction

**class** pynlpl.formats.folia.**Correction** (*doc*, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractElement*, *pynlpl.formats.folia.AllowGenerateID*

Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, token annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.

Despite their complexity, the library treats correction transparently. Whenever you query for a particular element, and it is part of a correction, you get the corrected version rather than the original. The original is always *non-authoritative* and normal selection methods will ignore it.

**This class takes four classes as children, that in turn encapsulate the actual annotations:**

- *New* - Encapsulates the newly corrected annotation(s)
- *Original* - Encapsulated the old original annotation(s)
- *Current* - Encapsulates the current authoritative annotation(s)
- *Suggestions* - Encapsulates the annotation(s) that are a non-authoritative suggestion for correction

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.

Continued on next page

Table 95 – continued from previous page

<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element’s ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>current([index])</code>	Get the current authoritative annotation (used with suggestions in a structural context)
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	See <code>AbstractElement.gettextdelimiter()</code>
<code>hascurrent([allowempty])</code>	Does the correction record the current authoritative annotation (needed only in a structural context when suggestions are proposed)
<code>hasnew([allowempty])</code>	Does the correction define new corrected annotations?
<code>hasoriginal([allowempty])</code>	Does the correction record the old annotations prior to correction?
<code>hasphon([cls, strict, correctionhandling])</code>	See <code>AbstractElement.hasphon()</code>
<code>hassuggestions([allowempty])</code>	Does the correction propose suggestions for correction?
<code>hastext([cls, strict, correctionhandling])</code>	See <code>AbstractElement.hastext()</code>
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to <code>AbstractElement</code> )

Continued on next page

Table 95 – continued from previous page

<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>new([index])</code>	Get the new corrected annotation.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>original([index])</code>	Get the old annotation prior to correction.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	See <code>AbstractElement.phon()</code>
<code>phoncontent([cls, correctionhandling])</code>	See <code>AbstractElement.phoncontent()</code>
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>suggestions([index])</code>	Get suggestions for correction.
<code>text([cls, retaintokenisation, ...])</code>	See <code>AbstractElement.text()</code>
<code>textcontent([cls, correctionhandling])</code>	See <code>AbstractElement.textcontent()</code>
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.

Continued on next page

Table 95 – continued from previous page

<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.  
ANNOTATIONTYPE = 16  
AUTH = True  
AUTO_GENERATE_ID = False  
LABEL = 'Correction'  
OCCURRENCES = 0  
OCCURRENCES_PER_SET = 0  
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)  
PHONCONTAINER = False  
PRIMARYELEMENT = True  
PRINTABLE = True  
REQUIRED_ATTRIBS = None  
REQUIRED_DATA = None  
SETONLY = False  
SPEAKABLE = True  
SUBSET = None  
TEXTCONTAINER = False  
TEXTDELIMITER = None  
XLINK = False  
XMLTAG = 'correction'
```

### Method Details

```
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
  
__init__(doc, *args, **kwargs)  
    Initialize self. See help(type(self)) for accurate signature.  
  
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)  
  
add(child, *args, **kwargs)  
  
classmethod addable(parent, set=None, raiseexceptions=True)  
    Tests whether a new element of this class can be added to the parent.  
  
    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden  
    by subclasses for more customised behaviour.
```

**Parameters**

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** *bool***Raises** *ValueError***addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**current** (*index=None*)

Get the current authoritative annotation (used with suggestions in a structural context)

This returns only one annotation if multiple exist, use *index* to select another in the sequence.

**Returns** an annotation element (`AbstractElement`)

**Raises** `NoSuchAnnotation`

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

See `AbstractElement.gettextdelimiter()`

**hascurrent** (*allowempty=False*)

Does the correction record the current authoritative annotation (needed only in a structural context when suggestions are proposed)

**hasnew** (*allowempty=False*)

Does the correction define new corrected annotations?

**hasoriginal** (*allowempty=False*)

Does the correction record the old annotations prior to correction?

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

See [AbstractElement.hasphon\(\)](#)

**hassuggestions** (*allowempty=False*)

Does the correction propose suggestions for correction?

**hastext** (*cls='current', strict=True, correctionhandling=1*)

See [AbstractElement.hastext\(\)](#)

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**new** (*index=None*)

Get the new corrected annotation.

This returns only one annotation if multiple exist, use *index* to select another in the sequence.

**Returns** an annotation element ([AbstractElement](#))

**Raises** `NoSuchAnnotation`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '[AbstractElement](#)', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence, Paragraph, Division, Event, ListItem, Caption`), set to `None` to not constrain at all.

**original** (*index=None*)

Get the old annotation prior to correction.

This returns only one annotation if multiple exist, use *index* to select another in the sequence.

**Returns** an annotation element (*AbstractElement*)

**Raises** *NoSuchAnnotation*

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** - *XML Element* (\*) –
- **doc** - *Document* (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=", strict=False, correctionhandling=1*)

See *AbstractElement.phon()*

**phoncontent** (*cls='current', correctionhandling=1*)

See *AbstractElement.phoncontent()*

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –



See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text

- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls='current'*)

Alias for `text()` with `strict=True`

**suggestions** (*index=None*)

Get suggestions for correction.

**Yields** `Suggestion` element that encapsulate the suggested annotations (if `index` is `None`, default)

**Returns** a `Suggestion` element that encapsulate the suggested annotations (if `index` is set)

**Raises** `IndexError`

**text** (*cls='current', retaintokenisation=False, previousdelimiter=", strict=False, correctionhandling=1, normalize\_spaces=False*)  
See `AbstractElement.text()`

**textcontent** (*cls='current', correctionhandling=1*)

See `AbstractElement.textcontent()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (`True`) or raise exceptions (`False`). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

See also:

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** str

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.Current

**class** pynlpl.formats.folia.**Current** (*doc, \*args, \*\*kwargs*)

Bases: pynlpl.formats.folia.AbstractCorrectionChild

Used in the context of *Correction* to encapsulate the currently authoritative annotations.

Needed only when suggestions for correction are proposed (*Suggestion*) for structural elements.

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.

Continued on next page

Table 96 – continued from previous page

<code>correct(**kwargs)</code>	
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	

Continued on next page

Table 96 – continued from previous page

<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractSpanAnnotation'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False

```

```
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'current'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child*, *\*args*, *\*\*kwargs*)

**context** (*size*, *placeholder=None*, *scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None*, *idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str* or *bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None*, *idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

**count** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)



**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘*AbstractElement*’, may also be a tuple of multiple classes. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence, Paragraph, Division, Event, ListItem, Caption*), set to *None* to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling=CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter="", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.

- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls*='current')

Alias for `text()` with `retain_tokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## pynlpl.formats.folia.ErrorDetection

**class** `pynlpl.formats.folia.ErrorDetection` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractExtendedTokenAnnotation`

The `ErrorDetection` element is used to signal the presence of errors in a structural element.

### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	See <code>AbstractElement.append()</code>

Continued on next page

Table 97 – continued from previous page

<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element

Continued on next page

Table 97 – continued from previous page

<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.LiA'>)
ANNOTATIONTYPE = 17
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Error Detection'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False

```



```

REQUIRED_ATTRIBS = (1,)
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'errordetection'

```

### Method Details

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc, \*args, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class, raiseexceptions=True, parentinstance=None*)

**add** (*child, \*args, \*\*kwargs*)

**classmethod addable** (*parent, set=None, raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

See *AbstractElement.append()*

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls='current', correctionhandling=1*)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`



**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.New**

**class** `pynlpl.formats.folia.New` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractCorrectionChild`

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

Continued on next page

Table 98 – continued from previous page

<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>correct(**kwargs)</code>	
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 98 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractSpanAnnotation'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True

```

```
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'new'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**correct** (*\*\*kwargs*)

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.



**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `xml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.Original**

**class** `pynlpl.formats.folia.Original` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractCorrectionChild`

Used in the context of `Correction` to encapsulate the original annotations *prior* to correction.

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 99 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page

Table 99 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractSpanAnnotation'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = False
AUTO_GENERATE_ID = False
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True

```

```

PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'original'

```

### Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description** ()

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list



**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

**pynlpl.formats.folia.Suggestion**

**class** `pynlpl.formats.folia.Suggestion` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractCorrectionChild`

Suggestions are used in the context of `Correction`, but rather than provide an authoritative correction, it instead offers a suggestion for correction.

**Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.

Continued on next page

Table 100 – continued from previous page

<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>generate_id(cls)</code>	
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.

Continued on next page



Table 100 – continued from previous page

<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element ( <code>lxml.etree</code> ) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AbstractSpanAnnotation'>, <class 'pynlpl
ANNOTATIONTYPE = None
AUTH = False
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4)
PHONCONTAINER = False
PRIMARYELEMENT = True

```

```
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'suggestion'
```

## Method Details

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \**args*, \*\**kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**generate\_id** (*cls*)

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs=None*, *recurse=True*, *ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder=None*, *scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True*, *scope=True*, *reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current'*, *previousdelimiter=''*, *strict=False*, *correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.



- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

See also:

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext ()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_ ()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_ ()**

Returns the number of child elements under the current element.

**\_\_str\_\_ ()**

Alias for `text()`

## 4.4.5 Alignments

Alignments are used to make reference to external documents. It concerns references as annotation rather than references which are explicitly part of the text, such as hyperlinks and [Reference](#).

The following elements are relevant for alignments:

<code>Alignment</code>	The Alignment element is a form of higher-order annotation taht is used to point to an external resource.
<code>AlignReference</code>	The AlignReference element is used to point to specific elements inside the aligned source.

### `pynlpl.formats.folia.Alignment`

**class** `pynlpl.formats.folia.Alignment` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`

The Alignment element is a form of higher-order annotation taht is used to point to an external resource.

It concerns references as annotation rather than references which are explicitly part of the text, such as hyperlinks and [Reference](#).

Inside the Alignment element, the [AlignReference](#) element may be used to point to specific elements

(multiple denotes a span).

## Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.

Continued on next page

Table 102 – continued from previous page

<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattrs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve([documents])</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attrs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.

Continued on next page

Table 102 – continued from previous page

<code>__str__()</code>	Alias for <code>text()</code>
------------------------	-------------------------------

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.AlignReference'>, <class 'pynlpl.formats
ANNOTATIONTYPE = 26
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Alignment'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = True
XMLTAG = 'alignment'

```

### Method Details

```

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)

add(child, *args, **kwargs)

classmethod addable(parent, set=None, raiseexceptions=True)
    Tests whether a new element of this class can be added to the parent.

    This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden
    by subclasses for more customised behaviour.

```

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by *copy()*

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation()**

Perform deep validation of this element.

**Raises** `DeepValidationError`

**description()**

Obtain the description associated with the element.

**Raises** `NoSuchAnnotation` if there is no associated description.

**feat(subset)**

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables(parent, set=None, \*\*kwargs)**

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex(child, recursive=True, ignore=True)**

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata(key=None)**

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter(retaintokenisation=False)**

Return the text delimiter for this class.

Uses the `TEXTDELIMITER` attribute but may return a customised one instead.

**hasphon(cls='current', strict=True, correctionhandling=1)**

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**hastext** (*cls*='current', *strict*=True, *correctionhandling*=1)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`', may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**originaltext** (*cls*='original')

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`



**classmethod** `parsexml (node, doc, **kwargs)`

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*='current', *previousdelimiter*="", *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New element`), and it returns the `PhonContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*PhonContent*)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off '*AbstractElement*'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattribs=None, extraelements=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

**Keyword Arguments**

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolve** (*documents=None*)

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class*, *set=None*, *recursive=True*, *ignore=True*, *node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative] ):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** str or None if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** str or None if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the

corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (*TextContent*)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to None then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

**pynlpl.formats.folia.AlignReference**

**class** pynlpl.formats.folia.**AlignReference** (*doc*, \*args, \*\*kwargs)

Bases: *pynlpl.formats.folia.AbstractElement*

The AlignReference element is used to point to specific elements inside the aligned source.

It is used with *Alignment* which is responsible for pointing to the external resource.

**Method Summary**

<i>__init__</i> ( <i>doc</i> , *args, **kwargs)	Initialize self.
<i>accepts</i> (Class[, raiseexceptions, parentinstance])	
<i>add</i> (child, *args, **kwargs)	
<i>addable</i> (parent[, set, raiseexceptions])	Tests whether a new element of this class can be added to the parent.
<i>addidsuffix</i> (idsuffix[, recursive])	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<i>addtoindex</i> ([norecurse])	Makes sure this element (and all subelements), are properly added to the index.
<i>ancestor</i> (*Classes)	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<i>ancestors</i> ([Class])	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<i>append</i> (child, *args, **kwargs)	
<i>context</i> (size[, placeholder, scope])	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<i>copy</i> ([newdoc, idsuffix])	Make a deep copy of this element and all its children.
<i>copychildren</i> ([newdoc, idsuffix])	Generator creating a deep copy of the children of this element.
<i>count</i> (Class[, set, recursive, ignore, node])	Like <i>AbstractElement.select()</i> , but instead of returning the elements, it merely counts them.
<i>deepvalidation</i> ()	Perform deep validation of this element.
<i>description</i> ()	Obtain the description associated with the element.
<i>feat</i> (subset)	Obtain the feature class value of the specific subset.
<i>findcorrectionhandling</i> (cls)	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<i>findreplaceables</i> (parent[, set])	Internal method to find replaceable elements.
<i>getindex</i> (child[, recursive, ignore])	Get the index at which an element occurs, recursive by default!
<i>getmetadata</i> ([key])	Get the metadata that applies to this element, automatically inherited from parent elements
<i>gettextdelimiter</i> ([retaintokenisation])	Return the text delimiter for this class.
<i>hasphon</i> ([cls, strict, correctionhandling])	Does this element have phonetic content (of the specified class)
<i>hastext</i> ([cls, strict, correctionhandling])	Does this element have text (of the specified class)
<i>incorrection</i> ()	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

Continued on next page

Table 103 – continued from previous page

<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolve([alignmentcontext, documents])</code>	
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.

Continued on next page

Table 103 – continued from previous page

<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```
ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = None
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'aref'
```

### Method Details

```
__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__init__(doc, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
add(child, *args, **kwargs)
```



**classmethod addable** (*parent, set=None, raiseexceptions=True*)

Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

#### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix, recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (*\*Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** **\*Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

#### Parameters

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes `copy()` on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** DeepValidationError

**description** ()

Obtain the description associated with the element.

**Raises** NoSuchAnnotation if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike `phon()`, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike `text()`, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `True`.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** `bool`

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to `True`), otherwise it returns `None`

**insert** (*index, child, \*args, \*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to `AbstractElement`)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** `dict`

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*, may also be a tuple of multiple classes. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to `None` to not constrain at all.

**originaltext** (*cls*=*'original'*)

Alias for retrieving the original uncorrect text.

A call to *text()* with *correctionhandling*=*CorrectionHandling.ORIGINAL*

**classmethod parsexml** (*node*, *doc*, *\*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – **XML Element** (\*) –
- **doc** – **Document** (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=*'current'*, *previousdelimiter*=*"*, *strict*=*False*, *correctionhandling*=*1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to *False*.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to *phon()*. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *False*.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (*unicode* instance in Python 2, *str* in Python 3)

**Raises** *NoSuchPhon* – if no phonetic content is found at all.

See also:

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the PhonContent instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=True, *scope*=True)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off '`AbstractElement`'. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren*=True, *extraattribs*=None, *extraelements*=None)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child*, *\*args*, *\*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolve** (*alignmentcontext=None, documents={}*)

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to None (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to True.
- **ignore** – A list of Classes to ignore, if set to True instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean True as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (`Document`) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text, cls='current'*)

Set the text for this element.

#### Parameters

- **text** (*str*) – The text

- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker()**

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**speech\_src()**

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** `str` or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `TextContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:



```
for annotation in word:
    ...
```

`__len__()`

Returns the number of child elements under the current element.

`__str__()`

Alias for `text()`

#### 4.4.6 Descriptions, Metrics

FoLiA allows arbitrary descriptions to be assigned with any element. It also allows assigning metrics to any annotation, which consist of a key/value pair that often express a quantitative or qualitative measure. This is accomplished, respectively, with the following element classes:

<i>Description</i>	Description is an element that can be used to associate a description with almost any other FoLiA element
<i>Metric</i>	Metric elements provide a key/value pair to allow the annotation of any kind of metric with any kind of annotation element.

#### `pynlpl.formats.folia.Description`

**class** `pynlpl.formats.folia.Description`(*doc*, \**args*, \*\**kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`

Description is an element that can be used to associate a description with almost any other FoLiA element

#### Method Summary

<code>__init__(doc, *args, **kwargs)</code>	Required keyword arguments: * <i>value</i> =: The text content for the description ( <code>str</code> or <code>unicode</code> )
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, { <i>size</i> } words to the left, the current word, and { <i>size</i> } words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.

Continued on next page

Table 105 – continued from previous page

<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element
<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	

Continued on next page

Table 105 – continued from previous page

<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Description'>, <class 'pynlpl.formats.folia.Annotation'>)
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Description'
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 3, 5, 4, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None

```

```
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'desc'
```

## Method Details

`__init__(doc, *args, **kwargs)`  
Required keyword arguments: \* `value=`: The text content for the description (str or unicode)

`__init__(doc, *args, **kwargs)`  
Required keyword arguments: \* `value=`: The text content for the description (str or unicode)

**classmethod** `accepts` (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \**args*, \*\**kwargs*)

**classmethod** `addable` (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the `OCCURRENCES` property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str or None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** `ValueError`

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to *True*, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list

**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent*, *set=None*, *\*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child*, *recursive=True*, *ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current'*, *strict=True*, *correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to *True*), otherwise it returns *None*

**insert** (*index*, *child*, *\*args*, *\*\*kwargs*)

**items** (*founditems=[]*)

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs=None, recurse=True, ignorelist=False*)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size, placeholder=None, scope=None*)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class=True, scope=True, reverse=False*)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

**Parameters**

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls='original'*)

Alias for retrieving the original uncorrected text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node, doc, \*\*kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

**Parameters**

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls='current', previousdelimiter=",", strict=False, correctionhandling=1*)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the Correction/New element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend** ()

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class*=`True`, *scope*=`True`)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns `None` if no next element is found. Non-authoritative elements are never returned.

**Parameters**



- **Class** (\*) – The class to select; any python class subclassed off `AbstractElement`. Set to `True` to constrain to the same class as that of the current instance, set to `None` to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to `True` to constrain to a default list of structure elements (`Sentence`, `Paragraph`, `Division`, `Event`, `ListItem`, `Caption`), set to `None` to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, originalclass=None*)

Returns a RelaxNG definition for this element (as an XML element (`lxml.etree`) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like `append()`, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as `append()`

#### Keyword Arguments

- **alternative** (*bool*) – If set to `True`, the *replaced* element will be made into an alternative. Simply use `AbstractElement.append()` if you want the added element
- **be an alternative.** (*to*) –

See `AbstractElement.append()` for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting `scope`

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on `set`.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to `None` (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to `True`.
- **ignore** – A list of Classes to ignore, if set to `True` instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: `Alternative`, `AlternativeLayer`, `Suggestion`, and `folia.Original`. These elements and those contained within are never *authorative*. You may also include the boolean `True` as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from `AbstractElement`)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by `copy()`

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by `copy()`

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to `current` (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the `src` attribute directly.

**Returns** *str* or `None` if not found

**stricttext** (*cls*='current')

Alias for `text()` with `strict=True`

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*="", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.

- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls*='current', *correctionhandling*=1)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly*=None)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls*='current')

Alias for `text()` with `retain_tokenisation=True`

**updatetext** ()

Recompute textual value based on the text content of the children. Only supported on elements that are a `TEXTCONTAINER`

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_** ()

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_** ()

Returns the number of child elements under the current element.

**\_\_str\_\_** ()

Alias for `text()`

## **pynlpl.formats.folia.Metric**

**class** `pynlpl.formats.folia.Metric` (*doc, \*args, \*\*kwargs*)

Bases: `pynlpl.formats.folia.AbstractElement`

Metric elements provide a key/value pair to allow the annotation of any kind of metric with any kind of annotation element.

It is used for example for statistical measures to be added to elements as annotation.

### **Method Summary**

<code>__init__(doc, *args, **kwargs)</code>	Initialize self.
<code>accepts(Class[, raiseexceptions, parentinstance])</code>	
<code>add(child, *args, **kwargs)</code>	
<code>addable(parent[, set, raiseexceptions])</code>	Tests whether a new element of this class can be added to the parent.
<code>addidsuffix(idsuffix[, recursive])</code>	Appends a suffix to this element's ID, and optionally to all child IDs as well.
<code>addtoindex([norecurse])</code>	Makes sure this element (and all subelements), are properly added to the index.
<code>ancestor(*Classes)</code>	Find the most immediate ancestor of the specified type, multiple classes may be specified.
<code>ancestors([Class])</code>	Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

Continued on next page

Table 106 – continued from previous page

<code>append(child, *args, **kwargs)</code>	
<code>context(size[, placeholder, scope])</code>	Returns this word in context, {size} words to the left, the current word, and {size} words to the right
<code>copy([newdoc, idsuffix])</code>	Make a deep copy of this element and all its children.
<code>copychildren([newdoc, idsuffix])</code>	Generator creating a deep copy of the children of this element.
<code>count(Class[, set, recursive, ignore, node])</code>	Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.
<code>deepvalidation()</code>	Perform deep validation of this element.
<code>description()</code>	Obtain the description associated with the element.
<code>feat(subset)</code>	Obtain the feature class value of the specific subset.
<code>findcorrectionhandling(cls)</code>	Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused
<code>findreplaceables(parent[, set])</code>	Internal method to find replaceable elements.
<code>getindex(child[, recursive, ignore])</code>	Get the index at which an element occurs, recursive by default!
<code>getmetadata([key])</code>	Get the metadata that applies to this element, automatically inherited from parent elements
<code>gettextdelimiter([retaintokenisation])</code>	Return the text delimiter for this class.
<code>hasphon([cls, strict, correctionhandling])</code>	Does this element have phonetic content (of the specified class)
<code>hastext([cls, strict, correctionhandling])</code>	Does this element have text (of the specified class)
<code>incorrection()</code>	Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None
<code>insert(index, child, *args, **kwargs)</code>	
<code>items([founditems])</code>	Returns a depth-first flat list of <i>all</i> items below this element (not limited to AbstractElement)
<code>json([attribs, recurse, ignorelist])</code>	Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
<code>leftcontext(size[, placeholder, scope])</code>	Returns the left context for an element, as a list.
<code>next([Class, scope, reverse])</code>	Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>originaltext([cls])</code>	Alias for retrieving the original uncorrect text.
<code>parsexml(node, doc, **kwargs)</code>	Internal class method used for turning an XML element into an instance of the Class.
<code>phon([cls, previousdelimiter, strict, ...])</code>	Get the phonetic representation associated with this element (of the specified class)
<code>phoncontent([cls, correctionhandling])</code>	Get the phonetic content explicitly associated with this element (of the specified class).
<code>postappend()</code>	This method will be called after an element is added to another and does some checks.
<code>previous([Class, scope])</code>	Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.
<code>relaxng([includechildren, extraattribs, ...])</code>	Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)
<code>remove(child)</code>	Removes the child element

Continued on next page

Table 106 – continued from previous page

<code>replace(child, *args, **kwargs)</code>	Appends a child element like <code>append()</code> , but replaces any existing child element of the same type and set.
<code>resolveword(id)</code>	
<code>rightcontext(size[, placeholder, scope])</code>	Returns the right context for an element, as a list.
<code>select(Class[, set, recursive, ignore, node])</code>	Select child elements of the specified class.
<code>setdoc(newdoc)</code>	Set a different document.
<code>setdocument(doc)</code>	Associate a document with this element.
<code>setparents()</code>	Correct all parent relations for elements within the scop.
<code>settext(text[, cls])</code>	Set the text for this element.
<code>speech_speaker()</code>	Retrieves the speaker of the audio or video file associated with the element.
<code>speech_src()</code>	Retrieves the URL/filename of the audio or video file associated with the element.
<code>stricttext([cls])</code>	Alias for <code>text()</code> with <code>strict=True</code>
<code>text([cls, retaintokenisation, ...])</code>	Get the text associated with this element (of the specified class)
<code>textcontent([cls, correctionhandling])</code>	Get the text content explicitly associated with this element (of the specified class).
<code>textvalidation([warnonly])</code>	Run text validation on this element.
<code>toktext([cls])</code>	Alias for <code>text()</code> with <code>retaintokenisation=True</code>
<code>updatetext()</code>	Recompute textual value based on the text content of the children.
<code>xml([attribs, elements, skipchildren])</code>	Serialises the FoLiA element and all its contents to XML.
<code>xmlstring([pretty_print])</code>	Serialises this FoLiA element and all its contents to XML.
<code>__iter__()</code>	Iterate over all children of this element.
<code>__len__()</code>	Returns the number of child elements under the current element.
<code>__str__()</code>	Alias for <code>text()</code>

### Class Attributes

```

ACCEPTED_DATA = (<class 'pynlpl.formats.folia.Comment'>, <class 'pynlpl.formats.folia.L
ANNOTATIONTYPE = 30
AUTH = True
AUTO_GENERATE_ID = False
LABEL = 'Metric'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = False

```

```

REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
XLINK = False
XMLTAG = 'metric'

```

## Method Details

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**\_\_init\_\_** (*doc*, \*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**classmethod accepts** (*Class*, *raiseexceptions=True*, *parentinstance=None*)

**add** (*child*, \*args, \*\*kwargs)

**classmethod addable** (*parent*, *set=None*, *raiseexceptions=True*)  
Tests whether a new element of this class can be added to the parent.

This method is mostly for internal use. This will use the OCCURRENCES property, but may be overridden by subclasses for more customised behaviour.

### Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str* or *None*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can't be added?

**Returns** bool

**Raises** ValueError

**addidsuffix** (*idsuffix*, *recursive=True*)

Appends a suffix to this element's ID, and optionally to all child IDs as well. There is usually no need to call this directly, invoked implicitly by `copy()`

**addtoindex** (*norecurse=[]*)

Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

**ancestor** (\**Classes*)

Find the most immediate ancestor of the specified type, multiple classes may be specified.

**Parameters** \***Classes** – The possible classes (*AbstractElement* or subclasses) to select from. Not instances!

Example:

```
paragraph = word.ancestor(folia.Paragraph)
```

**ancestors** (*Class=None*)

Generator yielding all ancestors of this element, effectively back-tracing its path to the root element. A tuple of multiple classes may be specified.

**Parameters** **\*Class** – The class or classes (*AbstractElement* or subclasses). Not instances!

**Yields** elements (instances derived from *AbstractElement*)

**append** (*child, \*args, \*\*kwargs*)

**context** (*size, placeholder=None, scope=None*)

Returns this word in context, {size} words to the left, the current word, and {size} words to the right

**copy** (*newdoc=None, idsuffix=""*)

Make a deep copy of this element and all its children.

**Parameters**

- **newdoc** (*Document*) – The document the copy should be associated with.
- **idsuffix** (*str or bool*) – If set to a string, the ID of the copy will be append with this (prevents duplicate IDs when making copies for the same document). If set to `True`, a random suffix will be generated.

**Returns** a copy of the element

**copychildren** (*newdoc=None, idsuffix=""*)

Generator creating a deep copy of the children of this element.

Invokes *copy()* on all children, parameters are the same.

**count** (*Class, set=None, recursive=True, ignore=True, node=None*)

Like *AbstractElement.select()*, but instead of returning the elements, it merely counts them.

**Returns** int

**deepvalidation** ()

Perform deep validation of this element.

**Raises** *DeepValidationError*

**description** ()

Obtain the description associated with the element.

**Raises** *NoSuchAnnotation* if there is no associated description.

**feat** (*subset*)

Obtain the feature class value of the specific subset.

If a feature occurs multiple times, the values will be returned in a list.

Example:

```
sense = word.annotation(folia.Sense)
synset = sense.feat('synset')
```

**Returns** str or list



**findcorrectionhandling** (*cls*)

Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables** (*parent, set=None, \*\*kwargs*)

Internal method to find replaceable elements. Auxiliary function used by *AbstractElement.replace()*. Can be overridden for more fine-grained control.

**getindex** (*child, recursive=True, ignore=True*)

Get the index at which an element occurs, recursive by default!

**Returns** int

**getmetadata** (*key=None*)

Get the metadata that applies to this element, automatically inherited from parent elements

**gettextdelimiter** (*retaintokenisation=False*)

Return the text delimiter for this class.

Uses the TEXTDELIMITER attribute but may return a customised one instead.

**hasphon** (*cls='current', strict=True, correctionhandling=1*)

Does this element have phonetic content (of the specified class)

By default, and unlike *phon()*, this checks strictly, i.e. the element itself must have the phonetic content and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what phonetic content to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current phonetic content. You can set this to *CorrectionHandling.ORIGINAL* if you want the phonetic content prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**hastext** (*cls='current', strict=True, correctionhandling=1*)

Does this element have text (of the specified class)

By default, and unlike *text()*, this checks strictly, i.e. the element itself must have the text and it is not inherited from its children.

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to *True*.
- **correctionhandling** – Specifies what text to check for when corrections are encountered. The default is *CorrectionHandling.CURRENT*, which will retrieve the corrected/current text. You can set this to *CorrectionHandling.ORIGINAL* if you want the text prior to correction, and *CorrectionHandling.EITHER* if you don't care.

**Returns** bool

**incorrection** ()

Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None

**insert** (*index*, *child*, \**args*, \*\**kwargs*)

**items** (*founditems*=[])

Returns a depth-first flat list of *all* items below this element (not limited to AbstractElement)

**json** (*attrs*=None, *recurse*=True, *ignorelist*=False)

Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.

Example:

```
import json
json.dumps(word.json())
```

**Returns** dict

**leftcontext** (*size*, *placeholder*=None, *scope*=None)

Returns the left context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting scope

**next** (*Class*=True, *scope*=True, *reverse*=False)

Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to True to constrain to the same class as that of the current instance, set to None to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to True to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to None to not constrain at all.

**originaltext** (*cls*=‘original’)

Alias for retrieving the original uncorrect text.

A call to `text()` with `correctionhandling=CorrectionHandling.ORIGINAL`

**classmethod parsexml** (*node*, *doc*, \*\**kwargs*)

Internal class method used for turning an XML element into an instance of the Class.

#### Parameters

- **node** – XML Element (\*) –
- **doc** – Document (\*) –

**Returns** An instance of the current Class.

**phon** (*cls*=‘current’, *previousdelimiter*=‘’, *strict*=False, *correctionhandling*=1)

Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements wherever possible, as they are more specific. If no phonetic content can be obtained from the children and the element has itself phonetic content associated with it, then that will be used.

#### Parameters

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and phonetic content will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `phon()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what phonetic content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current phonetic content. You can set this to `CorrectionHandling.ORIGINAL` if you want the phonetic content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

Example:

```
word.phon()
```

**Returns** The phonetic content of the element (`unicode` instance in Python 2, `str` in Python 3)

**Raises** `NoSuchPhon` – if no phonetic content is found at all.

**See also:**

`phoncontent()`: Retrieves the phonetic content as an element rather than a string `text()`  
`textcontent()`

**phoncontent** (*cls*='current', *correctionhandling*=1)

Get the phonetic content explicitly associated with this element (of the specified class).

Unlike `phon()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `PhonContent` instance rather than the actual text!

**Parameters**

- **cls** (*str*) – The class of the phonetic content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`PhonContent`)

**Raises** `NoSuchPhon` if there is no phonetic content for the element

**See also:**

`phon()` `textcontent()` `text()`

**postappend()**

This method will be called after an element is added to another and does some checks.

It can do extra checks and if necessary raise exceptions to prevent addition. By default makes sure the right document is associated.

This method is mostly for internal use.

**previous** (*Class=True, scope=True*)

Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope. Returns None if no next element is found. Non-authoritative elements are never returned.

#### Parameters

- **Class** (\*) – The class to select; any python class subclassed off *AbstractElement*. Set to *True* to constrain to the same class as that of the current instance, set to *None* to not constrain at all
- **scope** (\*) – A list of classes which are never crossed looking for a next element. Set to *True* to constrain to a default list of structure elements (*Sentence*, *Paragraph*, *Division*, *Event*, *ListItem*, *Caption*), set to *None* to not constrain at all.

**classmethod relaxng** (*includechildren=True, extraattrs=None, extraelements=None, orig-class=None*)

Returns a RelaxNG definition for this element (as an XML element (*lxml.etree*) rather than a string)

**remove** (*child*)

Removes the child element

**replace** (*child, \*args, \*\*kwargs*)

Appends a child element like *append()*, but replaces any existing child element of the same type and set. If no such child element exists, this will act the same as *append()*

#### Keyword Arguments

- **alternative** (*bool*) – If set to *True*, the *replaced* element will be made into an alternative. Simply use *AbstractElement.append()* if you want the added element
- **be an alternative.** (*to*) –

See *AbstractElement.append()* for more information and all parameters.

**resolveword** (*id*)

**rightcontext** (*size, placeholder=None, scope=None*)

Returns the right context for an element, as a list. This method crosses sentence/paragraph boundaries by default, which can be restricted by setting *scope*

**select** (*Class, set=None, recursive=True, ignore=True, node=None*)

Select child elements of the specified class.

A further restriction can be made based on set.

#### Parameters

- **Class** (*class*) – The class to select; any python class (not instance) subclassed off *AbstractElement*
- **Set** (*str*) – The set to match against, only elements pertaining to this set will be returned. If set to *None* (default), all elements regardless of set will be returned.
- **recursive** (*bool*) – Select recursively? Descending into child elements? Defaults to *True*.
- **ignore** – A list of Classes to ignore, if set to *True* instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: *Alternative*, *AlternativeLayer*, *Suggestion*, and *folia.Original*. These elements and those contained within are never *authorative*. You may also include the boolean *True* as a member of a list, if you want to skip additional tags along the predefined non-authoritative ones.
- **node** (\*) – Reserved for internal usage, used in recursion.

**Yields** Elements (instances derived from *AbstractElement*)

Example:

```
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original,
↳ folia.Suggestion, folia.Alternative]):
    ..
```

**setdoc** (*newdoc*)

Set a different document. Usually no need to call this directly, invoked implicitly by *copy()*

**setdocument** (*doc*)

Associate a document with this element.

**Parameters** *doc* (*Document*) – A document

Each element must be associated with a FoLiA document.

**setparents** ()

Correct all parent relations for elements within the scop. There is usually no need to call this directly, invoked implicitly by *copy()*

**settext** (*text*, *cls*='current')

Set the text for this element.

**Parameters**

- **text** (*str*) – The text
- **cls** (*str*) – The class of the text, defaults to *current* (leave this unless you know what you are doing). There may be only one text content element of each class associated with the element.

**speech\_speaker** ()

Retrieves the speaker of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**speech\_src** ()

Retrieves the URL/filename of the audio or video file associated with the element.

The source is inherited from ancestor elements if none is specified. For this reason, always use this method rather than access the *src* attribute directly.

**Returns** *str* or *None* if not found

**stricttext** (*cls*='current')

Alias for *text()* with *strict*=True

**text** (*cls*='current', *retaintokenisation*=False, *previousdelimiter*=",", *strict*=False, *correctionhandling*=1, *normalize\_spaces*=False)

Get the text associated with this element (of the specified class)

The text will be constructed from child-elements wherever possible, as they are more specific. If no text can be obtained from the children and the element has itself text associated with it, then that will be used.

**Parameters**

- **cls** (*str*) – The class of the text content to obtain, defaults to *current*.

- **retaintokenisation** (*bool*) – If set, the space attribute on words will be ignored, otherwise it will be adhered to and text will be detokenised as much as possible. Defaults to `False`.
- **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputted, useful when chaining calls to `text()`. Defaults to an empty string.
- **strict** (*bool*) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to `False`.
- **correctionhandling** – Specifies what text to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current text. You can set this to `CorrectionHandling.ORIGINAL` if you want the text prior to correction, and `CorrectionHandling.EITHER` if you don't care.
- **normalize\_spaces** (*bool*) – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

Example:

```
word.text()
```

**Returns** The text of the element (unicode instance in Python 2, `str` in Python 3)

**Raises** `NoSuchText` – if no text is found at all.

**textcontent** (*cls='current', correctionhandling=1*)

Get the text content explicitly associated with this element (of the specified class).

Unlike `text()`, this method does not recurse into child elements (with the sole exception of the `Correction/New` element), and it returns the `TextContent` instance rather than the actual text!

#### Parameters

- **cls** (*str*) – The class of the text content to obtain, defaults to `current`.
- **correctionhandling** – Specifies what content to retrieve when corrections are encountered. The default is `CorrectionHandling.CURRENT`, which will retrieve the corrected/current content. You can set this to `CorrectionHandling.ORIGINAL` if you want the content prior to correction, and `CorrectionHandling.EITHER` if you don't care.

**Returns** The phonetic content (`TextContent`)

**Raises** `NoSuchText` if there is no text content for the element

**See also:**

`text()` `phoncontent()` `phon()`

**textvalidation** (*warnonly=None*)

Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

**Parameters** **warnonly** (*bool*) – Warn only (True) or raise exceptions (False). If set to `None` then this value will be determined based on the document's FoLiA version (Warn only before FoLiA v1.5)

**Returns** `bool`

**toktext** (*cls='current'*)

Alias for `text()` with `retaintokenisation=True`

**updatetext()**

Recompute textual value based on the text content of the children. Only supported on elements that are a TEXTCONTAINER

**xml** (*attrs=None, elements=None, skipchildren=False*)

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

**Returns** an `lxml.etree.Element`

**See also:**

`AbstractElement.xmlstring()` - for direct string output

**xmlstring** (*pretty\_print=False*)

Serialises this FoLiA element and all its contents to XML.

**Returns** a string with XML representation for this element and all its children

**Return type** `str`

**\_\_iter\_\_()**

Iterate over all children of this element.

Example:

```
for annotation in word:
    ...
```

**\_\_len\_\_()**

Returns the number of child elements under the current element.

**\_\_str\_\_()**

Alias for `text()`

## 4.5 Metadata

FoLiA can be used with a variety of more advanced metadata schemes (e.g. Dublin Core, CMDI). If this is too much, you can use its own simple *native* metadata facility, a simple key value store. After instantiation of a `Document`, the metadata can be accessed through the `metadata` attribute, which behaves like a Python dictionary:

```
doc = folia.Document(file="/path/to/document.xml")
doc.metadata['language'] = "en"
```





## 5.1 Corpus Gesproken Nederlands

```
exception pynlpl.formats.cgn.InvalidFeatureException
exception pynlpl.formats.cgn.InvalidTagException
pynlpl.formats.cgn.parse_cgn_postag (rawtag, raisefeatureexceptions=False)
```

## 5.2 FoLiA

See *folia* : [folia.html](#)

## 5.3 GIZA++

```
class pynlpl.formats.giza.GizaModel (filename, encoding='utf-8')
class pynlpl.formats.giza.GizaSentenceAlignment (sourceline, targetline, index)

    getalignedtarget (index)
        Returns target range only if source index aligns to a single consecutive range of target tokens.

    intersect (other)

class pynlpl.formats.giza.IntersectionAlignment (source2target, target2source, encoding=False)

    reset ()

class pynlpl.formats.giza.MultiWordAlignment (filename, encoding=False)
    Source to Target alignment: reads source-target.A3.final files, in which each source word may be aligned to
    multiple target words (adapted from code by Sander Canisius)
```

```
reset ()

targetword (index, targetwords, alignment)
    Return the aligned targeword for a specified index in the source words. Multiple words are concatenated
    together with a space in between

targetwords (index, targetwords, alignment)
    Return the aligned targetwords for a specified index in the source words

class pynlpl.formats.giza.WordAlignment (filename, encoding=False)
    Target to Source alignment: reads target-source.A3.final files, in which each source word is aligned to one target
    word

reset ()

targetword (index, targetwords, alignment)
    Return the aligned targetword for a specified index in the source words

pynlpl.formats.giza.parseAlignment (tokens)
```

## 5.4 Moses

```
class pynlpl.formats.moses.PhraseTable (filename, quiet=False, reverse=False, delim-  
                                         iter='|||', score_column=3, max_sourcen=0,  
                                         sourceencoder=None, targetencoder=None, score-  
                                         filter=None)

class pynlpl.formats.moses.PhraseTableClient (host='localhost', port=65432)
```

## 5.5 SoNaR

```
class pynlpl.formats.sonar.Corpus (corpusdir, extension='pos', restrict_to_collection="",  
                                   conditionf=<function Corpus.<lambda>>, ignoreer-  
                                   rors=False)

class pynlpl.formats.sonar.CorpusDocument (filename, encoding='iso-8859-15')
    This class represent one document/text of the Corpus (read-only)

paragraphs (with_id=False)
    Extracts paragraphs, returns list of plain-text(!) paragraphs

sentences ()
    Iterate over all sentences (sentence_id, sentence) in the document, sentence is a list of 4-tuples
    (word,id,pos,lemma)

words ()

class pynlpl.formats.sonar.CorpusDocumentX (filename, tree=None, index=True)
    This class represent one document/text of the Corpus, loaded into memory at once and retaining the full structure

paragraphs (node=None)
    iterate over paragraphs

save (filename=None, encoding='iso-8859-15')

sentences (node=None)
    iterate over sentences
```

```

validate (formats_dir='../formats/')
    checks if the document is valid

words (node=None)
    iterate over words

xpath (expression)
    Executes an xpath expression using the correct namespaces

class pynlpl.formats.sonar.CorpusFiles (corpusdir,           extension='pos',           re-
                                         strict_to_collection=",           conditionf=<function
                                         Corpus.<lambda>>, ignoreerrors=False)

class pynlpl.formats.sonar.CorpusX (corpusdir, extension='pos', restrict_to_collection=",
                                         conditionf=<function Corpus.<lambda>>, ignoreer-
                                         rors=False)

pynlpl.formats.sonar.ns (namespace)
    Resolves the namespace identifier to a full URL

```

## 5.6 Taggerdata

```

class pynlpl.formats.taggerdata.Taggerdata (filename, encoding='utf-8', mode='r')

    align (referencewords, datatuple)
        align the reference sentence with the tagged data

    close ()

    next ()

    reset ()

    write (sentence)

```

## 5.7 TiMBL

```

class pynlpl.formats.timbl.TimblOutput (stream, delimiter=' ', ignorecolumns=[], ignoreval-
                                         ues=[])
    A class for reading Timbl classifier output, supports the +v+db option and ignores comments starting with #

    parseDistribution (instance, start, end=None)

```



---

Language Models

---

```
class pynlpl.lm.lm.ARPALanguageModel (filename,      encoding='utf-8',      encoder=None,
                                     base_e=True,    dounknown=True,    debug=False,
                                     mode='simple')
```

Full back-off language model, loaded from file in ARPA format.

This class does not build the model but allows you to use a pre-computed one. You can use the tool ngram-count from for instance SRILM to actually build the model.

```
class NgramsProbs (data, mode='simple', delim=' ')
```

Store Ngrams with their probabilities and backoffs.

This class is used in order to abstract the physical storage layout, and enable memory/speed tradeoffs.

```
backoff (ngram)
```

Return backoff value of a given ngram tuple

```
prob (ngram)
```

Return probability of given ngram tuple

```
score (data, history=None)
```

```
scoreword (word, history=None)
```

```
class pynlpl.lm.lm.SimpleLanguageModel (n=2, casesensitive=True, beginmarker='<begin>',
                                     endmarker='<end>')
```

This is a simple unsmoothed language model. This class can both hold and compute the model.

```
append (sentence)
```

```
load (filename)
```

```
save (filename)
```

```
scoresentence (sentence)
```

```
class pynlpl.lm.srilm.SRILM (filename, n)
```

```
logscore (ngram)
```

**scoresentence** (*sentence*, *unknownwordprob=-12*)

**exception** `pynlpl.lm.srilm.SRILMException`  
Base Exception for SRILM.

**class** `pynlpl.lm.client.LMClient` (*host='localhost', port=12346, n=0*)

**scoresentence** (*sentence*)

---

## Search Algorithms

---

This module contains various search algorithms.

```
class pynlpl.search.AbstractSearch (**kwargs)

    prune (state)
        Pruning method is called AFTER expansion of each node

    reset ()

    searchall ()
        Returns a list of all solutions

    searchbest ()
        Returns the single best result (if multiple have the same score, the first match is returned)

    searchfirst ()
        Returns the very first result (regardless of it being the best or not!)

    searchlast (n=10)
        Return the last n results (or possibly less if not found). Note that the last results are not necessarily the best
        ones! Depending on the search type.

    searchtop (n=10)
        Return the top n best results (or possibly less if not enough is found)

    traversal ()
        Returns all visited states (only when keeptraversal=True), note that this is not equal to the path, but contains
        all states that were checked!

    traversalsize ()
        Returns the number of nodes visited (also when keeptravel=False). Note that this is not equal to the path,
        but contains all states that were checked!

    visited (state)

class pynlpl.search.AbstractSearchState (parent=None, cost=0)
```

**depth()**

**expand()**  
Generates successor states, implement your custom operators in the derived method.

**path()**

**pathcost()**

**score()**  
Should return a heuristic value. This needs to be set if you plan to used an informed search algorithm.

**test** (*goalstates=None*)  
Checks whether this state is a valid goal state, returns a boolean. If no goalstate is defined, then all states will test positively, this is what you usually want for optimisation problems.

**class** `pynlpl.search.BeamSearch` (*states, beamsize, \*\*kwargs*)  
Local beam search algorithm

**class** `pynlpl.search.BeamedBestFirstSearch` (*states, beamsize, \*\*kwargs*)  
Best first search with a beamsize (non-optimal!)

**prune** (*state*)  
Pruning method is called AFTER expansion of each node

**class** `pynlpl.search.BestFirstSearch` (*state, \*\*kwargs*)

**class** `pynlpl.search.BreadthFirstSearch` (*state, \*\*kwargs*)

**class** `pynlpl.search.DepthFirstSearch` (*state, \*\*kwargs*)

**class** `pynlpl.search.EarlyEagerBeamSearch` (*state, beamsize, \*\*kwargs*)  
A beam search that prunes early (after each state expansion) and eagerly (weeding out worse successors)

**prune** (*state*)  
Pruning method is called AFTER expansion of each node

**class** `pynlpl.search.HillClimbingSearch` (*state, \*\*kwargs*)  
(identical to beamsearch with beam 1, but implemented differently)

**class** `pynlpl.search.IterativeDeepening` (*state, \*\*kwargs*)

**traversal()**  
Returns all visited states (only when `keeptraversal=True`), note that this is not equal to the path, but contains all states that were checked!

**traversalsize()**  
Returns the number of nodes visited (also when `keeptravel=False`). Note that this is not equal to the path, but contains all states that were checked!

**class** `pynlpl.search.StochasticBeamSearch` (*states, beamsize, \*\*kwargs*)

**prune** (*state*)  
Pruning method is called AFTER expansion of each node

`pynlpl.search.binary_search` (*a, x, lo=0, hi=None*)



---

## Statistics and Information Theory

---

This module contains classes and functions for statistics and information theory. It is imported as follows:

```
import pynlpl.statistics
```

### 8.1 Generic functions

Amongst others, the following generic statistical functions are available:

```
* ``mean(list)`` - Computes the mean of a given list of numbers
```

- `median(list)` - Computes the median of a given list of numbers
- `stddev(list)` - Computes the standard deviation of a given list of numbers
- `normalize(list)` - Normalizes a list of numbers so that the sum is 1.0 .

### 8.2 Frequency Lists and Distributions

One of the most basic and widespread tasks in NLP is the creation of a frequency list. Counting is established by simply appending lists to the frequencylist:

```
freqlist = pynlpl.statistics.FrequencyList()  
freqlist.append(['to', 'be', 'or', 'not', 'to', 'be'])
```

Take care not to append lists rather than strings unless you mean to create a frequency list over its characters rather than words. You may want to use the `pynlpl.textprocessors.crudetokeniser` first:

```
freqlist.append(pynlpl.textprocessors.crude_tokeniser("to be or not to be"))
```

The count can also be incremented explicitly explicitly for a single item:

```
freqlist.count('shakespeare')
```

The FrequencyList offers dictionary-like access. For example, the following statement will be true for the frequency list just created:

```
freqlist['be'] == 2
```

Normalised counts (pseudo-probabilities) can be obtained using the `p()` method:

```
freqlist.p('be')
```

Normalised counts can also be obtained by instantiation a Distribution instance using the frequency list:

```
dist = pynlpl.statistics.Distribution(freqlist)
```

This too offers a dictionary-like interface, where values are by definition normalised. The advantage of a Distribution class is that it offers information-theoretic methods such as `entropy()`, `maxentropy()`, `perplexity()` and `poslog()`.

A frequency list can be saved to file using the `save(filename)` method, and loaded back from file using the `load(filename)` method. The `output()` method is a generator yielding strings for each line of output, in ranked order.

## 8.3 API Reference

This is a Python library containing classes for Statistic and Information Theoretical computations. It also contains some code from Peter Norvig, AI: A Modern Approach : <http://aima.cs.berkeley.edu/python/utils.html>

**class** `pynlpl.statistics.Distribution(data, base=2)`

A distribution can be created over a FrequencyList or a plain dictionary with numeric values. It will be normalized automatically. This implementation uses dictionaries/hashing

**entropy** (`base=2`)

Compute the entropy of the distribution

**information** (`type`)

Computes the information content of the specified type:  $-\log_e(p(X))$

**items** ()

Returns an *unranked* list of (type, prob) pairs. Use this only if you are not interested in the order.

**keys** ()

**maxentropy** (`base=2`)

Compute the maximum entropy of the distribution:  $\log_e(N)$

**mode** ()

Returns the type that occurs the most frequently in the probability distribution

**output** (`delimiter='\t', freqlist=None`)

Generator yielding formatted strings expressing the time and probability for each item in the distribution

**perplexity** (`base=2`)

**poslog** (`type`)

alias for information content

**values** ()

---

```

class pynlpl.statistics.FrequencyList (tokens=None, casesensitive=True, dovalidation=True)
    A frequency list (implemented using dictionaries)

    append (tokens)
        Add a list of tokens to the frequencylist. This method will count them for you.

    count (type, amount=1)
        Count a certain type. The counter will increase by the amount specified (defaults to one)

    dict ()

    items ()
        Returns an unranked list of (type, count) pairs. Use this only if you are not interested in the order.

    load (filename)
        Load a frequency list from file (in the format produced by the save method)

    mode ()
        Returns the type that occurs the most frequently in the frequency list

    output (delimiter='\\n', addnormalised=False)
        Print a representation of the frequency list

    p (type)
        Returns the probability (relative frequency) of the token

    save (filename, addnormalised=False)
        Save a frequency list to file, can be loaded later using the load method

    sum ()
        Returns the total amount of tokens

    tokens ()
        Returns the total amount of tokens

    typetokenratio ()
        Computes the type/token ratio

    values ()

class pynlpl.statistics.HiddenMarkovModel (startstate, endstate=None)

    print_dptable (V)

    setemission (state, distribution)

    viterbi (observations, doprint=False)

class pynlpl.statistics.MarkovChain (startstate, endstate=None)

    accessible (fromstate, tostate)
        Is state tonode directly accessible (in one step) from state fromnode? (i.e. is there an edge between the nodes). If so, return the probability, else zero

    communicates (fromstate, tostate, maxlength=999999)
        See if a node communicates (directly or indirectly) with another. Returns the probability of the shortest path (probably, but not necessarily the highest probability)

    p (sequence, subsequence=True)
        Returns the probability of the given sequence or subsequence (if subsequence=True, default).

    reducible ()

```

**settransitions** (*state, distribution*)

**size** ()

`pynlpl.statistics.dotproduct (X, Y)`

Return the sum of the element-wise product of vectors x and y. >>> dotproduct([1, 2, 3], [1000, 100, 10]) 1230

`pynlpl.statistics.histogram (values, mode=0, bin_function=None)`

Return a list of (value, count) pairs, summarizing the input values. Sorted by increasing value, or if mode=1, by decreasing count. If bin\_function is given, map it over values first.

`pynlpl.statistics.levenshtein (s1, s2, maxdistance=9999)`

Computes the levenshtein distance between two strings. Adapted from: [http://en.wikibooks.org/wiki/Algorithm\\_Implementation/Strings/Levenshtein\\_distance#Python](http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#Python)

`pynlpl.statistics.log2 (x)`

Base 2 logarithm. >>> log2(1024) 10.0

`pynlpl.statistics.mean (values)`

Return the arithmetic average of the values.

`pynlpl.statistics.median (values)`

Return the middle value, when the values are sorted. If there are an odd number of elements, try to average the middle two. If they can't be averaged (e.g. they are strings), choose one at random. >>> median([10, 100, 11]) 11 >>> median([1, 2, 3, 4]) 2.5

`pynlpl.statistics.mode (values)`

Return the most common value in the list of values. >>> mode([1, 2, 3, 2]) 2

`pynlpl.statistics.normalize (numbers, total=1.0)`

Multiply each number by a constant such that the sum is 1.0 (or total). >>> normalize([1,2,1]) [0.25, 0.5, 0.25]

`pynlpl.statistics.product (seq)`

Return the product of a sequence of numerical values. >>> product([1,2,6]) 12

`pynlpl.statistics.stddev (values, meanval=None)`

The standard deviation of a set of values. Pass in the mean if you already know it.

`pynlpl.statistics.vector_add (a, b)`

Component-wise addition of two vectors. >>> vector\_add((0, 1), (8, 9)) (8, 10)

This module contains classes and functions for text processing. It is imported as follows:

```
import pynlpl.textprocessors
```

## 9.1 Tokenisation

A very crude tokeniser is available in the form of the function `pynlpl.textprocessors.crude_tokeniser(string)`. This will split punctuation characters from words and returns a list of tokens. It however has no regard for abbreviations and end-of-sentence detection, which is functionality a more sophisticated tokeniser can provide:

```
tokens = pynlpl.textprocessors.crude_tokeniser("to be, or not to be.")
```

This will result in:

```
tokens == ['to','be',',','or','not','to','be',',']
```

## 9.2 N-gram extraction

The extraction of n-grams is an elemental operation in Natural Language Processing. PyNLPI offers the `Windower` class to accomplish this task:

```
tokens = pynlpl.textprocessors.crude_tokeniser("to be or not to be")
for trigram in Windower(tokens, 3):
    print trigram
```

The input to the `Windower` should be a list of words and a value for `n`. In addition, the windower can output extra symbols at the beginning of the input sequence and at the end of it. By default, this behaviour is enabled and the input symbol is `<begin>`, whereas the output symbol is `<end>`. If this behaviour is unwanted you can suppress it by instantiating the `Windower` as follows:

```
Windower(tokens, 3, None, None)
```

The Windower is implemented as a Python generator and at each iteration yields a tuple of length n.

```
class pynlpl.textprocessors.MultiWindower(tokens, min_n=1, max_n=9, begin-  
marker=None, endmarker=None)
```

Extract n-grams of various configurations from a sequence

```
class pynlpl.textprocessors.ReflowText(stream, filternontext=True)
```

Attempts to re-flow a text that has arbitrary line endings in it. Also undoes hyphenisation

```
class pynlpl.textprocessors.Tokenizer(stream, splitsentences=True,  
onesentenceperline=False,  
regexps=(re.compile('^((?:https?):(?:/(?:\\|\\|))|www\\.|(?:[\\w\\d:#@%/$  
=\\\\&](?:#!)?)*)', re.compile('[A-Za-z0-9\\+_-  
]+@[A-Za-z0-9\\+_-]+(?:\\.[a-zA-Z]+)+')))
```

A tokenizer and sentence splitter, which acts on a file/stream-like object and when iterating over the object it yields a lists of tokens (in case the sentence splitter is active (default)), or a token (if the sentence splitter is deactivated).

```
class pynlpl.textprocessors.Windower(tokens, n=1, beginmarker='<begin>', end-  
marker='<end>')
```

Moves a sliding window over a list of tokens, upon iteration in yields all n-grams of specified size in a tuple.

Example without markers:

```
>>> for ngram in Windower("This is a test .", 3, None, None):  
...     print(" ".join(ngram))  
This is a  
is a test  
a test .
```

Example with default markers:

```
>>> for ngram in Windower("This is a test .", 3):  
...     print(" ".join(ngram))  
<begin> <begin> This  
<begin> This is  
This is a  
is a test  
a test .  
test . <end>  
. <end> <end>
```

```
pynlpl.textprocessors.calculate_overlap(haystack, needle, allowpartial=True)
```

Calculate the overlap between two sequences. Yields (overlap, placement) tuples (multiple because there may be multiple overlaps!). The former is the part of the sequence that overlaps, and the latter is -1 if the overlap is on the left side, 0 if it is a subset, 1 if it overlaps on the right side, 2 if its an identical match

```
pynlpl.textprocessors.crude_tokenizer(text)
```

Replaced by tokenize(). Alias

```
pynlpl.textprocessors.find_keyword_in_context(tokens, keyword, contextsize=1)
```

Find a keyword in a particular sequence of tokens, and return the local context. Contextsize is the number of words to the left and right. The keyword may have multiple word, in which case it should to passed as a tuple or list

```
pynlpl.textprocessors.is_end_of_sentence(tokens, i)
```

`pynlpl.textprocessors.swap(tokens, maxdist=2)`  
 Perform a swap operation on a sequence of tokens, exhaustively swapping all tokens up to the maximum specified distance. This is a subset of all permutations.

## Alias for the British

Tokenizes a string and returns a list of tokens

- **text** (*string*) – The text to tokenise
- **regexps** (*Tuple/list of regular expressions to use in tokenisation*) – Regular expressions to use as tokeniser rules in tokenisation (default=`_pynlpl.textprocessors.TOKENIZER_RULES_`)

Examples:

1067





## CHAPTER 10

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### p

- `pynlpl.common`, 3
- `pynlpl.datatypes`, 5
- `pynlpl.evaluation`, 9
- `pynlpl.formats.cgn`, 1053
- `pynlpl.formats.folia`, 11
- `pynlpl.formats.giza`, 1053
- `pynlpl.formats.moses`, 1054
- `pynlpl.formats.sonar`, 1054
- `pynlpl.formats.taggerdata`, 1055
- `pynlpl.formats.timbl`, 1055
- `pynlpl.lm.client`, 1058
- `pynlpl.lm.lm`, 1057
- `pynlpl.lm.srilm`, 1057
- `pynlpl.search`, 1059
- `pynlpl.statistics`, 1062
- `pynlpl.textprocessors`, 1066



## Symbols

- `__init__()` (pynlpl.formats.folia.AbstractAnnotationLayer method), 75
- `__init__()` (pynlpl.formats.folia.AbstractElement method), 26
- `__init__()` (pynlpl.formats.folia.AbstractSpanAnnotation method), 52
- `__init__()` (pynlpl.formats.folia.AbstractStructureElement method), 38
- `__init__()` (pynlpl.formats.folia.AbstractTextMarkup method), 87
- `__init__()` (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- `__init__()` (pynlpl.formats.folia.ActorFeature method), 885
- `__init__()` (pynlpl.formats.folia.AlignReference method), 1020
- `__init__()` (pynlpl.formats.folia.Alignment method), 1009
- `__init__()` (pynlpl.formats.folia.AllowTokenAnnotation method), 48
- `__init__()` (pynlpl.formats.folia.Alternative method), 919
- `__init__()` (pynlpl.formats.folia.AlternativeLayers method), 932
- `__init__()` (pynlpl.formats.folia.BegindatetimeFeature method), 896
- `__init__()` (pynlpl.formats.folia.Cell method), 101
- `__init__()` (pynlpl.formats.folia.Chunk method), 518
- `__init__()` (pynlpl.formats.folia.ChunkingLayer method), 647
- `__init__()` (pynlpl.formats.folia.CoreferenceChain method), 530
- `__init__()` (pynlpl.formats.folia.CoreferenceLayer method), 659
- `__init__()` (pynlpl.formats.folia.CoreferenceLink method), 765
- `__init__()` (pynlpl.formats.folia.Correction method), 944
- `__init__()` (pynlpl.formats.folia.Current method), 954
- `__init__()` (pynlpl.formats.folia.Definition method), 114
- `__init__()` (pynlpl.formats.folia.DependenciesLayer method), 670
- `__init__()` (pynlpl.formats.folia.Dependency method), 541
- `__init__()` (pynlpl.formats.folia.DependencyDependent method), 776
- `__init__()` (pynlpl.formats.folia.Description method), 1032
- `__init__()` (pynlpl.formats.folia.Division method), 127
- `__init__()` (pynlpl.formats.folia.Document method), 15, 16
- `__init__()` (pynlpl.formats.folia.DomainAnnotation method), 429
- `__init__()` (pynlpl.formats.folia.EnddatetimeFeature method), 907
- `__init__()` (pynlpl.formats.folia.EntitiesLayer method), 682
- `__init__()` (pynlpl.formats.folia.Entity method), 553
- `__init__()` (pynlpl.formats.folia.Entry method), 140
- `__init__()` (pynlpl.formats.folia.ErrorDetection method), 965
- `__init__()` (pynlpl.formats.folia.Event method), 153
- `__init__()` (pynlpl.formats.folia.Example method), 166
- `__init__()` (pynlpl.formats.folia.Feature method), 863
- `__init__()` (pynlpl.formats.folia.Figure method), 179
- `__init__()` (pynlpl.formats.folia.Gap method), 191
- `__init__()` (pynlpl.formats.folia.Head method), 203
- `__init__()` (pynlpl.formats.folia.Headspace method), 788
- `__init__()` (pynlpl.formats.folia.LangAnnotation method), 451
- `__init__()` (pynlpl.formats.folia.LemmaAnnotation method), 462
- `__init__()` (pynlpl.formats.folia.Linebreak method), 216
- `__init__()` (pynlpl.formats.folia.List method), 229
- `__init__()` (pynlpl.formats.folia.ListItem method), 242
- `__init__()` (pynlpl.formats.folia.Metric method), 1043
- `__init__()` (pynlpl.formats.folia.New method), 976
- `__init__()` (pynlpl.formats.folia.Note method), 255
- `__init__()` (pynlpl.formats.folia.Observation method), 565
- `__init__()` (pynlpl.formats.folia.ObservationLayer method), 670

method), 694  
\_\_init\_\_() (pynlpl.formats.folia.Original method), 987  
\_\_init\_\_() (pynlpl.formats.folia.Paragraph method), 268  
\_\_init\_\_() (pynlpl.formats.folia.Part method), 281  
\_\_init\_\_() (pynlpl.formats.folia.PhonContent method), 506  
\_\_init\_\_() (pynlpl.formats.folia.PosAnnotation method), 440  
\_\_init\_\_() (pynlpl.formats.folia.Predicate method), 576  
\_\_init\_\_() (pynlpl.formats.folia.Quote method), 294  
\_\_init\_\_() (pynlpl.formats.folia.Reader method), 805  
\_\_init\_\_() (pynlpl.formats.folia.Reference method), 307  
\_\_init\_\_() (pynlpl.formats.folia.Row method), 320  
\_\_init\_\_() (pynlpl.formats.folia.SemanticRole method), 623  
\_\_init\_\_() (pynlpl.formats.folia.SemanticRolesLayer method), 741  
\_\_init\_\_() (pynlpl.formats.folia.SenseAnnotation method), 473  
\_\_init\_\_() (pynlpl.formats.folia.Sentence method), 333  
\_\_init\_\_() (pynlpl.formats.folia.Sentiment method), 588  
\_\_init\_\_() (pynlpl.formats.folia.SentimentLayer method), 706  
\_\_init\_\_() (pynlpl.formats.folia.Statement method), 600  
\_\_init\_\_() (pynlpl.formats.folia.StatementLayer method), 717  
\_\_init\_\_() (pynlpl.formats.folia.SubjectivityAnnotation method), 484  
\_\_init\_\_() (pynlpl.formats.folia.Suggestion method), 998  
\_\_init\_\_() (pynlpl.formats.folia.SynsetFeature method), 874  
\_\_init\_\_() (pynlpl.formats.folia.SyntacticUnit method), 611  
\_\_init\_\_() (pynlpl.formats.folia.SyntaxLayer method), 729  
\_\_init\_\_() (pynlpl.formats.folia.Table method), 347  
\_\_init\_\_() (pynlpl.formats.folia.TableHead method), 373  
\_\_init\_\_() (pynlpl.formats.folia.Term method), 360  
\_\_init\_\_() (pynlpl.formats.folia.Text method), 386  
\_\_init\_\_() (pynlpl.formats.folia.TextContent method), 495  
\_\_init\_\_() (pynlpl.formats.folia.TextMarkupCorrection method), 841  
\_\_init\_\_() (pynlpl.formats.folia.TextMarkupError method), 851  
\_\_init\_\_() (pynlpl.formats.folia.TextMarkupGap method), 809  
\_\_init\_\_() (pynlpl.formats.folia.TextMarkupString method), 819  
\_\_init\_\_() (pynlpl.formats.folia.TextMarkupStyle method), 830  
\_\_init\_\_() (pynlpl.formats.folia.TimeSegment method), 635  
\_\_init\_\_() (pynlpl.formats.folia.TimingLayer method), 753  
\_\_init\_\_() (pynlpl.formats.folia.Whitespace method), 399, 400  
\_\_init\_\_() (pynlpl.formats.folia.Word method), 413  
\_\_init\_\_() (pynlpl.formats.fql.Query method), 803  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractAnnotationLayer method), 83  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractElement method), 34  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractSpanAnnotation method), 61  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractStructureElement method), 47  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractTextMarkup method), 94  
\_\_iter\_\_() (pynlpl.formats.folia.AbstractTokenAnnotation method), 72  
\_\_iter\_\_() (pynlpl.formats.folia.ActorFeature method), 893  
\_\_iter\_\_() (pynlpl.formats.folia.AlignReference method), 1028  
\_\_iter\_\_() (pynlpl.formats.folia.Alignment method), 1017  
\_\_iter\_\_() (pynlpl.formats.folia.Alternative method), 928  
\_\_iter\_\_() (pynlpl.formats.folia.AlternativeLayers method), 939  
\_\_iter\_\_() (pynlpl.formats.folia.BegindatetimeFeature method), 904  
\_\_iter\_\_() (pynlpl.formats.folia.Cell method), 110  
\_\_iter\_\_() (pynlpl.formats.folia.Chunk method), 527  
\_\_iter\_\_() (pynlpl.formats.folia.ChunkingLayer method), 655  
\_\_iter\_\_() (pynlpl.formats.folia.CoreferenceChain method), 538  
\_\_iter\_\_() (pynlpl.formats.folia.CoreferenceLayer method), 667  
\_\_iter\_\_() (pynlpl.formats.folia.CoreferenceLink method), 773  
\_\_iter\_\_() (pynlpl.formats.folia.Correction method), 951  
\_\_iter\_\_() (pynlpl.formats.folia.Current method), 962  
\_\_iter\_\_() (pynlpl.formats.folia.Definition method), 123  
\_\_iter\_\_() (pynlpl.formats.folia.DependenciesLayer method), 679  
\_\_iter\_\_() (pynlpl.formats.folia.Dependency method), 550  
\_\_iter\_\_() (pynlpl.formats.folia.DependencyDependent method), 785  
\_\_iter\_\_() (pynlpl.formats.folia.Description method), 1040  
\_\_iter\_\_() (pynlpl.formats.folia.Division method), 136  
\_\_iter\_\_() (pynlpl.formats.folia.DomainAnnotation method), 437  
\_\_iter\_\_() (pynlpl.formats.folia.EnddatetimeFeature method), 915  
\_\_iter\_\_() (pynlpl.formats.folia.EntitiesLayer method),

- 691
- `__iter__()` (pynlpl.formats.folia.Entity method), 561
- `__iter__()` (pynlpl.formats.folia.Entry method), 149
- `__iter__()` (pynlpl.formats.folia.ErrorDetection method), 973
- `__iter__()` (pynlpl.formats.folia.Event method), 162
- `__iter__()` (pynlpl.formats.folia.Example method), 175
- `__iter__()` (pynlpl.formats.folia.Feature method), 871
- `__iter__()` (pynlpl.formats.folia.Figure method), 188
- `__iter__()` (pynlpl.formats.folia.Gap method), 199
- `__iter__()` (pynlpl.formats.folia.Head method), 212
- `__iter__()` (pynlpl.formats.folia.Headspace method), 796
- `__iter__()` (pynlpl.formats.folia.LangAnnotation method), 459
- `__iter__()` (pynlpl.formats.folia.LemmaAnnotation method), 470
- `__iter__()` (pynlpl.formats.folia.Linebreak method), 225
- `__iter__()` (pynlpl.formats.folia.List method), 238
- `__iter__()` (pynlpl.formats.folia.ListItem method), 251
- `__iter__()` (pynlpl.formats.folia.Metric method), 1051
- `__iter__()` (pynlpl.formats.folia.New method), 984
- `__iter__()` (pynlpl.formats.folia.Note method), 264
- `__iter__()` (pynlpl.formats.folia.Observation method), 573
- `__iter__()` (pynlpl.formats.folia.ObservationLayer method), 702
- `__iter__()` (pynlpl.formats.folia.Original method), 995
- `__iter__()` (pynlpl.formats.folia.Paragraph method), 277
- `__iter__()` (pynlpl.formats.folia.Part method), 290
- `__iter__()` (pynlpl.formats.folia.PhonContent method), 513
- `__iter__()` (pynlpl.formats.folia.PosAnnotation method), 448
- `__iter__()` (pynlpl.formats.folia.Predicate method), 585
- `__iter__()` (pynlpl.formats.folia.Quote method), 303
- `__iter__()` (pynlpl.formats.folia.Reference method), 316
- `__iter__()` (pynlpl.formats.folia.Row method), 329
- `__iter__()` (pynlpl.formats.folia.SemanticRole method), 631
- `__iter__()` (pynlpl.formats.folia.SemanticRolesLayer method), 749
- `__iter__()` (pynlpl.formats.folia.SenseAnnotation method), 481
- `__iter__()` (pynlpl.formats.folia.Sentence method), 344
- `__iter__()` (pynlpl.formats.folia.Sentiment method), 596
- `__iter__()` (pynlpl.formats.folia.SentimentLayer method), 714
- `__iter__()` (pynlpl.formats.folia.Statement method), 608
- `__iter__()` (pynlpl.formats.folia.StatementLayer method), 726
- `__iter__()` (pynlpl.formats.folia.SubjectivityAnnotation method), 492
- `__iter__()` (pynlpl.formats.folia.Suggestion method), 1006
- `__iter__()` (pynlpl.formats.folia.SynsetFeature method), 882
- `__iter__()` (pynlpl.formats.folia.SyntacticUnit method), 620
- `__iter__()` (pynlpl.formats.folia.SyntaxLayer method), 738
- `__iter__()` (pynlpl.formats.folia.Table method), 357
- `__iter__()` (pynlpl.formats.folia.TableHead method), 383
- `__iter__()` (pynlpl.formats.folia.Term method), 370
- `__iter__()` (pynlpl.formats.folia.Text method), 396
- `__iter__()` (pynlpl.formats.folia.TextContent method), 503
- `__iter__()` (pynlpl.formats.folia.TextMarkupCorrection method), 848
- `__iter__()` (pynlpl.formats.folia.TextMarkupError method), 859
- `__iter__()` (pynlpl.formats.folia.TextMarkupGap method), 816
- `__iter__()` (pynlpl.formats.folia.TextMarkupString method), 827
- `__iter__()` (pynlpl.formats.folia.TextMarkupStyle method), 838
- `__iter__()` (pynlpl.formats.folia.TimeSegment method), 643
- `__iter__()` (pynlpl.formats.folia.TimingLayer method), 761
- `__iter__()` (pynlpl.formats.folia.Whitespace method), 409
- `__iter__()` (pynlpl.formats.folia.Word method), 424
- `__len__()` (pynlpl.formats.folia.AbstractAnnotationLayer method), 84
- `__len__()` (pynlpl.formats.folia.AbstractElement method), 34
- `__len__()` (pynlpl.formats.folia.AbstractSpanAnnotation method), 61
- `__len__()` (pynlpl.formats.folia.AbstractStructureElement method), 47
- `__len__()` (pynlpl.formats.folia.AbstractTextMarkup method), 94
- `__len__()` (pynlpl.formats.folia.AbstractTokenAnnotation method), 72
- `__len__()` (pynlpl.formats.folia.ActorFeature method), 893
- `__len__()` (pynlpl.formats.folia.AlignReference method), 1029
- `__len__()` (pynlpl.formats.folia.Alignment method), 1017
- `__len__()` (pynlpl.formats.folia.Alternative method), 929
- `__len__()` (pynlpl.formats.folia.AlternativeLayers method), 940
- `__len__()` (pynlpl.formats.folia.BegindatetimeFeature method), 904
- `__len__()` (pynlpl.formats.folia.Cell method), 110
- `__len__()` (pynlpl.formats.folia.Chunk method), 527
- `__len__()` (pynlpl.formats.folia.ChunkingLayer method), 655

`__len__()` (pynlpl.formats.folia.CoreferenceChain method), 538

`__len__()` (pynlpl.formats.folia.CoreferenceLayer method), 667

`__len__()` (pynlpl.formats.folia.CoreferenceLink method), 773

`__len__()` (pynlpl.formats.folia.Correction method), 951

`__len__()` (pynlpl.formats.folia.Current method), 962

`__len__()` (pynlpl.formats.folia.Definition method), 123

`__len__()` (pynlpl.formats.folia.DependenciesLayer method), 679

`__len__()` (pynlpl.formats.folia.Dependency method), 550

`__len__()` (pynlpl.formats.folia.DependencyDependent method), 785

`__len__()` (pynlpl.formats.folia.Description method), 1040

`__len__()` (pynlpl.formats.folia.Division method), 136

`__len__()` (pynlpl.formats.folia.DomainAnnotation method), 437

`__len__()` (pynlpl.formats.folia.EnddatetimeFeature method), 915

`__len__()` (pynlpl.formats.folia.EntitiesLayer method), 691

`__len__()` (pynlpl.formats.folia.Entity method), 562

`__len__()` (pynlpl.formats.folia.Entry method), 149

`__len__()` (pynlpl.formats.folia.ErrorDetection method), 973

`__len__()` (pynlpl.formats.folia.Event method), 162

`__len__()` (pynlpl.formats.folia.Example method), 175

`__len__()` (pynlpl.formats.folia.Feature method), 871

`__len__()` (pynlpl.formats.folia.Figure method), 188

`__len__()` (pynlpl.formats.folia.Gap method), 199

`__len__()` (pynlpl.formats.folia.Head method), 212

`__len__()` (pynlpl.formats.folia.Headspan method), 796

`__len__()` (pynlpl.formats.folia.LangAnnotation method), 459

`__len__()` (pynlpl.formats.folia.LemmaAnnotation method), 470

`__len__()` (pynlpl.formats.folia.Linebreak method), 225

`__len__()` (pynlpl.formats.folia.List method), 238

`__len__()` (pynlpl.formats.folia.ListItem method), 251

`__len__()` (pynlpl.formats.folia.Metric method), 1051

`__len__()` (pynlpl.formats.folia.New method), 984

`__len__()` (pynlpl.formats.folia.Note method), 264

`__len__()` (pynlpl.formats.folia.Observation method), 573

`__len__()` (pynlpl.formats.folia.ObservationLayer method), 703

`__len__()` (pynlpl.formats.folia.Original method), 995

`__len__()` (pynlpl.formats.folia.Paragraph method), 277

`__len__()` (pynlpl.formats.folia.Part method), 290

`__len__()` (pynlpl.formats.folia.PhonContent method), 514

`__len__()` (pynlpl.formats.folia.PosAnnotation method), 448

`__len__()` (pynlpl.formats.folia.Predicate method), 585

`__len__()` (pynlpl.formats.folia.Quote method), 303

`__len__()` (pynlpl.formats.folia.Reference method), 316

`__len__()` (pynlpl.formats.folia.Row method), 329

`__len__()` (pynlpl.formats.folia.SemanticRole method), 631

`__len__()` (pynlpl.formats.folia.SemanticRolesLayer method), 750

`__len__()` (pynlpl.formats.folia.SenseAnnotation method), 481

`__len__()` (pynlpl.formats.folia.Sentence method), 344

`__len__()` (pynlpl.formats.folia.Sentiment method), 596

`__len__()` (pynlpl.formats.folia.SentimentLayer method), 714

`__len__()` (pynlpl.formats.folia.Statement method), 608

`__len__()` (pynlpl.formats.folia.StatementLayer method), 726

`__len__()` (pynlpl.formats.folia.SubjectivityAnnotation method), 492

`__len__()` (pynlpl.formats.folia.Suggestion method), 1006

`__len__()` (pynlpl.formats.folia.SynsetFeature method), 882

`__len__()` (pynlpl.formats.folia.SyntacticUnit method), 620

`__len__()` (pynlpl.formats.folia.SyntaxLayer method), 738

`__len__()` (pynlpl.formats.folia.Table method), 357

`__len__()` (pynlpl.formats.folia.TableHead method), 383

`__len__()` (pynlpl.formats.folia.Term method), 370

`__len__()` (pynlpl.formats.folia.Text method), 396

`__len__()` (pynlpl.formats.folia.TextContent method), 503

`__len__()` (pynlpl.formats.folia.TextMarkupCorrection method), 848

`__len__()` (pynlpl.formats.folia.TextMarkupError method), 859

`__len__()` (pynlpl.formats.folia.TextMarkupGap method), 816

`__len__()` (pynlpl.formats.folia.TextMarkupString method), 827

`__len__()` (pynlpl.formats.folia.TextMarkupStyle method), 838

`__len__()` (pynlpl.formats.folia.TimeSegment method), 643

`__len__()` (pynlpl.formats.folia.TimingLayer method), 761

`__len__()` (pynlpl.formats.folia.Whitespace method), 409

`__len__()` (pynlpl.formats.folia.Word method), 424

`__str__()` (pynlpl.formats.folia.AbstractAnnotationLayer method), 84

`__str__()` (pynlpl.formats.folia.AbstractElement method), 34

`__str__()` (pynlpl.formats.folia.AbstractSpanAnnotation



- method), 61
- \_\_str\_\_() (pynlpl.formats.folia.AbstractStructureElement method), 47
- \_\_str\_\_() (pynlpl.formats.folia.AbstractTextMarkup method), 94
- \_\_str\_\_() (pynlpl.formats.folia.AbstractTokenAnnotation method), 72
- \_\_str\_\_() (pynlpl.formats.folia.ActorFeature method), 893
- \_\_str\_\_() (pynlpl.formats.folia.AlignReference method), 1029
- \_\_str\_\_() (pynlpl.formats.folia.Alignment method), 1017
- \_\_str\_\_() (pynlpl.formats.folia.AllowTokenAnnotation method), 49
- \_\_str\_\_() (pynlpl.formats.folia.Alternative method), 929
- \_\_str\_\_() (pynlpl.formats.folia.AlternativeLayers method), 940
- \_\_str\_\_() (pynlpl.formats.folia.BegindatetimeFeature method), 904
- \_\_str\_\_() (pynlpl.formats.folia.Cell method), 110
- \_\_str\_\_() (pynlpl.formats.folia.Chunk method), 527
- \_\_str\_\_() (pynlpl.formats.folia.ChunkingLayer method), 655
- \_\_str\_\_() (pynlpl.formats.folia.CoreferenceChain method), 538
- \_\_str\_\_() (pynlpl.formats.folia.CoreferenceLayer method), 667
- \_\_str\_\_() (pynlpl.formats.folia.CoreferenceLink method), 773
- \_\_str\_\_() (pynlpl.formats.folia.Correction method), 951
- \_\_str\_\_() (pynlpl.formats.folia.Current method), 962
- \_\_str\_\_() (pynlpl.formats.folia.Definition method), 123
- \_\_str\_\_() (pynlpl.formats.folia.DependenciesLayer method), 679
- \_\_str\_\_() (pynlpl.formats.folia.Dependency method), 550
- \_\_str\_\_() (pynlpl.formats.folia.DependencyDependent method), 785
- \_\_str\_\_() (pynlpl.formats.folia.Description method), 1040
- \_\_str\_\_() (pynlpl.formats.folia.Division method), 136
- \_\_str\_\_() (pynlpl.formats.folia.DomainAnnotation method), 437
- \_\_str\_\_() (pynlpl.formats.folia.EnddatetimeFeature method), 915
- \_\_str\_\_() (pynlpl.formats.folia.EntitiesLayer method), 691
- \_\_str\_\_() (pynlpl.formats.folia.Entity method), 562
- \_\_str\_\_() (pynlpl.formats.folia.Entry method), 149
- \_\_str\_\_() (pynlpl.formats.folia.ErrorDetection method), 973
- \_\_str\_\_() (pynlpl.formats.folia.Event method), 162
- \_\_str\_\_() (pynlpl.formats.folia.Example method), 175
- \_\_str\_\_() (pynlpl.formats.folia.Feature method), 871
- \_\_str\_\_() (pynlpl.formats.folia.Figure method), 188
- \_\_str\_\_() (pynlpl.formats.folia.Gap method), 199
- \_\_str\_\_() (pynlpl.formats.folia.Head method), 212
- \_\_str\_\_() (pynlpl.formats.folia.Headspeed method), 797
- \_\_str\_\_() (pynlpl.formats.folia.LangAnnotation method), 459
- \_\_str\_\_() (pynlpl.formats.folia.LemmaAnnotation method), 470
- \_\_str\_\_() (pynlpl.formats.folia.Linebreak method), 225
- \_\_str\_\_() (pynlpl.formats.folia.List method), 238
- \_\_str\_\_() (pynlpl.formats.folia.ListItem method), 251
- \_\_str\_\_() (pynlpl.formats.folia.Metric method), 1051
- \_\_str\_\_() (pynlpl.formats.folia.New method), 984
- \_\_str\_\_() (pynlpl.formats.folia.Note method), 264
- \_\_str\_\_() (pynlpl.formats.folia.Observation method), 573
- \_\_str\_\_() (pynlpl.formats.folia.ObservationLayer method), 703
- \_\_str\_\_() (pynlpl.formats.folia.Original method), 995
- \_\_str\_\_() (pynlpl.formats.folia.Paragraph method), 277
- \_\_str\_\_() (pynlpl.formats.folia.Part method), 290
- \_\_str\_\_() (pynlpl.formats.folia.PhonContent method), 514
- \_\_str\_\_() (pynlpl.formats.folia.PosAnnotation method), 448
- \_\_str\_\_() (pynlpl.formats.folia.Predicate method), 585
- \_\_str\_\_() (pynlpl.formats.folia.Quote method), 303
- \_\_str\_\_() (pynlpl.formats.folia.Reference method), 316
- \_\_str\_\_() (pynlpl.formats.folia.Row method), 329
- \_\_str\_\_() (pynlpl.formats.folia.SemanticRole method), 632
- \_\_str\_\_() (pynlpl.formats.folia.SemanticRolesLayer method), 750
- \_\_str\_\_() (pynlpl.formats.folia.SenseAnnotation method), 481
- \_\_str\_\_() (pynlpl.formats.folia.Sentence method), 344
- \_\_str\_\_() (pynlpl.formats.folia.Sentiment method), 597
- \_\_str\_\_() (pynlpl.formats.folia.SentimentLayer method), 714
- \_\_str\_\_() (pynlpl.formats.folia.Statement method), 608
- \_\_str\_\_() (pynlpl.formats.folia.StatementLayer method), 726
- \_\_str\_\_() (pynlpl.formats.folia.SubjectivityAnnotation method), 492
- \_\_str\_\_() (pynlpl.formats.folia.Suggestion method), 1006
- \_\_str\_\_() (pynlpl.formats.folia.SynsetFeature method), 882
- \_\_str\_\_() (pynlpl.formats.folia.SyntacticUnit method), 620
- \_\_str\_\_() (pynlpl.formats.folia.SyntaxLayer method), 738
- \_\_str\_\_() (pynlpl.formats.folia.Table method), 357
- \_\_str\_\_() (pynlpl.formats.folia.TableHead method), 383
- \_\_str\_\_() (pynlpl.formats.folia.Term method), 370
- \_\_str\_\_() (pynlpl.formats.folia.Text method), 396
- \_\_str\_\_() (pynlpl.formats.folia.TextContent method), 503

`__str__()` (pynlpl.formats.folia.TextMarkupCorrection method), 848  
`__str__()` (pynlpl.formats.folia.TextMarkupError method), 859  
`__str__()` (pynlpl.formats.folia.TextMarkupGap method), 816  
`__str__()` (pynlpl.formats.folia.TextMarkupString method), 827  
`__str__()` (pynlpl.formats.folia.TextMarkupStyle method), 838  
`__str__()` (pynlpl.formats.folia.TimeSegment method), 643  
`__str__()` (pynlpl.formats.folia.TimingLayer method), 761  
`__str__()` (pynlpl.formats.folia.Whitespace method), 409  
`__str__()` (pynlpl.formats.folia.Word method), 424

## A

`AbstractAnnotationLayer` (class in pynlpl.formats.folia), 72  
`AbstractElement` (class in pynlpl.formats.folia), 23  
`AbstractExperiment` (class in pynlpl.evaluation), 9  
`AbstractSearch` (class in pynlpl.search), 1059  
`AbstractSearchState` (class in pynlpl.search), 1059  
`AbstractSpanAnnotation` (class in pynlpl.formats.folia), 49  
`AbstractStructureElement` (class in pynlpl.formats.folia), 34  
`AbstractTextMarkup` (class in pynlpl.formats.folia), 84  
`AbstractTokenAnnotation` (class in pynlpl.formats.folia), 61  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractElement attribute), 26  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractStructureElement attribute), 37  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractTextMarkup attribute), 86  
`ACCEPTED_DATA` (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63  
`ACCEPTED_DATA` (pynlpl.formats.folia.ActorFeature attribute), 884  
`ACCEPTED_DATA` (pynlpl.formats.folia.Alignment attribute), 1009  
`ACCEPTED_DATA` (pynlpl.formats.folia.AlignReference attribute), 1020  
`ACCEPTED_DATA` (pynlpl.formats.folia.Alternative attribute), 919  
`ACCEPTED_DATA` (pynlpl.formats.folia.AlternativeLayers attribute), 931  
`ACCEPTED_DATA` (pynlpl.formats.folia.BegindatetimeFeature attribute), 895  
`ACCEPTED_DATA` (pynlpl.formats.folia.Cell attribute), 100  
`ACCEPTED_DATA` (pynlpl.formats.folia.Chunk attribute), 518  
`ACCEPTED_DATA` (pynlpl.formats.folia.ChunkingLayer attribute), 646  
`ACCEPTED_DATA` (pynlpl.formats.folia.CoreferenceChain attribute), 529  
`ACCEPTED_DATA` (pynlpl.formats.folia.CoreferenceLayer attribute), 658  
`ACCEPTED_DATA` (pynlpl.formats.folia.CoreferenceLink attribute), 764  
`ACCEPTED_DATA` (pynlpl.formats.folia.Correction attribute), 944  
`ACCEPTED_DATA` (pynlpl.formats.folia.Current attribute), 953  
`ACCEPTED_DATA` (pynlpl.formats.folia.Definition attribute), 113  
`ACCEPTED_DATA` (pynlpl.formats.folia.DependenciesLayer attribute), 670  
`ACCEPTED_DATA` (pynlpl.formats.folia.Dependency attribute), 541  
`ACCEPTED_DATA` (pynlpl.formats.folia.DependencyDependent attribute), 776  
`ACCEPTED_DATA` (pynlpl.formats.folia.Description attribute), 1031  
`ACCEPTED_DATA` (pynlpl.formats.folia.Division attribute), 126  
`ACCEPTED_DATA` (pynlpl.formats.folia.DomainAnnotation attribute), 428  
`ACCEPTED_DATA` (pynlpl.formats.folia.EnddatetimeFeature attribute), 906  
`ACCEPTED_DATA` (pynlpl.formats.folia.EntitiesLayer attribute), 682  
`ACCEPTED_DATA` (pynlpl.formats.folia.Entity attribute), 553  
`ACCEPTED_DATA` (pynlpl.formats.folia.Entry attribute), 139  
`ACCEPTED_DATA` (pynlpl.formats.folia.ErrorDetection attribute), 964  
`ACCEPTED_DATA` (pynlpl.formats.folia.Event attribute), 152  
`ACCEPTED_DATA` (pynlpl.formats.folia.Example attribute), 165  
`ACCEPTED_DATA` (pynlpl.formats.folia.Feature attribute), 862  
`ACCEPTED_DATA` (pynlpl.formats.folia.Figure attribute), 178  
`ACCEPTED_DATA` (pynlpl.formats.folia.Gap attribute), 191  
`ACCEPTED_DATA` (pynlpl.formats.folia.Head attribute), 202

- ACCEPTED\_DATA (pynlpl.formats.folia.Headspan attribute), 787
- ACCEPTED\_DATA (pynlpl.formats.folia.LangAnnotation attribute), 450
- ACCEPTED\_DATA (pynlpl.formats.folia.LemmaAnnotation attribute), 461
- ACCEPTED\_DATA (pynlpl.formats.folia.Linebreak attribute), 215
- ACCEPTED\_DATA (pynlpl.formats.folia.List attribute), 228
- ACCEPTED\_DATA (pynlpl.formats.folia.ListItem attribute), 241
- ACCEPTED\_DATA (pynlpl.formats.folia.Metric attribute), 1042
- ACCEPTED\_DATA (pynlpl.formats.folia.New attribute), 975
- ACCEPTED\_DATA (pynlpl.formats.folia.Note attribute), 254
- ACCEPTED\_DATA (pynlpl.formats.folia.Observation attribute), 564
- ACCEPTED\_DATA (pynlpl.formats.folia.ObservationLayer attribute), 693
- ACCEPTED\_DATA (pynlpl.formats.folia.Original attribute), 986
- ACCEPTED\_DATA (pynlpl.formats.folia.Paragraph attribute), 267
- ACCEPTED\_DATA (pynlpl.formats.folia.Part attribute), 280
- ACCEPTED\_DATA (pynlpl.formats.folia.PhonContent attribute), 505
- ACCEPTED\_DATA (pynlpl.formats.folia.PosAnnotation attribute), 439
- ACCEPTED\_DATA (pynlpl.formats.folia.Predicate attribute), 576
- ACCEPTED\_DATA (pynlpl.formats.folia.Quote attribute), 293
- ACCEPTED\_DATA (pynlpl.formats.folia.Reference attribute), 306
- ACCEPTED\_DATA (pynlpl.formats.folia.Row attribute), 319
- ACCEPTED\_DATA (pynlpl.formats.folia.SemanticRole attribute), 622
- ACCEPTED\_DATA (pynlpl.formats.folia.SemanticRolesLayer attribute), 740
- ACCEPTED\_DATA (pynlpl.formats.folia.SenseAnnotation attribute), 472
- ACCEPTED\_DATA (pynlpl.formats.folia.Sentence attribute), 332
- ACCEPTED\_DATA (pynlpl.formats.folia.Sentiment attribute), 587
- ACCEPTED\_DATA (pynlpl.formats.folia.SentimentLayer attribute), 705
- ACCEPTED\_DATA (pynlpl.formats.folia.Statement attribute), 599
- ACCEPTED\_DATA (pynlpl.formats.folia.StatementLayer attribute), 717
- ACCEPTED\_DATA (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- ACCEPTED\_DATA (pynlpl.formats.folia.Suggestion attribute), 997
- ACCEPTED\_DATA (pynlpl.formats.folia.SynsetFeature attribute), 873
- ACCEPTED\_DATA (pynlpl.formats.folia.SyntacticUnit attribute), 611
- ACCEPTED\_DATA (pynlpl.formats.folia.SyntaxLayer attribute), 729
- ACCEPTED\_DATA (pynlpl.formats.folia.Table attribute), 347
- ACCEPTED\_DATA (pynlpl.formats.folia.TableHead attribute), 373
- ACCEPTED\_DATA (pynlpl.formats.folia.Term attribute), 360
- ACCEPTED\_DATA (pynlpl.formats.folia.Text attribute), 386
- ACCEPTED\_DATA (pynlpl.formats.folia.TextContent attribute), 495
- ACCEPTED\_DATA (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- ACCEPTED\_DATA (pynlpl.formats.folia.TextMarkupError attribute), 851
- ACCEPTED\_DATA (pynlpl.formats.folia.TextMarkupGap attribute), 808
- ACCEPTED\_DATA (pynlpl.formats.folia.TextMarkupString attribute), 819
- ACCEPTED\_DATA (pynlpl.formats.folia.TextMarkupStyle attribute), 829
- ACCEPTED\_DATA (pynlpl.formats.folia.TimeSegment attribute), 634
- ACCEPTED\_DATA (pynlpl.formats.folia.TimingLayer attribute), 752
- ACCEPTED\_DATA (pynlpl.formats.folia.Whitespace attribute), 399
- ACCEPTED\_DATA (pynlpl.formats.folia.Word attribute), 412
- accepts() (pynlpl.formats.folia.AbstractAnnotationLayer class method), 75
- accepts() (pynlpl.formats.folia.AbstractElement class method), 26
- accepts() (pynlpl.formats.folia.AbstractSpanAnnotation class method), 52
- accepts() (pynlpl.formats.folia.AbstractStructureElement class method), 38
- accepts() (pynlpl.formats.folia.AbstractTextMarkup class method), 87
- accepts() (pynlpl.formats.folia.AbstractTokenAnnotation class method), 64
- accepts() (pynlpl.formats.folia.ActorFeature class method), 885

`accepts()` (`pynlpl.formats.folia.Alignment` class method), 1009

`accepts()` (`pynlpl.formats.folia.AlignReference` class method), 1020

`accepts()` (`pynlpl.formats.folia.Alternative` class method), 919

`accepts()` (`pynlpl.formats.folia.AlternativeLayers` class method), 932

`accepts()` (`pynlpl.formats.folia.BegindatetimeFeature` class method), 896

`accepts()` (`pynlpl.formats.folia.Cell` class method), 101

`accepts()` (`pynlpl.formats.folia.Chunk` class method), 518

`accepts()` (`pynlpl.formats.folia.ChunkingLayer` class method), 647

`accepts()` (`pynlpl.formats.folia.CoreferenceChain` class method), 530

`accepts()` (`pynlpl.formats.folia.CoreferenceLayer` class method), 659

`accepts()` (`pynlpl.formats.folia.CoreferenceLink` class method), 765

`accepts()` (`pynlpl.formats.folia.Correction` class method), 944

`accepts()` (`pynlpl.formats.folia.Current` class method), 954

`accepts()` (`pynlpl.formats.folia.Definition` class method), 114

`accepts()` (`pynlpl.formats.folia.DependenciesLayer` class method), 670

`accepts()` (`pynlpl.formats.folia.Dependency` class method), 542

`accepts()` (`pynlpl.formats.folia.DependencyDependent` class method), 776

`accepts()` (`pynlpl.formats.folia.Description` class method), 1032

`accepts()` (`pynlpl.formats.folia.Division` class method), 127

`accepts()` (`pynlpl.formats.folia.DomainAnnotation` class method), 429

`accepts()` (`pynlpl.formats.folia.EnddatetimeFeature` class method), 907

`accepts()` (`pynlpl.formats.folia.EntitiesLayer` class method), 682

`accepts()` (`pynlpl.formats.folia.Entity` class method), 553

`accepts()` (`pynlpl.formats.folia.Entry` class method), 140

`accepts()` (`pynlpl.formats.folia.ErrorDetection` class method), 965

`accepts()` (`pynlpl.formats.folia.Event` class method), 153

`accepts()` (`pynlpl.formats.folia.Example` class method), 166

`accepts()` (`pynlpl.formats.folia.Feature` class method), 863

`accepts()` (`pynlpl.formats.folia.Figure` class method), 179

`accepts()` (`pynlpl.formats.folia.Gap` class method), 191

`accepts()` (`pynlpl.formats.folia.Head` class method), 203

`accepts()` (`pynlpl.formats.folia.Headspan` class method), 788

`accepts()` (`pynlpl.formats.folia.LangAnnotation` class method), 451

`accepts()` (`pynlpl.formats.folia.LemmaAnnotation` class method), 462

`accepts()` (`pynlpl.formats.folia.Linebreak` class method), 216

`accepts()` (`pynlpl.formats.folia.List` class method), 229

`accepts()` (`pynlpl.formats.folia.ListItem` class method), 242

`accepts()` (`pynlpl.formats.folia.Metric` class method), 1043

`accepts()` (`pynlpl.formats.folia.New` class method), 976

`accepts()` (`pynlpl.formats.folia.Note` class method), 255

`accepts()` (`pynlpl.formats.folia.Observation` class method), 565

`accepts()` (`pynlpl.formats.folia.ObservationLayer` class method), 694

`accepts()` (`pynlpl.formats.folia.Original` class method), 987

`accepts()` (`pynlpl.formats.folia.Paragraph` class method), 268

`accepts()` (`pynlpl.formats.folia.Part` class method), 281

`accepts()` (`pynlpl.formats.folia.PhonContent` class method), 506

`accepts()` (`pynlpl.formats.folia.PosAnnotation` class method), 440

`accepts()` (`pynlpl.formats.folia.Predicate` class method), 576

`accepts()` (`pynlpl.formats.folia.Quote` class method), 294

`accepts()` (`pynlpl.formats.folia.Reference` class method), 307

`accepts()` (`pynlpl.formats.folia.Row` class method), 320

`accepts()` (`pynlpl.formats.folia.SemanticRole` class method), 623

`accepts()` (`pynlpl.formats.folia.SemanticRolesLayer` class method), 741

`accepts()` (`pynlpl.formats.folia.SenseAnnotation` class method), 473

`accepts()` (`pynlpl.formats.folia.Sentence` class method), 334

`accepts()` (`pynlpl.formats.folia.Sentiment` class method), 588

`accepts()` (`pynlpl.formats.folia.SentimentLayer` class method), 706

`accepts()` (`pynlpl.formats.folia.Statement` class method), 600

`accepts()` (`pynlpl.formats.folia.StatementLayer` class method), 717

`accepts()` (`pynlpl.formats.folia.SubjectivityAnnotation` class method), 484

`accepts()` (`pynlpl.formats.folia.Suggestion` class method), 998

`accepts()` (`pynlpl.formats.folia.SynsetFeature` class method), 874

- accepts() (pynlpl.formats.folia.SyntacticUnit class method), 611
- accepts() (pynlpl.formats.folia.SyntaxLayer class method), 729
- accepts() (pynlpl.formats.folia.Table class method), 347
- accepts() (pynlpl.formats.folia.TableHead class method), 373
- accepts() (pynlpl.formats.folia.Term class method), 360
- accepts() (pynlpl.formats.folia.Text class method), 387
- accepts() (pynlpl.formats.folia.TextContent class method), 496
- accepts() (pynlpl.formats.folia.TextMarkupCorrection class method), 841
- accepts() (pynlpl.formats.folia.TextMarkupError class method), 851
- accepts() (pynlpl.formats.folia.TextMarkupGap class method), 809
- accepts() (pynlpl.formats.folia.TextMarkupString class method), 819
- accepts() (pynlpl.formats.folia.TextMarkupStyle class method), 830
- accepts() (pynlpl.formats.folia.TimeSegment class method), 635
- accepts() (pynlpl.formats.folia.TimingLayer class method), 753
- accepts() (pynlpl.formats.folia.Whitespace class method), 400
- accepts() (pynlpl.formats.folia.Word class method), 414
- accessible() (pynlpl.statistics.MarkovChain method), 1063
- accuracy() (pynlpl.evaluation.ClassEvaluation method), 9
- ActorFeature (class in pynlpl.formats.folia), 882
- add() (pynlpl.datatypes.PatternSet method), 5
- add() (pynlpl.formats.folia.AbstractAnnotationLayer method), 75
- add() (pynlpl.formats.folia.AbstractElement method), 26
- add() (pynlpl.formats.folia.AbstractSpanAnnotation method), 52
- add() (pynlpl.formats.folia.AbstractStructureElement method), 38
- add() (pynlpl.formats.folia.AbstractTextMarkup method), 87
- add() (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- add() (pynlpl.formats.folia.ActorFeature method), 885
- add() (pynlpl.formats.folia.Alignment method), 1009
- add() (pynlpl.formats.folia.AlignReference method), 1020
- add() (pynlpl.formats.folia.Alternative method), 919
- add() (pynlpl.formats.folia.AlternativeLayers method), 932
- add() (pynlpl.formats.folia.BegindatetimeFeature method), 896
- add() (pynlpl.formats.folia.Cell method), 101
- add() (pynlpl.formats.folia.Chunk method), 518
- add() (pynlpl.formats.folia.ChunkingLayer method), 647
- add() (pynlpl.formats.folia.CoreferenceChain method), 530
- add() (pynlpl.formats.folia.CoreferenceLayer method), 659
- add() (pynlpl.formats.folia.CoreferenceLink method), 765
- add() (pynlpl.formats.folia.Correction method), 944
- add() (pynlpl.formats.folia.Current method), 954
- add() (pynlpl.formats.folia.Definition method), 114
- add() (pynlpl.formats.folia.DependenciesLayer method), 670
- add() (pynlpl.formats.folia.Dependency method), 542
- add() (pynlpl.formats.folia.DependencyDependent method), 776
- add() (pynlpl.formats.folia.Description method), 1032
- add() (pynlpl.formats.folia.Division method), 127
- add() (pynlpl.formats.folia.Document method), 17
- add() (pynlpl.formats.folia.DomainAnnotation method), 429
- add() (pynlpl.formats.folia.EnddatetimeFeature method), 907
- add() (pynlpl.formats.folia.EntitiesLayer method), 682
- add() (pynlpl.formats.folia.Entity method), 553
- add() (pynlpl.formats.folia.Entry method), 140
- add() (pynlpl.formats.folia.ErrorDetection method), 965
- add() (pynlpl.formats.folia.Event method), 153
- add() (pynlpl.formats.folia.Example method), 166
- add() (pynlpl.formats.folia.Feature method), 863
- add() (pynlpl.formats.folia.Figure method), 179
- add() (pynlpl.formats.folia.Gap method), 191
- add() (pynlpl.formats.folia.Head method), 203
- add() (pynlpl.formats.folia.Headspan method), 788
- add() (pynlpl.formats.folia.LangAnnotation method), 451
- add() (pynlpl.formats.folia.LemmaAnnotation method), 462
- add() (pynlpl.formats.folia.Linebreak method), 216
- add() (pynlpl.formats.folia.List method), 229
- add() (pynlpl.formats.folia.ListItem method), 242
- add() (pynlpl.formats.folia.Metric method), 1043
- add() (pynlpl.formats.folia.New method), 976
- add() (pynlpl.formats.folia.Note method), 255
- add() (pynlpl.formats.folia.Observation method), 565
- add() (pynlpl.formats.folia.ObservationLayer method), 694
- add() (pynlpl.formats.folia.Original method), 987
- add() (pynlpl.formats.folia.Paragraph method), 268
- add() (pynlpl.formats.folia.Part method), 281
- add() (pynlpl.formats.folia.PhonContent method), 506
- add() (pynlpl.formats.folia.PosAnnotation method), 440
- add() (pynlpl.formats.folia.Predicate method), 576
- add() (pynlpl.formats.folia.Quote method), 294
- add() (pynlpl.formats.folia.Reference method), 307



- `add()` (`pynlpl.formats.folia.Row` method), 320
- `add()` (`pynlpl.formats.folia.SemanticRole` method), 623
- `add()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 741
- `add()` (`pynlpl.formats.folia.SenseAnnotation` method), 473
- `add()` (`pynlpl.formats.folia.Sentence` method), 334
- `add()` (`pynlpl.formats.folia.Sentiment` method), 588
- `add()` (`pynlpl.formats.folia.SentimentLayer` method), 706
- `add()` (`pynlpl.formats.folia.Statement` method), 600
- `add()` (`pynlpl.formats.folia.StatementLayer` method), 717
- `add()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 484
- `add()` (`pynlpl.formats.folia.Suggestion` method), 998
- `add()` (`pynlpl.formats.folia.SynsetFeature` method), 874
- `add()` (`pynlpl.formats.folia.SyntacticUnit` method), 611
- `add()` (`pynlpl.formats.folia.SyntaxLayer` method), 729
- `add()` (`pynlpl.formats.folia.Table` method), 347
- `add()` (`pynlpl.formats.folia.TableHead` method), 373
- `add()` (`pynlpl.formats.folia.Term` method), 360
- `add()` (`pynlpl.formats.folia.Text` method), 387
- `add()` (`pynlpl.formats.folia.TextContent` method), 496
- `add()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 841
- `add()` (`pynlpl.formats.folia.TextMarkupError` method), 851
- `add()` (`pynlpl.formats.folia.TextMarkupGap` method), 809
- `add()` (`pynlpl.formats.folia.TextMarkupString` method), 819
- `add()` (`pynlpl.formats.folia.TextMarkupStyle` method), 830
- `add()` (`pynlpl.formats.folia.TimeSegment` method), 635
- `add()` (`pynlpl.formats.folia.TimingLayer` method), 753
- `add()` (`pynlpl.formats.folia.Whitespace` method), 400
- `add()` (`pynlpl.formats.folia.Word` method), 414
- `addable()` (`pynlpl.formats.folia.AbstractAnnotationLayer` class method), 75
- `addable()` (`pynlpl.formats.folia.AbstractElement` class method), 26
- `addable()` (`pynlpl.formats.folia.AbstractSpanAnnotation` class method), 52
- `addable()` (`pynlpl.formats.folia.AbstractStructureElement` class method), 38
- `addable()` (`pynlpl.formats.folia.AbstractTextMarkup` class method), 87
- `addable()` (`pynlpl.formats.folia.AbstractTokenAnnotation` class method), 64
- `addable()` (`pynlpl.formats.folia.ActorFeature` class method), 885
- `addable()` (`pynlpl.formats.folia.Alignment` class method), 1009
- `addable()` (`pynlpl.formats.folia.AlignReference` class method), 1020
- `addable()` (`pynlpl.formats.folia.Alternative` class method), 919
- `addable()` (`pynlpl.formats.folia.AlternativeLayers` class method), 932
- `addable()` (`pynlpl.formats.folia.BegindatetimeFeature` class method), 896
- `addable()` (`pynlpl.formats.folia.Cell` class method), 101
- `addable()` (`pynlpl.formats.folia.Chunk` class method), 518
- `addable()` (`pynlpl.formats.folia.ChunkingLayer` class method), 647
- `addable()` (`pynlpl.formats.folia.CoreferenceChain` class method), 530
- `addable()` (`pynlpl.formats.folia.CoreferenceLayer` class method), 659
- `addable()` (`pynlpl.formats.folia.CoreferenceLink` class method), 765
- `addable()` (`pynlpl.formats.folia.Correction` class method), 944
- `addable()` (`pynlpl.formats.folia.Current` class method), 954
- `addable()` (`pynlpl.formats.folia.Definition` class method), 114
- `addable()` (`pynlpl.formats.folia.DependenciesLayer` class method), 670
- `addable()` (`pynlpl.formats.folia.Dependency` class method), 542
- `addable()` (`pynlpl.formats.folia.DependencyDependent` class method), 777
- `addable()` (`pynlpl.formats.folia.Description` class method), 1032
- `addable()` (`pynlpl.formats.folia.Division` class method), 127
- `addable()` (`pynlpl.formats.folia.DomainAnnotation` class method), 429
- `addable()` (`pynlpl.formats.folia.EnddatetimeFeature` class method), 907
- `addable()` (`pynlpl.formats.folia.EntitiesLayer` class method), 682
- `addable()` (`pynlpl.formats.folia.Entity` class method), 553
- `addable()` (`pynlpl.formats.folia.Entry` class method), 140
- `addable()` (`pynlpl.formats.folia.ErrorDetection` class method), 965
- `addable()` (`pynlpl.formats.folia.Event` class method), 153
- `addable()` (`pynlpl.formats.folia.Example` class method), 166
- `addable()` (`pynlpl.formats.folia.Feature` class method), 863
- `addable()` (`pynlpl.formats.folia.Figure` class method), 179
- `addable()` (`pynlpl.formats.folia.Gap` class method), 191
- `addable()` (`pynlpl.formats.folia.Head` class method), 203
- `addable()` (`pynlpl.formats.folia.Headspan` class method), 788
- `addable()` (`pynlpl.formats.folia.LangAnnotation` class method), 451
- `addable()` (`pynlpl.formats.folia.LemmaAnnotation` class

- method), 462
- addable() (pynlpl.formats.folia.Linebreak class method), 216
- addable() (pynlpl.formats.folia.List class method), 229
- addable() (pynlpl.formats.folia.ListItem class method), 242
- addable() (pynlpl.formats.folia.Metric class method), 1043
- addable() (pynlpl.formats.folia.New class method), 976
- addable() (pynlpl.formats.folia.Note class method), 255
- addable() (pynlpl.formats.folia.Observation class method), 565
- addable() (pynlpl.formats.folia.ObservationLayer class method), 694
- addable() (pynlpl.formats.folia.Original class method), 987
- addable() (pynlpl.formats.folia.Paragraph class method), 268
- addable() (pynlpl.formats.folia.Part class method), 281
- addable() (pynlpl.formats.folia.PhonContent class method), 506
- addable() (pynlpl.formats.folia.PosAnnotation class method), 440
- addable() (pynlpl.formats.folia.Predicate class method), 577
- addable() (pynlpl.formats.folia.Quote class method), 294
- addable() (pynlpl.formats.folia.Reference class method), 307
- addable() (pynlpl.formats.folia.Row class method), 320
- addable() (pynlpl.formats.folia.SemanticRole class method), 623
- addable() (pynlpl.formats.folia.SemanticRolesLayer class method), 741
- addable() (pynlpl.formats.folia.SenseAnnotation class method), 473
- addable() (pynlpl.formats.folia.Sentence class method), 334
- addable() (pynlpl.formats.folia.Sentiment class method), 588
- addable() (pynlpl.formats.folia.SentimentLayer class method), 706
- addable() (pynlpl.formats.folia.Statement class method), 600
- addable() (pynlpl.formats.folia.StatementLayer class method), 717
- addable() (pynlpl.formats.folia.SubjectivityAnnotation class method), 484
- addable() (pynlpl.formats.folia.Suggestion class method), 998
- addable() (pynlpl.formats.folia.SynsetFeature class method), 874
- addable() (pynlpl.formats.folia.SyntacticUnit class method), 612
- addable() (pynlpl.formats.folia.SyntaxLayer class method), 729
- addable() (pynlpl.formats.folia.Table class method), 348
- addable() (pynlpl.formats.folia.TableHead class method), 374
- addable() (pynlpl.formats.folia.Term class method), 361
- addable() (pynlpl.formats.folia.Text class method), 387
- addable() (pynlpl.formats.folia.TextContent class method), 496
- addable() (pynlpl.formats.folia.TextMarkupCorrection class method), 841
- addable() (pynlpl.formats.folia.TextMarkupError class method), 852
- addable() (pynlpl.formats.folia.TextMarkupGap class method), 809
- addable() (pynlpl.formats.folia.TextMarkupString class method), 819
- addable() (pynlpl.formats.folia.TextMarkupStyle class method), 830
- addable() (pynlpl.formats.folia.TimeSegment class method), 635
- addable() (pynlpl.formats.folia.TimingLayer class method), 753
- addable() (pynlpl.formats.folia.Whitespace class method), 400
- addable() (pynlpl.formats.folia.Word class method), 414
- addidsuffix() (pynlpl.formats.folia.AbstractAnnotationLayer method), 75
- addidsuffix() (pynlpl.formats.folia.AbstractElement method), 27
- addidsuffix() (pynlpl.formats.folia.AbstractSpanAnnotation method), 53
- addidsuffix() (pynlpl.formats.folia.AbstractStructureElement method), 38
- addidsuffix() (pynlpl.formats.folia.AbstractTextMarkup method), 87
- addidsuffix() (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- addidsuffix() (pynlpl.formats.folia.ActorFeature method), 886
- addidsuffix() (pynlpl.formats.folia.Alignment method), 1010
- addidsuffix() (pynlpl.formats.folia.AlignReference method), 1021
- addidsuffix() (pynlpl.formats.folia.Alternative method), 920
- addidsuffix() (pynlpl.formats.folia.AlternativeLayers method), 932
- addidsuffix() (pynlpl.formats.folia.BegindatetimeFeature method), 897
- addidsuffix() (pynlpl.formats.folia.Cell method), 101
- addidsuffix() (pynlpl.formats.folia.Chunk method), 519
- addidsuffix() (pynlpl.formats.folia.ChunkingLayer method), 647
- addidsuffix() (pynlpl.formats.folia.CoreferenceChain

method), 530  
addidsuffix() (pynlpl.formats.folia.CoreferenceLayer method), 659  
addidsuffix() (pynlpl.formats.folia.CoreferenceLink method), 765  
addidsuffix() (pynlpl.formats.folia.Correction method), 945  
addidsuffix() (pynlpl.formats.folia.Current method), 954  
addidsuffix() (pynlpl.formats.folia.Definition method), 114  
addidsuffix() (pynlpl.formats.folia.DependenciesLayer method), 671  
addidsuffix() (pynlpl.formats.folia.Dependency method), 542  
addidsuffix() (pynlpl.formats.folia.DependencyDependent method), 777  
addidsuffix() (pynlpl.formats.folia.Description method), 1032  
addidsuffix() (pynlpl.formats.folia.Division method), 127  
addidsuffix() (pynlpl.formats.folia.DomainAnnotation method), 429  
addidsuffix() (pynlpl.formats.folia.EnddatetimeFeature method), 908  
addidsuffix() (pynlpl.formats.folia.EntitiesLayer method), 683  
addidsuffix() (pynlpl.formats.folia.Entity method), 554  
addidsuffix() (pynlpl.formats.folia.Entry method), 140  
addidsuffix() (pynlpl.formats.folia.ErrorDetection method), 965  
addidsuffix() (pynlpl.formats.folia.Event method), 153  
addidsuffix() (pynlpl.formats.folia.Example method), 166  
addidsuffix() (pynlpl.formats.folia.Feature method), 863  
addidsuffix() (pynlpl.formats.folia.Figure method), 179  
addidsuffix() (pynlpl.formats.folia.Gap method), 192  
addidsuffix() (pynlpl.formats.folia.Head method), 203  
addidsuffix() (pynlpl.formats.folia.Headspan method), 788  
addidsuffix() (pynlpl.formats.folia.LangAnnotation method), 451  
addidsuffix() (pynlpl.formats.folia.LemmaAnnotation method), 462  
addidsuffix() (pynlpl.formats.folia.Linebreak method), 216  
addidsuffix() (pynlpl.formats.folia.List method), 229  
addidsuffix() (pynlpl.formats.folia.ListItem method), 242  
addidsuffix() (pynlpl.formats.folia.Metric method), 1043  
addidsuffix() (pynlpl.formats.folia.New method), 976  
addidsuffix() (pynlpl.formats.folia.Note method), 255  
addidsuffix() (pynlpl.formats.folia.Observation method), 565  
addidsuffix() (pynlpl.formats.folia.ObservationLayer method), 694  
addidsuffix() (pynlpl.formats.folia.Original method), 987  
addidsuffix() (pynlpl.formats.folia.Paragraph method), 268  
addidsuffix() (pynlpl.formats.folia.Part method), 281  
addidsuffix() (pynlpl.formats.folia.PhonContent method), 506  
addidsuffix() (pynlpl.formats.folia.PosAnnotation method), 440  
addidsuffix() (pynlpl.formats.folia.Predicate method), 577  
addidsuffix() (pynlpl.formats.folia.Quote method), 294  
addidsuffix() (pynlpl.formats.folia.Reference method), 307  
addidsuffix() (pynlpl.formats.folia.Row method), 320  
addidsuffix() (pynlpl.formats.folia.SemanticRole method), 623  
addidsuffix() (pynlpl.formats.folia.SemanticRolesLayer method), 741  
addidsuffix() (pynlpl.formats.folia.SenseAnnotation method), 473  
addidsuffix() (pynlpl.formats.folia.Sentence method), 334  
addidsuffix() (pynlpl.formats.folia.Sentiment method), 588  
addidsuffix() (pynlpl.formats.folia.SentimentLayer method), 706  
addidsuffix() (pynlpl.formats.folia.Statement method), 600  
addidsuffix() (pynlpl.formats.folia.StatementLayer method), 718  
addidsuffix() (pynlpl.formats.folia.SubjectivityAnnotation method), 484  
addidsuffix() (pynlpl.formats.folia.Suggestion method), 998  
addidsuffix() (pynlpl.formats.folia.SynsetFeature method), 874  
addidsuffix() (pynlpl.formats.folia.SyntacticUnit method), 612  
addidsuffix() (pynlpl.formats.folia.SyntaxLayer method), 730  
addidsuffix() (pynlpl.formats.folia.Table method), 348  
addidsuffix() (pynlpl.formats.folia.TableHead method), 374  
addidsuffix() (pynlpl.formats.folia.Term method), 361  
addidsuffix() (pynlpl.formats.folia.Text method), 387  
addidsuffix() (pynlpl.formats.folia.TextContent method), 496  
addidsuffix() (pynlpl.formats.folia.TextMarkupCorrection method), 841  
addidsuffix() (pynlpl.formats.folia.TextMarkupError method), 852  
addidsuffix() (pynlpl.formats.folia.TextMarkupGap method), 809  
addidsuffix() (pynlpl.formats.folia.TextMarkupString method), 820  
addidsuffix() (pynlpl.formats.folia.TextMarkupStyle method), 830



[addsuffix\(\) \(pynlpl.formats.folia.TimeSegment method\), 635](#)  
[addsuffix\(\) \(pynlpl.formats.folia.TimingLayer method\), 753](#)  
[addsuffix\(\) \(pynlpl.formats.folia.Whitespace method\), 400](#)  
[addsuffix\(\) \(pynlpl.formats.folia.Word method\), 414](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 75](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractElement method\), 27](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 53](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 38](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractTextMarkup method\), 87](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AbstractTokenAnnotation method\), 64](#)  
[addtoindex\(\) \(pynlpl.formats.folia.ActorFeature method\), 886](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Alignment method\), 1010](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AlignReference method\), 1021](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Alternative method\), 920](#)  
[addtoindex\(\) \(pynlpl.formats.folia.AlternativeLayers method\), 932](#)  
[addtoindex\(\) \(pynlpl.formats.folia.BegindatetimeFeature method\), 897](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Cell method\), 101](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Chunk method\), 519](#)  
[addtoindex\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 647](#)  
[addtoindex\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 530](#)  
[addtoindex\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 659](#)  
[addtoindex\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 765](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Correction method\), 945](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Current method\), 954](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Definition method\), 114](#)  
[addtoindex\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 671](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Dependency method\), 542](#)  
[addtoindex\(\) \(pynlpl.formats.folia.DependencyDependent method\), 777](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Description method\), 1032](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Division method\), 127](#)  
[addtoindex\(\) \(pynlpl.formats.folia.DomainAnnotation method\), 429](#)  
[addtoindex\(\) \(pynlpl.formats.folia.EnddatetimeFeature method\), 908](#)  
[addtoindex\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 683](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Entity method\), 554](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Entry method\), 140](#)  
[addtoindex\(\) \(pynlpl.formats.folia.ErrorDetection method\), 965](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Event method\), 153](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Example method\), 166](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Feature method\), 863](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Figure method\), 179](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Gap method\), 192](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Head method\), 203](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Headspan method\), 789](#)  
[addtoindex\(\) \(pynlpl.formats.folia.LangAnnotation method\), 451](#)  
[addtoindex\(\) \(pynlpl.formats.folia.LemmaAnnotation method\), 462](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Linebreak method\), 216](#)  
[addtoindex\(\) \(pynlpl.formats.folia.List method\), 229](#)  
[addtoindex\(\) \(pynlpl.formats.folia.ListItem method\), 242](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Metric method\), 1043](#)  
[addtoindex\(\) \(pynlpl.formats.folia.New method\), 976](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Note method\), 255](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Observation method\), 565](#)  
[addtoindex\(\) \(pynlpl.formats.folia.ObservationLayer method\), 694](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Original method\), 987](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Paragraph method\), 268](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Part method\), 281](#)  
[addtoindex\(\) \(pynlpl.formats.folia.PhonContent method\), 507](#)  
[addtoindex\(\) \(pynlpl.formats.folia.PosAnnotation method\), 440](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Predicate method\), 577](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Quote method\), 294](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Reference method\), 307](#)  
[addtoindex\(\) \(pynlpl.formats.folia.Row method\), 320](#)  
[addtoindex\(\) \(pynlpl.formats.folia.SemanticRole method\), 624](#)  
[addtoindex\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 741](#)  
[addtoindex\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 473](#)

`addtoindex()` (`pynlpl.formats.folia.Sentence` method), 334  
`addtoindex()` (`pynlpl.formats.folia.Sentiment` method), 589  
`addtoindex()` (`pynlpl.formats.folia.SentimentLayer` method), 706  
`addtoindex()` (`pynlpl.formats.folia.Statement` method), 600  
`addtoindex()` (`pynlpl.formats.folia.StatementLayer` method), 718  
`addtoindex()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 484  
`addtoindex()` (`pynlpl.formats.folia.Suggestion` method), 998  
`addtoindex()` (`pynlpl.formats.folia.SynsetFeature` method), 874  
`addtoindex()` (`pynlpl.formats.folia.SyntacticUnit` method), 612  
`addtoindex()` (`pynlpl.formats.folia.SyntaxLayer` method), 730  
`addtoindex()` (`pynlpl.formats.folia.Table` method), 348  
`addtoindex()` (`pynlpl.formats.folia.TableHead` method), 374  
`addtoindex()` (`pynlpl.formats.folia.Term` method), 361  
`addtoindex()` (`pynlpl.formats.folia.Text` method), 387  
`addtoindex()` (`pynlpl.formats.folia.TextContent` method), 496  
`addtoindex()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 841  
`addtoindex()` (`pynlpl.formats.folia.TextMarkupError` method), 852  
`addtoindex()` (`pynlpl.formats.folia.TextMarkupGap` method), 809  
`addtoindex()` (`pynlpl.formats.folia.TextMarkupString` method), 820  
`addtoindex()` (`pynlpl.formats.folia.TextMarkupStyle` method), 830  
`addtoindex()` (`pynlpl.formats.folia.TimeSegment` method), 635  
`addtoindex()` (`pynlpl.formats.folia.TimingLayer` method), 753  
`addtoindex()` (`pynlpl.formats.folia.Whitespace` method), 400  
`addtoindex()` (`pynlpl.formats.folia.Word` method), 414  
`alias()` (`pynlpl.formats.folia.Document` method), 17  
`align()` (`pynlpl.formats.taggerdata.Taggerdata` method), 1055  
`Alignment` (class in `pynlpl.formats.folia`), 1006  
`AlignReference` (class in `pynlpl.formats.folia`), 1018  
`AllowTokenAnnotation` (class in `pynlpl.formats.folia`), 47  
`Alternative` (class in `pynlpl.formats.folia`), 916  
`AlternativeLayers` (class in `pynlpl.formats.folia`), 929  
`alternatives()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 75  
`alternatives()` (`pynlpl.formats.folia.AbstractStructureElement` method), 38  
`alternatives()` (`pynlpl.formats.folia.AllowTokenAnnotation` method), 48  
`alternatives()` (`pynlpl.formats.folia.Alternative` method), 920  
`alternatives()` (`pynlpl.formats.folia.Cell` method), 101  
`alternatives()` (`pynlpl.formats.folia.ChunkingLayer` method), 647  
`alternatives()` (`pynlpl.formats.folia.CoreferenceLayer` method), 659  
`alternatives()` (`pynlpl.formats.folia.Definition` method), 114  
`alternatives()` (`pynlpl.formats.folia.DependenciesLayer` method), 671  
`alternatives()` (`pynlpl.formats.folia.Division` method), 127  
`alternatives()` (`pynlpl.formats.folia.EntitiesLayer` method), 683  
`alternatives()` (`pynlpl.formats.folia.Entry` method), 140  
`alternatives()` (`pynlpl.formats.folia.Event` method), 153  
`alternatives()` (`pynlpl.formats.folia.Example` method), 166  
`alternatives()` (`pynlpl.formats.folia.Figure` method), 179  
`alternatives()` (`pynlpl.formats.folia.Head` method), 203  
`alternatives()` (`pynlpl.formats.folia.Linebreak` method), 216  
`alternatives()` (`pynlpl.formats.folia.List` method), 229  
`alternatives()` (`pynlpl.formats.folia.ListItem` method), 242  
`alternatives()` (`pynlpl.formats.folia.Note` method), 255  
`alternatives()` (`pynlpl.formats.folia.ObservationLayer` method), 694  
`alternatives()` (`pynlpl.formats.folia.Paragraph` method), 268  
`alternatives()` (`pynlpl.formats.folia.Part` method), 281  
`alternatives()` (`pynlpl.formats.folia.Quote` method), 294  
`alternatives()` (`pynlpl.formats.folia.Reference` method), 307  
`alternatives()` (`pynlpl.formats.folia.Row` method), 320  
`alternatives()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 741  
`alternatives()` (`pynlpl.formats.folia.Sentence` method), 334  
`alternatives()` (`pynlpl.formats.folia.SentimentLayer` method), 706  
`alternatives()` (`pynlpl.formats.folia.StatementLayer` method), 718  
`alternatives()` (`pynlpl.formats.folia.SyntaxLayer` method), 730  
`alternatives()` (`pynlpl.formats.folia.Table` method), 348  
`alternatives()` (`pynlpl.formats.folia.TableHead` method), 374  
`alternatives()` (`pynlpl.formats.folia.Term` method), 361  
`alternatives()` (`pynlpl.formats.folia.Text` method), 387  
`alternatives()` (`pynlpl.formats.folia.TimingLayer` method), 753  
`alternatives()` (`pynlpl.formats.folia.Whitespace` method),

- 400
- alternatives() (pynlpl.formats.folia.Word method), 414
- ancestor() (pynlpl.formats.folia.AbstractAnnotationLayer method), 76
- ancestor() (pynlpl.formats.folia.AbstractElement method), 27
- ancestor() (pynlpl.formats.folia.AbstractSpanAnnotation method), 53
- ancestor() (pynlpl.formats.folia.AbstractStructureElement method), 38
- ancestor() (pynlpl.formats.folia.AbstractTextMarkup method), 87
- ancestor() (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- ancestor() (pynlpl.formats.folia.ActorFeature method), 886
- ancestor() (pynlpl.formats.folia.Alignment method), 1010
- ancestor() (pynlpl.formats.folia.AlignReference method), 1021
- ancestor() (pynlpl.formats.folia.Alternative method), 920
- ancestor() (pynlpl.formats.folia.AlternativeLayers method), 932
- ancestor() (pynlpl.formats.folia.BegindatetimeFeature method), 897
- ancestor() (pynlpl.formats.folia.Cell method), 101
- ancestor() (pynlpl.formats.folia.Chunk method), 519
- ancestor() (pynlpl.formats.folia.ChunkingLayer method), 647
- ancestor() (pynlpl.formats.folia.CoreferenceChain method), 530
- ancestor() (pynlpl.formats.folia.CoreferenceLayer method), 659
- ancestor() (pynlpl.formats.folia.CoreferenceLink method), 765
- ancestor() (pynlpl.formats.folia.Correction method), 945
- ancestor() (pynlpl.formats.folia.Current method), 954
- ancestor() (pynlpl.formats.folia.Definition method), 114
- ancestor() (pynlpl.formats.folia.DependenciesLayer method), 671
- ancestor() (pynlpl.formats.folia.Dependency method), 542
- ancestor() (pynlpl.formats.folia.DependencyDependent method), 777
- ancestor() (pynlpl.formats.folia.Description method), 1032
- ancestor() (pynlpl.formats.folia.Division method), 127
- ancestor() (pynlpl.formats.folia.DomainAnnotation method), 429
- ancestor() (pynlpl.formats.folia.EnddatetimeFeature method), 908
- ancestor() (pynlpl.formats.folia.EntitiesLayer method), 683
- ancestor() (pynlpl.formats.folia.Entity method), 554
- ancestor() (pynlpl.formats.folia.Entry method), 140
- ancestor() (pynlpl.formats.folia.ErrorDetection method), 965
- ancestor() (pynlpl.formats.folia.Event method), 153
- ancestor() (pynlpl.formats.folia.Example method), 166
- ancestor() (pynlpl.formats.folia.Feature method), 863
- ancestor() (pynlpl.formats.folia.Figure method), 179
- ancestor() (pynlpl.formats.folia.Gap method), 192
- ancestor() (pynlpl.formats.folia.Head method), 203
- ancestor() (pynlpl.formats.folia.Headspace method), 789
- ancestor() (pynlpl.formats.folia.LangAnnotation method), 451
- ancestor() (pynlpl.formats.folia.LemmaAnnotation method), 462
- ancestor() (pynlpl.formats.folia.Linebreak method), 216
- ancestor() (pynlpl.formats.folia.List method), 229
- ancestor() (pynlpl.formats.folia.ListItem method), 242
- ancestor() (pynlpl.formats.folia.Metric method), 1043
- ancestor() (pynlpl.formats.folia.New method), 976
- ancestor() (pynlpl.formats.folia.Note method), 255
- ancestor() (pynlpl.formats.folia.Observation method), 565
- ancestor() (pynlpl.formats.folia.ObservationLayer method), 695
- ancestor() (pynlpl.formats.folia.Original method), 987
- ancestor() (pynlpl.formats.folia.Paragraph method), 268
- ancestor() (pynlpl.formats.folia.Part method), 281
- ancestor() (pynlpl.formats.folia.PhonContent method), 507
- ancestor() (pynlpl.formats.folia.PosAnnotation method), 440
- ancestor() (pynlpl.formats.folia.Predicate method), 577
- ancestor() (pynlpl.formats.folia.Quote method), 294
- ancestor() (pynlpl.formats.folia.Reference method), 307
- ancestor() (pynlpl.formats.folia.Row method), 320
- ancestor() (pynlpl.formats.folia.SemanticRole method), 624
- ancestor() (pynlpl.formats.folia.SemanticRolesLayer method), 742
- ancestor() (pynlpl.formats.folia.SenseAnnotation method), 473
- ancestor() (pynlpl.formats.folia.Sentence method), 334
- ancestor() (pynlpl.formats.folia.Sentiment method), 589
- ancestor() (pynlpl.formats.folia.SentimentLayer method), 706
- ancestor() (pynlpl.formats.folia.Statement method), 600
- ancestor() (pynlpl.formats.folia.StatementLayer method), 718
- ancestor() (pynlpl.formats.folia.SubjectivityAnnotation method), 484
- ancestor() (pynlpl.formats.folia.Suggestion method), 998
- ancestor() (pynlpl.formats.folia.SynsetFeature method), 875
- ancestor() (pynlpl.formats.folia.SyntacticUnit method), 612

`ancestor()` (`pynlpl.formats.folia.SyntaxLayer` method), 730

`ancestor()` (`pynlpl.formats.folia.Table` method), 348

`ancestor()` (`pynlpl.formats.folia.TableHead` method), 374

`ancestor()` (`pynlpl.formats.folia.Term` method), 361

`ancestor()` (`pynlpl.formats.folia.Text` method), 387

`ancestor()` (`pynlpl.formats.folia.TextContent` method), 496

`ancestor()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 841

`ancestor()` (`pynlpl.formats.folia.TextMarkupError` method), 852

`ancestor()` (`pynlpl.formats.folia.TextMarkupGap` method), 809

`ancestor()` (`pynlpl.formats.folia.TextMarkupString` method), 820

`ancestor()` (`pynlpl.formats.folia.TextMarkupStyle` method), 831

`ancestor()` (`pynlpl.formats.folia.TimeSegment` method), 635

`ancestor()` (`pynlpl.formats.folia.TimingLayer` method), 753

`ancestor()` (`pynlpl.formats.folia.Whitespace` method), 400

`ancestor()` (`pynlpl.formats.folia.Word` method), 414

`ancestors()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 76

`ancestors()` (`pynlpl.formats.folia.AbstractElement` method), 27

`ancestors()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 53

`ancestors()` (`pynlpl.formats.folia.AbstractStructureElement` method), 38

`ancestors()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 87

`ancestors()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 64

`ancestors()` (`pynlpl.formats.folia.ActorFeature` method), 886

`ancestors()` (`pynlpl.formats.folia.Alignment` method), 1010

`ancestors()` (`pynlpl.formats.folia.AlignReference` method), 1021

`ancestors()` (`pynlpl.formats.folia.Alternative` method), 920

`ancestors()` (`pynlpl.formats.folia.AlternativeLayers` method), 932

`ancestors()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 897

`ancestors()` (`pynlpl.formats.folia.Cell` method), 102

`ancestors()` (`pynlpl.formats.folia.Chunk` method), 519

`ancestors()` (`pynlpl.formats.folia.ChunkingLayer` method), 648

`ancestors()` (`pynlpl.formats.folia.CoreferenceChain` method), 530

`ancestors()` (`pynlpl.formats.folia.CoreferenceLayer` method), 659

`ancestors()` (`pynlpl.formats.folia.CoreferenceLink` method), 765

`ancestors()` (`pynlpl.formats.folia.Correction` method), 945

`ancestors()` (`pynlpl.formats.folia.Current` method), 954

`ancestors()` (`pynlpl.formats.folia.Definition` method), 115

`ancestors()` (`pynlpl.formats.folia.DependenciesLayer` method), 671

`ancestors()` (`pynlpl.formats.folia.Dependency` method), 542

`ancestors()` (`pynlpl.formats.folia.DependencyDependent` method), 777

`ancestors()` (`pynlpl.formats.folia.Description` method), 1032

`ancestors()` (`pynlpl.formats.folia.Division` method), 128

`ancestors()` (`pynlpl.formats.folia.DomainAnnotation` method), 430

`ancestors()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 908

`ancestors()` (`pynlpl.formats.folia.EntitiesLayer` method), 683

`ancestors()` (`pynlpl.formats.folia.Entity` method), 554

`ancestors()` (`pynlpl.formats.folia.Entry` method), 140

`ancestors()` (`pynlpl.formats.folia.ErrorDetection` method), 966

`ancestors()` (`pynlpl.formats.folia.Event` method), 154

`ancestors()` (`pynlpl.formats.folia.Example` method), 166

`ancestors()` (`pynlpl.formats.folia.Feature` method), 864

`ancestors()` (`pynlpl.formats.folia.Figure` method), 180

`ancestors()` (`pynlpl.formats.folia.Gap` method), 192

`ancestors()` (`pynlpl.formats.folia.Head` method), 204

`ancestors()` (`pynlpl.formats.folia.Headspan` method), 789

`ancestors()` (`pynlpl.formats.folia.LangAnnotation` method), 452

`ancestors()` (`pynlpl.formats.folia.LemmaAnnotation` method), 463

`ancestors()` (`pynlpl.formats.folia.Linebreak` method), 217

`ancestors()` (`pynlpl.formats.folia.List` method), 229

`ancestors()` (`pynlpl.formats.folia.ListItem` method), 242

`ancestors()` (`pynlpl.formats.folia.Metric` method), 1044

`ancestors()` (`pynlpl.formats.folia.New` method), 977

`ancestors()` (`pynlpl.formats.folia.Note` method), 255

`ancestors()` (`pynlpl.formats.folia.Observation` method), 565

`ancestors()` (`pynlpl.formats.folia.ObservationLayer` method), 695

`ancestors()` (`pynlpl.formats.folia.Original` method), 988

`ancestors()` (`pynlpl.formats.folia.Paragraph` method), 269

`ancestors()` (`pynlpl.formats.folia.Part` method), 282

`ancestors()` (`pynlpl.formats.folia.PhonContent` method), 507

`ancestors()` (`pynlpl.formats.folia.PosAnnotation` method), 441

- `ancestors()` (`pynlpl.formats.folia.Predicate` method), 577
- `ancestors()` (`pynlpl.formats.folia.Quote` method), 294
- `ancestors()` (`pynlpl.formats.folia.Reference` method), 308
- `ancestors()` (`pynlpl.formats.folia.Row` method), 320
- `ancestors()` (`pynlpl.formats.folia.SemanticRole` method), 624
- `ancestors()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 742
- `ancestors()` (`pynlpl.formats.folia.SenseAnnotation` method), 474
- `ancestors()` (`pynlpl.formats.folia.Sentence` method), 334
- `ancestors()` (`pynlpl.formats.folia.Sentiment` method), 589
- `ancestors()` (`pynlpl.formats.folia.SentimentLayer` method), 707
- `ancestors()` (`pynlpl.formats.folia.Statement` method), 600
- `ancestors()` (`pynlpl.formats.folia.StatementLayer` method), 718
- `ancestors()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 485
- `ancestors()` (`pynlpl.formats.folia.Suggestion` method), 999
- `ancestors()` (`pynlpl.formats.folia.SynsetFeature` method), 875
- `ancestors()` (`pynlpl.formats.folia.SyntacticUnit` method), 612
- `ancestors()` (`pynlpl.formats.folia.SyntaxLayer` method), 730
- `ancestors()` (`pynlpl.formats.folia.Table` method), 348
- `ancestors()` (`pynlpl.formats.folia.TableHead` method), 374
- `ancestors()` (`pynlpl.formats.folia.Term` method), 361
- `ancestors()` (`pynlpl.formats.folia.Text` method), 387
- `ancestors()` (`pynlpl.formats.folia.TextContent` method), 496
- `ancestors()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 841
- `ancestors()` (`pynlpl.formats.folia.TextMarkupError` method), 852
- `ancestors()` (`pynlpl.formats.folia.TextMarkupGap` method), 809
- `ancestors()` (`pynlpl.formats.folia.TextMarkupString` method), 820
- `ancestors()` (`pynlpl.formats.folia.TextMarkupStyle` method), 831
- `ancestors()` (`pynlpl.formats.folia.TimeSegment` method), 635
- `ancestors()` (`pynlpl.formats.folia.TimingLayer` method), 754
- `ancestors()` (`pynlpl.formats.folia.Whitespace` method), 400
- `ancestors()` (`pynlpl.formats.folia.Word` method), 414
- `annotation()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 76
- `annotation()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 53
- `annotation()` (`pynlpl.formats.folia.AbstractStructureElement` method), 39
- `annotation()` (`pynlpl.formats.folia.AllowTokenAnnotation` method), 48
- `annotation()` (`pynlpl.formats.folia.Alternative` method), 920
- `annotation()` (`pynlpl.formats.folia.Cell` method), 102
- `annotation()` (`pynlpl.formats.folia.Chunk` method), 519
- `annotation()` (`pynlpl.formats.folia.ChunkingLayer` method), 648
- `annotation()` (`pynlpl.formats.folia.CoreferenceChain` method), 531
- `annotation()` (`pynlpl.formats.folia.CoreferenceLayer` method), 660
- `annotation()` (`pynlpl.formats.folia.CoreferenceLink` method), 765
- `annotation()` (`pynlpl.formats.folia.Definition` method), 115
- `annotation()` (`pynlpl.formats.folia.DependenciesLayer` method), 671
- `annotation()` (`pynlpl.formats.folia.Dependency` method), 542
- `annotation()` (`pynlpl.formats.folia.DependencyDependent` method), 777
- `annotation()` (`pynlpl.formats.folia.Division` method), 128
- `annotation()` (`pynlpl.formats.folia.EntitiesLayer` method), 683
- `annotation()` (`pynlpl.formats.folia.Entity` method), 554
- `annotation()` (`pynlpl.formats.folia.Entry` method), 141
- `annotation()` (`pynlpl.formats.folia.Event` method), 154
- `annotation()` (`pynlpl.formats.folia.Example` method), 167
- `annotation()` (`pynlpl.formats.folia.Figure` method), 180
- `annotation()` (`pynlpl.formats.folia.Head` method), 204
- `annotation()` (`pynlpl.formats.folia.Headspan` method), 789
- `annotation()` (`pynlpl.formats.folia.Linebreak` method), 217
- `annotation()` (`pynlpl.formats.folia.List` method), 230
- `annotation()` (`pynlpl.formats.folia.ListItem` method), 243
- `annotation()` (`pynlpl.formats.folia.Note` method), 256
- `annotation()` (`pynlpl.formats.folia.Observation` method), 566
- `annotation()` (`pynlpl.formats.folia.ObservationLayer` method), 695
- `annotation()` (`pynlpl.formats.folia.Paragraph` method), 269
- `annotation()` (`pynlpl.formats.folia.Part` method), 282
- `annotation()` (`pynlpl.formats.folia.Predicate` method), 577
- `annotation()` (`pynlpl.formats.folia.Quote` method), 295
- `annotation()` (`pynlpl.formats.folia.Reference` method), 308
- `annotation()` (`pynlpl.formats.folia.Row` method), 321
- `annotation()` (`pynlpl.formats.folia.SemanticRole` method), 624



`annotation()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 742

`annotation()` (`pynlpl.formats.folia.Sentence` method), 335

`annotation()` (`pynlpl.formats.folia.Sentiment` method), 589

`annotation()` (`pynlpl.formats.folia.SentimentLayer` method), 707

`annotation()` (`pynlpl.formats.folia.Statement` method), 600

`annotation()` (`pynlpl.formats.folia.StatementLayer` method), 718

`annotation()` (`pynlpl.formats.folia.SyntacticUnit` method), 612

`annotation()` (`pynlpl.formats.folia.SyntaxLayer` method), 730

`annotation()` (`pynlpl.formats.folia.Table` method), 348

`annotation()` (`pynlpl.formats.folia.TableHead` method), 374

`annotation()` (`pynlpl.formats.folia.Term` method), 361

`annotation()` (`pynlpl.formats.folia.Text` method), 387

`annotation()` (`pynlpl.formats.folia.TimeSegment` method), 635

`annotation()` (`pynlpl.formats.folia.TimingLayer` method), 754

`annotation()` (`pynlpl.formats.folia.Whitespace` method), 400

`annotation()` (`pynlpl.formats.folia.Word` method), 415

`annotations()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 76

`annotations()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 53

`annotations()` (`pynlpl.formats.folia.AbstractStructureElement` method), 39

`annotations()` (`pynlpl.formats.folia.AllowTokenAnnotation` method), 49

`annotations()` (`pynlpl.formats.folia.Alternative` method), 921

`annotations()` (`pynlpl.formats.folia.Cell` method), 102

`annotations()` (`pynlpl.formats.folia.Chunk` method), 519

`annotations()` (`pynlpl.formats.folia.ChunkingLayer` method), 648

`annotations()` (`pynlpl.formats.folia.CoreferenceChain` method), 531

`annotations()` (`pynlpl.formats.folia.CoreferenceLayer` method), 660

`annotations()` (`pynlpl.formats.folia.CoreferenceLink` method), 765

`annotations()` (`pynlpl.formats.folia.Definition` method), 115

`annotations()` (`pynlpl.formats.folia.DependenciesLayer` method), 671

`annotations()` (`pynlpl.formats.folia.Dependency` method), 542

`annotations()` (`pynlpl.formats.folia.DependencyDependent` method), 777

`annotations()` (`pynlpl.formats.folia.Division` method), 128

`annotations()` (`pynlpl.formats.folia.EntitiesLayer` method), 683

`annotations()` (`pynlpl.formats.folia.Entity` method), 554

`annotations()` (`pynlpl.formats.folia.Entry` method), 141

`annotations()` (`pynlpl.formats.folia.Event` method), 154

`annotations()` (`pynlpl.formats.folia.Example` method), 167

`annotations()` (`pynlpl.formats.folia.Figure` method), 180

`annotations()` (`pynlpl.formats.folia.Head` method), 204

`annotations()` (`pynlpl.formats.folia.Headspan` method), 789

`annotations()` (`pynlpl.formats.folia.Linebreak` method), 217

`annotations()` (`pynlpl.formats.folia.List` method), 230

`annotations()` (`pynlpl.formats.folia.ListItem` method), 243

`annotations()` (`pynlpl.formats.folia.Note` method), 256

`annotations()` (`pynlpl.formats.folia.Observation` method), 566

`annotations()` (`pynlpl.formats.folia.ObservationLayer` method), 695

`annotations()` (`pynlpl.formats.folia.Paragraph` method), 269

`annotations()` (`pynlpl.formats.folia.Part` method), 282

`annotations()` (`pynlpl.formats.folia.Predicate` method), 577

`annotations()` (`pynlpl.formats.folia.Quote` method), 295

`annotations()` (`pynlpl.formats.folia.Reference` method), 308

`annotations()` (`pynlpl.formats.folia.Row` method), 321

`annotations()` (`pynlpl.formats.folia.SemanticRole` method), 624

`annotations()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 742

`annotations()` (`pynlpl.formats.folia.Sentence` method), 335

`annotations()` (`pynlpl.formats.folia.Sentiment` method), 589

`annotations()` (`pynlpl.formats.folia.SentimentLayer` method), 707

`annotations()` (`pynlpl.formats.folia.Statement` method), 601

`annotations()` (`pynlpl.formats.folia.StatementLayer` method), 718

`annotations()` (`pynlpl.formats.folia.SyntacticUnit` method), 612

`annotations()` (`pynlpl.formats.folia.SyntaxLayer` method), 730

`annotations()` (`pynlpl.formats.folia.Table` method), 349

`annotations()` (`pynlpl.formats.folia.TableHead` method), 375

`annotations()` (`pynlpl.formats.folia.Term` method), 362

`annotations()` (`pynlpl.formats.folia.Text` method), 388

annotations() (pynlpl.formats.folia.TimeSegment attribute), 776  
 annotations() (pynlpl.formats.folia.TimingLayer method), 636  
 annotations() (pynlpl.formats.folia.Whitespace method), 754  
 annotations() (pynlpl.formats.folia.Word method), 401  
 annotations() (pynlpl.formats.folia.Word method), 415  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractAnnotation attribute), 74  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractElement attribute), 26  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractStructuralAnnotation attribute), 37  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractTextMarkup attribute), 86  
 ANNOTATIONTYPE (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63  
 ANNOTATIONTYPE (pynlpl.formats.folia.ActorFeature attribute), 884  
 ANNOTATIONTYPE (pynlpl.formats.folia.Alignment attribute), 1009  
 ANNOTATIONTYPE (pynlpl.formats.folia.AlignReference attribute), 1020  
 ANNOTATIONTYPE (pynlpl.formats.folia.Alternative attribute), 919  
 ANNOTATIONTYPE (pynlpl.formats.folia.AlternativeLayer attribute), 931  
 ANNOTATIONTYPE (pynlpl.formats.folia.BegindatetimeFeature attribute), 896  
 ANNOTATIONTYPE (pynlpl.formats.folia.Cell attribute), 100  
 ANNOTATIONTYPE (pynlpl.formats.folia.Chunk attribute), 518  
 ANNOTATIONTYPE (pynlpl.formats.folia.ChunkingLayer attribute), 646  
 ANNOTATIONTYPE (pynlpl.formats.folia.CoreferenceChain attribute), 529  
 ANNOTATIONTYPE (pynlpl.formats.folia.CoreferenceLayer attribute), 658  
 ANNOTATIONTYPE (pynlpl.formats.folia.CoreferenceLink attribute), 764  
 ANNOTATIONTYPE (pynlpl.formats.folia.Correction attribute), 944  
 ANNOTATIONTYPE (pynlpl.formats.folia.Current attribute), 953  
 ANNOTATIONTYPE (pynlpl.formats.folia.Definition attribute), 113  
 ANNOTATIONTYPE (pynlpl.formats.folia.DependenciesLayer attribute), 670  
 ANNOTATIONTYPE (pynlpl.formats.folia.Dependency attribute), 541  
 ANNOTATIONTYPE (pynlpl.formats.folia.DependencyDependency attribute), 776  
 ANNOTATIONTYPE (pynlpl.formats.folia.Description attribute), 1031  
 ANNOTATIONTYPE (pynlpl.formats.folia.Division attribute), 126  
 ANNOTATIONTYPE (pynlpl.formats.folia.DomainAnnotation attribute), 428  
 ANNOTATIONTYPE (pynlpl.formats.folia.EnddatetimeFeature attribute), 907  
 ANNOTATIONTYPE (pynlpl.formats.folia.EntitiesLayer attribute), 682  
 ANNOTATIONTYPE (pynlpl.formats.folia.Entity attribute), 553  
 ANNOTATIONTYPE (pynlpl.formats.folia.Entry attribute), 139  
 ANNOTATIONTYPE (pynlpl.formats.folia.ErrorDetection attribute), 964  
 ANNOTATIONTYPE (pynlpl.formats.folia.Event attribute), 152  
 ANNOTATIONTYPE (pynlpl.formats.folia.Example attribute), 165  
 ANNOTATIONTYPE (pynlpl.formats.folia.Feature attribute), 862  
 ANNOTATIONTYPE (pynlpl.formats.folia.Figure attribute), 178  
 ANNOTATIONTYPE (pynlpl.formats.folia.Gap attribute), 191  
 ANNOTATIONTYPE (pynlpl.formats.folia.Head attribute), 202  
 ANNOTATIONTYPE (pynlpl.formats.folia.Headspace attribute), 788  
 ANNOTATIONTYPE (pynlpl.formats.folia.LangAnnotation attribute), 450  
 ANNOTATIONTYPE (pynlpl.formats.folia.LemmaAnnotation attribute), 461  
 ANNOTATIONTYPE (pynlpl.formats.folia.Linebreak attribute), 215  
 ANNOTATIONTYPE (pynlpl.formats.folia.List attribute), 228  
 ANNOTATIONTYPE (pynlpl.formats.folia.ListItem attribute), 241  
 ANNOTATIONTYPE (pynlpl.formats.folia.Metric attribute), 1042  
 ANNOTATIONTYPE (pynlpl.formats.folia.New attribute), 975  
 ANNOTATIONTYPE (pynlpl.formats.folia.Note attribute), 254  
 ANNOTATIONTYPE (pynlpl.formats.folia.Observation attribute), 564  
 ANNOTATIONTYPE (pynlpl.formats.folia.ObservationLayer attribute), 693  
 ANNOTATIONTYPE (pynlpl.formats.folia.Original attribute), 986  
 ANNOTATIONTYPE (pynlpl.formats.folia.Paragraph attribute), 1060

- tribute), 267
- ANNOTATIONTYPE (pynlpl.formats.folia.Part attribute), 280
- ANNOTATIONTYPE (pynlpl.formats.folia.PhonContent attribute), 505
- ANNOTATIONTYPE (pynlpl.formats.folia.PosAnnotation attribute), 439
- ANNOTATIONTYPE (pynlpl.formats.folia.Predicate attribute), 576
- ANNOTATIONTYPE (pynlpl.formats.folia.Quote attribute), 293
- ANNOTATIONTYPE (pynlpl.formats.folia.Reference attribute), 306
- ANNOTATIONTYPE (pynlpl.formats.folia.Row attribute), 319
- ANNOTATIONTYPE (pynlpl.formats.folia.SemanticRole attribute), 622
- ANNOTATIONTYPE (pynlpl.formats.folia.SemanticRolesLayer attribute), 740
- ANNOTATIONTYPE (pynlpl.formats.folia.SenseAnnotation attribute), 472
- ANNOTATIONTYPE (pynlpl.formats.folia.Sentence attribute), 332
- ANNOTATIONTYPE (pynlpl.formats.folia.Sentiment attribute), 587
- ANNOTATIONTYPE (pynlpl.formats.folia.SentimentLayer attribute), 705
- ANNOTATIONTYPE (pynlpl.formats.folia.Statement attribute), 599
- ANNOTATIONTYPE (pynlpl.formats.folia.StatementLayer attribute), 717
- ANNOTATIONTYPE (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- ANNOTATIONTYPE (pynlpl.formats.folia.Suggestion attribute), 997
- ANNOTATIONTYPE (pynlpl.formats.folia.SynsetFeature attribute), 873
- ANNOTATIONTYPE (pynlpl.formats.folia.SyntacticUnit attribute), 611
- ANNOTATIONTYPE (pynlpl.formats.folia.SyntaxLayer attribute), 729
- ANNOTATIONTYPE (pynlpl.formats.folia.Table attribute), 347
- ANNOTATIONTYPE (pynlpl.formats.folia.TableHead attribute), 373
- ANNOTATIONTYPE (pynlpl.formats.folia.Term attribute), 360
- ANNOTATIONTYPE (pynlpl.formats.folia.Text attribute), 386
- ANNOTATIONTYPE (pynlpl.formats.folia.TextContent attribute), 495
- ANNOTATIONTYPE (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- ANNOTATIONTYPE (pynlpl.formats.folia.TextMarkupError attribute), 851
- ANNOTATIONTYPE (pynlpl.formats.folia.TextMarkupGap attribute), 808
- ANNOTATIONTYPE (pynlpl.formats.folia.TextMarkupString attribute), 819
- ANNOTATIONTYPE (pynlpl.formats.folia.TextMarkupStyle attribute), 829
- ANNOTATIONTYPE (pynlpl.formats.folia.TimeSegment attribute), 634
- ANNOTATIONTYPE (pynlpl.formats.folia.TimingLayer attribute), 752
- ANNOTATIONTYPE (pynlpl.formats.folia.Whitespace attribute), 399
- ANNOTATIONTYPE (pynlpl.formats.folia.Word attribute), 412
- append() (pynlpl.datatypes.FIFOQueue method), 5
- append() (pynlpl.datatypes.PriorityQueue method), 6
- append() (pynlpl.datatypes.Tree method), 6
- append() (pynlpl.datatypes.Trie method), 6
- append() (pynlpl.evaluation.ClassEvaluation method), 9
- append() (pynlpl.evaluation.ExperimentPool method), 10
- append() (pynlpl.formats.folia.AbstractAnnotationLayer method), 76
- append() (pynlpl.formats.folia.AbstractElement method), 27
- append() (pynlpl.formats.folia.AbstractSpanAnnotation method), 53
- append() (pynlpl.formats.folia.AbstractStructureElement method), 39
- append() (pynlpl.formats.folia.AbstractTextMarkup method), 87
- append() (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- append() (pynlpl.formats.folia.ActorFeature method), 886
- append() (pynlpl.formats.folia.Alignment method), 1010
- append() (pynlpl.formats.folia.AlignReference method), 1021
- append() (pynlpl.formats.folia.Alternative method), 921
- append() (pynlpl.formats.folia.AlternativeLayers method), 932
- append() (pynlpl.formats.folia.BegindatetimeFeature method), 897
- append() (pynlpl.formats.folia.Cell method), 103
- append() (pynlpl.formats.folia.Chunk method), 519
- append() (pynlpl.formats.folia.ChunkingLayer method), 648
- append() (pynlpl.formats.folia.CoreferenceChain method), 531
- append() (pynlpl.formats.folia.CoreferenceLayer method), 660
- append() (pynlpl.formats.folia.CoreferenceLink method), 766
- append() (pynlpl.formats.folia.Correction method), 945



- append() (pynlpl.formats.folia.Current method), 955
- append() (pynlpl.formats.folia.Definition method), 116
- append() (pynlpl.formats.folia.DependenciesLayer method), 672
- append() (pynlpl.formats.folia.Dependency method), 543
- append() (pynlpl.formats.folia.DependencyDependent method), 777
- append() (pynlpl.formats.folia.Description method), 1033
- append() (pynlpl.formats.folia.Division method), 129
- append() (pynlpl.formats.folia.Document method), 17
- append() (pynlpl.formats.folia.DomainAnnotation method), 430
- append() (pynlpl.formats.folia.EnddatetimeFeature method), 908
- append() (pynlpl.formats.folia.EntitiesLayer method), 683
- append() (pynlpl.formats.folia.Entity method), 554
- append() (pynlpl.formats.folia.Entry method), 141
- append() (pynlpl.formats.folia.ErrorDetection method), 966
- append() (pynlpl.formats.folia.Event method), 155
- append() (pynlpl.formats.folia.Example method), 167
- append() (pynlpl.formats.folia.Feature method), 864
- append() (pynlpl.formats.folia.Figure method), 181
- append() (pynlpl.formats.folia.Gap method), 192
- append() (pynlpl.formats.folia.Head method), 205
- append() (pynlpl.formats.folia.Headspan method), 789
- append() (pynlpl.formats.folia.LangAnnotation method), 452
- append() (pynlpl.formats.folia.LemmaAnnotation method), 463
- append() (pynlpl.formats.folia.Linebreak method), 218
- append() (pynlpl.formats.folia.List method), 230
- append() (pynlpl.formats.folia.ListItem method), 243
- append() (pynlpl.formats.folia.Metric method), 1044
- append() (pynlpl.formats.folia.New method), 977
- append() (pynlpl.formats.folia.Note method), 256
- append() (pynlpl.formats.folia.Observation method), 566
- append() (pynlpl.formats.folia.ObservationLayer method), 695
- append() (pynlpl.formats.folia.Original method), 988
- append() (pynlpl.formats.folia.Paragraph method), 270
- append() (pynlpl.formats.folia.Part method), 283
- append() (pynlpl.formats.folia.PhonContent method), 507
- append() (pynlpl.formats.folia.PosAnnotation method), 441
- append() (pynlpl.formats.folia.Predicate method), 577
- append() (pynlpl.formats.folia.Quote method), 295
- append() (pynlpl.formats.folia.Reference method), 309
- append() (pynlpl.formats.folia.Row method), 321
- append() (pynlpl.formats.folia.SemanticRole method), 624
- append() (pynlpl.formats.folia.SemanticRolesLayer method), 742
- append() (pynlpl.formats.folia.SenseAnnotation method), 474
- append() (pynlpl.formats.folia.Sentence method), 335
- append() (pynlpl.formats.folia.Sentiment method), 589
- append() (pynlpl.formats.folia.SentimentLayer method), 707
- append() (pynlpl.formats.folia.Statement method), 601
- append() (pynlpl.formats.folia.StatementLayer method), 719
- append() (pynlpl.formats.folia.SubjectivityAnnotation method), 485
- append() (pynlpl.formats.folia.Suggestion method), 999
- append() (pynlpl.formats.folia.SynsetFeature method), 875
- append() (pynlpl.formats.folia.SyntacticUnit method), 612
- append() (pynlpl.formats.folia.SyntaxLayer method), 730
- append() (pynlpl.formats.folia.Table method), 349
- append() (pynlpl.formats.folia.TableHead method), 375
- append() (pynlpl.formats.folia.Term method), 362
- append() (pynlpl.formats.folia.Text method), 388
- append() (pynlpl.formats.folia.TextContent method), 496
- append() (pynlpl.formats.folia.TextMarkupCorrection method), 842
- append() (pynlpl.formats.folia.TextMarkupError method), 852
- append() (pynlpl.formats.folia.TextMarkupGap method), 810
- append() (pynlpl.formats.folia.TextMarkupString method), 820
- append() (pynlpl.formats.folia.TextMarkupStyle method), 831
- append() (pynlpl.formats.folia.TimeSegment method), 636
- append() (pynlpl.formats.folia.TimingLayer method), 754
- append() (pynlpl.formats.folia.Whitespace method), 401
- append() (pynlpl.formats.folia.Word method), 415
- append() (pynlpl.lm.lm.SimpleLanguageModel method), 1057
- append() (pynlpl.statistics.FrequencyList method), 1063
- ARPALanguageModel (class in pynlpl.lm.lm), 1057
- ARPALanguageModel.NgramsProbs (class in pynlpl.lm.lm), 1057
- auc() (in module pynlpl.evaluation), 10
- auc() (pynlpl.evaluation.ClassEvaluation method), 9
- AUTH (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74
- AUTH (pynlpl.formats.folia.AbstractElement attribute), 26
- AUTH (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52
- AUTH (pynlpl.formats.folia.AbstractStructureElement attribute), 37
- AUTH (pynlpl.formats.folia.AbstractTextMarkup at-

- tribute), 86
- AUTH (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63
- AUTH (pynlpl.formats.folia.ActorFeature attribute), 885
- AUTH (pynlpl.formats.folia.Alignment attribute), 1009
- AUTH (pynlpl.formats.folia.AlignReference attribute), 1020
- AUTH (pynlpl.formats.folia.Alternative attribute), 919
- AUTH (pynlpl.formats.folia.AlternativeLayers attribute), 931
- AUTH (pynlpl.formats.folia.BegindatetimeFeature attribute), 896
- AUTH (pynlpl.formats.folia.Cell attribute), 100
- AUTH (pynlpl.formats.folia.Chunk attribute), 518
- AUTH (pynlpl.formats.folia.ChunkingLayer attribute), 646
- AUTH (pynlpl.formats.folia.CoreferenceChain attribute), 529
- AUTH (pynlpl.formats.folia.CoreferenceLayer attribute), 658
- AUTH (pynlpl.formats.folia.CoreferenceLink attribute), 764
- AUTH (pynlpl.formats.folia.Correction attribute), 944
- AUTH (pynlpl.formats.folia.Current attribute), 953
- AUTH (pynlpl.formats.folia.Definition attribute), 113
- AUTH (pynlpl.formats.folia.DependenciesLayer attribute), 670
- AUTH (pynlpl.formats.folia.Dependency attribute), 541
- AUTH (pynlpl.formats.folia.DependencyDependent attribute), 776
- AUTH (pynlpl.formats.folia.Description attribute), 1031
- AUTH (pynlpl.formats.folia.Division attribute), 126
- AUTH (pynlpl.formats.folia.DomainAnnotation attribute), 428
- AUTH (pynlpl.formats.folia.EnddatetimeFeature attribute), 907
- AUTH (pynlpl.formats.folia.EntitiesLayer attribute), 682
- AUTH (pynlpl.formats.folia.Entity attribute), 553
- AUTH (pynlpl.formats.folia.Entry attribute), 139
- AUTH (pynlpl.formats.folia.ErrorDetection attribute), 964
- AUTH (pynlpl.formats.folia.Event attribute), 152
- AUTH (pynlpl.formats.folia.Example attribute), 165
- AUTH (pynlpl.formats.folia.Feature attribute), 862
- AUTH (pynlpl.formats.folia.Figure attribute), 178
- AUTH (pynlpl.formats.folia.Gap attribute), 191
- AUTH (pynlpl.formats.folia.Head attribute), 202
- AUTH (pynlpl.formats.folia.Headspan attribute), 788
- AUTH (pynlpl.formats.folia.LangAnnotation attribute), 450
- AUTH (pynlpl.formats.folia.LemmaAnnotation attribute), 461
- AUTH (pynlpl.formats.folia.Linebreak attribute), 215
- AUTH (pynlpl.formats.folia.List attribute), 228
- AUTH (pynlpl.formats.folia.ListItem attribute), 241
- AUTH (pynlpl.formats.folia.Metric attribute), 1042
- AUTH (pynlpl.formats.folia.New attribute), 975
- AUTH (pynlpl.formats.folia.Note attribute), 254
- AUTH (pynlpl.formats.folia.Observation attribute), 564
- AUTH (pynlpl.formats.folia.ObservationLayer attribute), 693
- AUTH (pynlpl.formats.folia.Original attribute), 986
- AUTH (pynlpl.formats.folia.Paragraph attribute), 267
- AUTH (pynlpl.formats.folia.Part attribute), 280
- AUTH (pynlpl.formats.folia.PhonContent attribute), 505
- AUTH (pynlpl.formats.folia.PosAnnotation attribute), 439
- AUTH (pynlpl.formats.folia.Predicate attribute), 576
- AUTH (pynlpl.formats.folia.Quote attribute), 293
- AUTH (pynlpl.formats.folia.Reference attribute), 306
- AUTH (pynlpl.formats.folia.Row attribute), 319
- AUTH (pynlpl.formats.folia.SemanticRole attribute), 622
- AUTH (pynlpl.formats.folia.SemanticRolesLayer attribute), 740
- AUTH (pynlpl.formats.folia.SenseAnnotation attribute), 472
- AUTH (pynlpl.formats.folia.Sentence attribute), 332
- AUTH (pynlpl.formats.folia.Sentiment attribute), 587
- AUTH (pynlpl.formats.folia.SentimentLayer attribute), 705
- AUTH (pynlpl.formats.folia.Statement attribute), 599
- AUTH (pynlpl.formats.folia.StatementLayer attribute), 717
- AUTH (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- AUTH (pynlpl.formats.folia.Suggestion attribute), 997
- AUTH (pynlpl.formats.folia.SynsetFeature attribute), 873
- AUTH (pynlpl.formats.folia.SyntacticUnit attribute), 611
- AUTH (pynlpl.formats.folia.SyntaxLayer attribute), 729
- AUTH (pynlpl.formats.folia.Table attribute), 347
- AUTH (pynlpl.formats.folia.TableHead attribute), 373
- AUTH (pynlpl.formats.folia.Term attribute), 360
- AUTH (pynlpl.formats.folia.Text attribute), 386
- AUTH (pynlpl.formats.folia.TextContent attribute), 495
- AUTH (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- AUTH (pynlpl.formats.folia.TextMarkupError attribute), 851
- AUTH (pynlpl.formats.folia.TextMarkupGap attribute), 808
- AUTH (pynlpl.formats.folia.TextMarkupString attribute), 819
- AUTH (pynlpl.formats.folia.TextMarkupStyle attribute), 829
- AUTH (pynlpl.formats.folia.TimeSegment attribute), 634
- AUTH (pynlpl.formats.folia.TimingLayer attribute), 752
- AUTH (pynlpl.formats.folia.Whitespace attribute), 399
- AUTH (pynlpl.formats.folia.Word attribute), 412

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractAnnotation attribute), 74

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractElement attribute), 26

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractSpan attribute), 52

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractStructure attribute), 37

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractText attribute), 86

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AbstractToken attribute), 63

AUTO\_GENERATE\_ID (pynlpl.formats.folia.ActorFeature attribute), 885

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Alignment attribute), 1009

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AlignReference attribute), 1020

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Alternative attribute), 919

AUTO\_GENERATE\_ID (pynlpl.formats.folia.AlternativeLayer attribute), 931

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Begindatetime attribute), 896

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Cell attribute), 100

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Chunk attribute), 518

AUTO\_GENERATE\_ID (pynlpl.formats.folia.ChunkingLayer attribute), 646

AUTO\_GENERATE\_ID (pynlpl.formats.folia.CoreferenceChain attribute), 529

AUTO\_GENERATE\_ID (pynlpl.formats.folia.CoreferenceLayer attribute), 658

AUTO\_GENERATE\_ID (pynlpl.formats.folia.CoreferenceLink attribute), 764

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Correction attribute), 944

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Current attribute), 953

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Definition attribute), 113

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Dependencies attribute), 670

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Dependency attribute), 541

AUTO\_GENERATE\_ID (pynlpl.formats.folia.DependencyDependency attribute), 776

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Description attribute), 1031

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Division attribute), 126

AUTO\_GENERATE\_ID (pynlpl.formats.folia.DomainAnnotation attribute), 428

AUTO\_GENERATE\_ID (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

AUTO\_GENERATE\_ID (pynlpl.formats.folia.EntitiesLayer attribute), 682

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Entity attribute), 553

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Entry attribute), 139

AUTO\_GENERATE\_ID (pynlpl.formats.folia.ErrorDetection attribute), 964

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Event attribute), 152

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Example attribute), 165

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Feature attribute), 862

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Figure attribute), 178

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Gap attribute), 191

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Head attribute), 202

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Headspan attribute), 788

AUTO\_GENERATE\_ID (pynlpl.formats.folia.LangAnnotation attribute), 450

AUTO\_GENERATE\_ID (pynlpl.formats.folia.LemmaAnnotation attribute), 461

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Linebreak attribute), 215

AUTO\_GENERATE\_ID (pynlpl.formats.folia.List attribute), 228

AUTO\_GENERATE\_ID (pynlpl.formats.folia.ListItem attribute), 241

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Metric attribute), 1042

AUTO\_GENERATE\_ID (pynlpl.formats.folia.New attribute), 975

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Note attribute), 254

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Observation attribute), 564

AUTO\_GENERATE\_ID (pynlpl.formats.folia.ObservationLayer attribute), 693

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Original attribute), 986

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Paragraph attribute), 267

AUTO\_GENERATE\_ID (pynlpl.formats.folia.Part attribute), 280

AUTO\_GENERATE\_ID (pynlpl.formats.folia.PhonContent attribute), 505

AUTO\_GENERATE\_ID (pynlpl.formats.folia.PosAnnotation attribute), 439

- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Predicate attribute), 576
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Quote attribute), 293
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Reference attribute), 306
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Row attribute), 319
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SemanticRole attribute), 623
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SemanticRolesLayer attribute), 740
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SenseAnnotation attribute), 472
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Sentence attribute), 332
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Sentiment attribute), 588
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SentimentLayer attribute), 705
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Statement attribute), 599
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.StatementLayer attribute), 717
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Suggestion attribute), 997
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SynsetFeature attribute), 873
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SyntacticUnit attribute), 611
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.SyntaxLayer attribute), 729
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Table attribute), 347
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TableHead attribute), 373
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Term attribute), 360
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Text attribute), 386
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextContent attribute), 495
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextMarkupError attribute), 851
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextMarkupGap attribute), 808
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextMarkupString attribute), 819
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TimeSegment attribute), 634
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.TimingLayer attribute), 752
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Whitespace attribute), 399
- AUTO\_GENERATE\_ID (pynlpl.formats.folia.Word attribute), 412
- ## B
- backoff() (in module pynlpl.common), 3
- backoff() (pynlpl.lm.lm.ARPALanguageModel.NgramsProbs method), 1057
- BeamedBestFirstSearch (class in pynlpl.search), 1060
- BeamSearch (class in pynlpl.search), 1060
- BeginatetimeFeature (class in pynlpl.formats.folia), 893
- BestFirstSearch (class in pynlpl.search), 1060
- binary\_search() (in module pynlpl.search), 1060
- BreadthFirstSearch (class in pynlpl.search), 1060
- ## C
- calculate\_overlap() (in module pynlpl.textprocessors), 1066
- caption() (pynlpl.formats.folia.Figure method), 181
- Cell (class in pynlpl.formats.folia), 98
- Chunk (class in pynlpl.formats.folia), 515
- ChunkingLayer (class in pynlpl.formats.folia), 644
- ClassEvaluation (class in pynlpl.evaluation), 9
- close() (pynlpl.formats.taggedata.Taggedata method), 1055
- communicates() (pynlpl.statistics.MarkovChain method), 1063
- compute() (pynlpl.evaluation.ClassEvaluation method), 9
- compute() (pynlpl.evaluation.OrdinalEvaluation method), 10
- ConfusionMatrix (class in pynlpl.evaluation), 10
- confusionmatrix() (pynlpl.evaluation.ClassEvaluation method), 9
- content() (pynlpl.formats.folia.Gap method), 192
- context() (pynlpl.formats.folia.AbstractAnnotationLayer method), 76
- context() (pynlpl.formats.folia.AbstractElement method), 27
- context() (pynlpl.formats.folia.AbstractSpanAnnotation method), 53
- context() (pynlpl.formats.folia.AbstractStructureElement method), 40
- context() (pynlpl.formats.folia.AbstractTextMarkup method), 87
- context() (pynlpl.formats.folia.AbstractTokenAnnotation method), 64
- context() (pynlpl.formats.folia.ActorFeature method), 886
- context() (pynlpl.formats.folia.Alignment method), 1010

- context() (pynlpl.formats.folia.AlignReference method), 1021
- context() (pynlpl.formats.folia.Alternative method), 921
- context() (pynlpl.formats.folia.AlternativeLayers method), 932
- context() (pynlpl.formats.folia.BegindatetimeFeature method), 897
- context() (pynlpl.formats.folia.Cell method), 103
- context() (pynlpl.formats.folia.Chunk method), 519
- context() (pynlpl.formats.folia.ChunkingLayer method), 648
- context() (pynlpl.formats.folia.CoreferenceChain method), 531
- context() (pynlpl.formats.folia.CoreferenceLayer method), 660
- context() (pynlpl.formats.folia.CoreferenceLink method), 766
- context() (pynlpl.formats.folia.Correction method), 945
- context() (pynlpl.formats.folia.Current method), 955
- context() (pynlpl.formats.folia.Definition method), 116
- context() (pynlpl.formats.folia.DependenciesLayer method), 672
- context() (pynlpl.formats.folia.Dependency method), 543
- context() (pynlpl.formats.folia.DependencyDependent method), 777
- context() (pynlpl.formats.folia.Description method), 1033
- context() (pynlpl.formats.folia.Division method), 129
- context() (pynlpl.formats.folia.DomainAnnotation method), 430
- context() (pynlpl.formats.folia.EnddatetimeFeature method), 908
- context() (pynlpl.formats.folia.EntitiesLayer method), 684
- context() (pynlpl.formats.folia.Entity method), 554
- context() (pynlpl.formats.folia.Entry method), 142
- context() (pynlpl.formats.folia.ErrorDetection method), 966
- context() (pynlpl.formats.folia.Event method), 155
- context() (pynlpl.formats.folia.Example method), 168
- context() (pynlpl.formats.folia.Feature method), 864
- context() (pynlpl.formats.folia.Figure method), 181
- context() (pynlpl.formats.folia.Gap method), 192
- context() (pynlpl.formats.folia.Head method), 205
- context() (pynlpl.formats.folia.Headspeed method), 789
- context() (pynlpl.formats.folia.LangAnnotation method), 452
- context() (pynlpl.formats.folia.LemmaAnnotation method), 463
- context() (pynlpl.formats.folia.Linebreak method), 218
- context() (pynlpl.formats.folia.List method), 231
- context() (pynlpl.formats.folia.ListItem method), 244
- context() (pynlpl.formats.folia.Metric method), 1044
- context() (pynlpl.formats.folia.New method), 977
- context() (pynlpl.formats.folia.Note method), 257
- context() (pynlpl.formats.folia.Observation method), 566
- context() (pynlpl.formats.folia.ObservationLayer method), 695
- context() (pynlpl.formats.folia.Original method), 988
- context() (pynlpl.formats.folia.Paragraph method), 270
- context() (pynlpl.formats.folia.Part method), 283
- context() (pynlpl.formats.folia.PhonContent method), 507
- context() (pynlpl.formats.folia.PosAnnotation method), 441
- context() (pynlpl.formats.folia.Predicate method), 577
- context() (pynlpl.formats.folia.Quote method), 296
- context() (pynlpl.formats.folia.Reference method), 309
- context() (pynlpl.formats.folia.Row method), 322
- context() (pynlpl.formats.folia.SemanticRole method), 624
- context() (pynlpl.formats.folia.SemanticRolesLayer method), 742
- context() (pynlpl.formats.folia.SenseAnnotation method), 474
- context() (pynlpl.formats.folia.Sentence method), 336
- context() (pynlpl.formats.folia.Sentiment method), 589
- context() (pynlpl.formats.folia.SentimentLayer method), 707
- context() (pynlpl.formats.folia.Statement method), 601
- context() (pynlpl.formats.folia.StatementLayer method), 719
- context() (pynlpl.formats.folia.SubjectivityAnnotation method), 485
- context() (pynlpl.formats.folia.Suggestion method), 999
- context() (pynlpl.formats.folia.SynsetFeature method), 875
- context() (pynlpl.formats.folia.SyntacticUnit method), 612
- context() (pynlpl.formats.folia.SyntaxLayer method), 731
- context() (pynlpl.formats.folia.Table method), 349
- context() (pynlpl.formats.folia.TableHead method), 375
- context() (pynlpl.formats.folia.Term method), 362
- context() (pynlpl.formats.folia.Text method), 388
- context() (pynlpl.formats.folia.TextContent method), 496
- context() (pynlpl.formats.folia.TextMarkupCorrection method), 842
- context() (pynlpl.formats.folia.TextMarkupError method), 852
- context() (pynlpl.formats.folia.TextMarkupGap method), 810
- context() (pynlpl.formats.folia.TextMarkupString method), 820
- context() (pynlpl.formats.folia.TextMarkupStyle method), 831
- context() (pynlpl.formats.folia.TimeSegment method), 636
- context() (pynlpl.formats.folia.TimingLayer method), 754



`context()` (`pynlpl.formats.folia.Whitespace` method), 401  
`context()` (`pynlpl.formats.folia.Word` method), 416  
`copy()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 76  
`copy()` (`pynlpl.formats.folia.AbstractElement` method), 27  
`copy()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 53  
`copy()` (`pynlpl.formats.folia.AbstractStructureElement` method), 40  
`copy()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 87  
`copy()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 65  
`copy()` (`pynlpl.formats.folia.ActorFeature` method), 886  
`copy()` (`pynlpl.formats.folia.Alignment` method), 1010  
`copy()` (`pynlpl.formats.folia.AlignReference` method), 1021  
`copy()` (`pynlpl.formats.folia.Alternative` method), 921  
`copy()` (`pynlpl.formats.folia.AlternativeLayers` method), 932  
`copy()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 897  
`copy()` (`pynlpl.formats.folia.Cell` method), 103  
`copy()` (`pynlpl.formats.folia.Chunk` method), 519  
`copy()` (`pynlpl.formats.folia.ChunkingLayer` method), 648  
`copy()` (`pynlpl.formats.folia.CoreferenceChain` method), 531  
`copy()` (`pynlpl.formats.folia.CoreferenceLayer` method), 660  
`copy()` (`pynlpl.formats.folia.CoreferenceLink` method), 766  
`copy()` (`pynlpl.formats.folia.Correction` method), 945  
`copy()` (`pynlpl.formats.folia.Current` method), 955  
`copy()` (`pynlpl.formats.folia.Definition` method), 116  
`copy()` (`pynlpl.formats.folia.DependenciesLayer` method), 672  
`copy()` (`pynlpl.formats.folia.Dependency` method), 543  
`copy()` (`pynlpl.formats.folia.DependencyDependent` method), 778  
`copy()` (`pynlpl.formats.folia.Description` method), 1033  
`copy()` (`pynlpl.formats.folia.Division` method), 129  
`copy()` (`pynlpl.formats.folia.DomainAnnotation` method), 430  
`copy()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 908  
`copy()` (`pynlpl.formats.folia.EntitiesLayer` method), 684  
`copy()` (`pynlpl.formats.folia.Entity` method), 554  
`copy()` (`pynlpl.formats.folia.Entry` method), 142  
`copy()` (`pynlpl.formats.folia.ErrorDetection` method), 966  
`copy()` (`pynlpl.formats.folia.Event` method), 155  
`copy()` (`pynlpl.formats.folia.Example` method), 168  
`copy()` (`pynlpl.formats.folia.Feature` method), 864  
`copy()` (`pynlpl.formats.folia.Figure` method), 181  
`copy()` (`pynlpl.formats.folia.Gap` method), 192  
`copy()` (`pynlpl.formats.folia.Head` method), 205  
`copy()` (`pynlpl.formats.folia.Headspeed` method), 789  
`copy()` (`pynlpl.formats.folia.LangAnnotation` method), 452  
`copy()` (`pynlpl.formats.folia.LemmaAnnotation` method), 463  
`copy()` (`pynlpl.formats.folia.Linebreak` method), 218  
`copy()` (`pynlpl.formats.folia.List` method), 231  
`copy()` (`pynlpl.formats.folia.ListItem` method), 244  
`copy()` (`pynlpl.formats.folia.Metric` method), 1044  
`copy()` (`pynlpl.formats.folia.New` method), 977  
`copy()` (`pynlpl.formats.folia.Note` method), 257  
`copy()` (`pynlpl.formats.folia.Observation` method), 566  
`copy()` (`pynlpl.formats.folia.ObservationLayer` method), 695  
`copy()` (`pynlpl.formats.folia.Original` method), 988  
`copy()` (`pynlpl.formats.folia.Paragraph` method), 270  
`copy()` (`pynlpl.formats.folia.Part` method), 283  
`copy()` (`pynlpl.formats.folia.PhonContent` method), 507  
`copy()` (`pynlpl.formats.folia.PosAnnotation` method), 441  
`copy()` (`pynlpl.formats.folia.Predicate` method), 578  
`copy()` (`pynlpl.formats.folia.Quote` method), 296  
`copy()` (`pynlpl.formats.folia.Reference` method), 309  
`copy()` (`pynlpl.formats.folia.Row` method), 322  
`copy()` (`pynlpl.formats.folia.SemanticRole` method), 624  
`copy()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 742  
`copy()` (`pynlpl.formats.folia.SenseAnnotation` method), 474  
`copy()` (`pynlpl.formats.folia.Sentence` method), 336  
`copy()` (`pynlpl.formats.folia.Sentiment` method), 589  
`copy()` (`pynlpl.formats.folia.SentimentLayer` method), 707  
`copy()` (`pynlpl.formats.folia.Statement` method), 601  
`copy()` (`pynlpl.formats.folia.StatementLayer` method), 719  
`copy()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 485  
`copy()` (`pynlpl.formats.folia.Suggestion` method), 999  
`copy()` (`pynlpl.formats.folia.SynsetFeature` method), 875  
`copy()` (`pynlpl.formats.folia.SyntacticUnit` method), 613  
`copy()` (`pynlpl.formats.folia.SyntaxLayer` method), 731  
`copy()` (`pynlpl.formats.folia.Table` method), 349  
`copy()` (`pynlpl.formats.folia.TableHead` method), 375  
`copy()` (`pynlpl.formats.folia.Term` method), 362  
`copy()` (`pynlpl.formats.folia.Text` method), 388  
`copy()` (`pynlpl.formats.folia.TextContent` method), 496  
`copy()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 842  
`copy()` (`pynlpl.formats.folia.TextMarkupError` method), 852

- copy() (pynlpl.formats.folia.TextMarkupGap method), 810
- copy() (pynlpl.formats.folia.TextMarkupString method), 820
- copy() (pynlpl.formats.folia.TextMarkupStyle method), 831
- copy() (pynlpl.formats.folia.TimeSegment method), 636
- copy() (pynlpl.formats.folia.TimingLayer method), 754
- copy() (pynlpl.formats.folia.Whitespace method), 401
- copy() (pynlpl.formats.folia.Word method), 416
- copychildren() (pynlpl.formats.folia.AbstractAnnotationLayer method), 76
- copychildren() (pynlpl.formats.folia.AbstractElement method), 27
- copychildren() (pynlpl.formats.folia.AbstractSpanAnnotation method), 54
- copychildren() (pynlpl.formats.folia.AbstractStructureElement method), 40
- copychildren() (pynlpl.formats.folia.AbstractTextMarkup method), 88
- copychildren() (pynlpl.formats.folia.AbstractTokenAnnotation method), 65
- copychildren() (pynlpl.formats.folia.ActorFeature method), 886
- copychildren() (pynlpl.formats.folia.Alignment method), 1010
- copychildren() (pynlpl.formats.folia.AlignReference method), 1021
- copychildren() (pynlpl.formats.folia.Alternative method), 921
- copychildren() (pynlpl.formats.folia.AlternativeLayers method), 933
- copychildren() (pynlpl.formats.folia.BegindatetimeFeature method), 897
- copychildren() (pynlpl.formats.folia.Cell method), 103
- copychildren() (pynlpl.formats.folia.Chunk method), 520
- copychildren() (pynlpl.formats.folia.ChunkingLayer method), 648
- copychildren() (pynlpl.formats.folia.CoreferenceChain method), 531
- copychildren() (pynlpl.formats.folia.CoreferenceLayer method), 660
- copychildren() (pynlpl.formats.folia.CoreferenceLink method), 766
- copychildren() (pynlpl.formats.folia.Correction method), 945
- copychildren() (pynlpl.formats.folia.Current method), 955
- copychildren() (pynlpl.formats.folia.Definition method), 116
- copychildren() (pynlpl.formats.folia.DependenciesLayer method), 672
- copychildren() (pynlpl.formats.folia.Dependency method), 543
- copychildren() (pynlpl.formats.folia.DependencyDependent method), 778
- copychildren() (pynlpl.formats.folia.Description method), 1033
- copychildren() (pynlpl.formats.folia.Division method), 129
- copychildren() (pynlpl.formats.folia.DomainAnnotation method), 430
- copychildren() (pynlpl.formats.folia.EnddatetimeFeature method), 908
- copychildren() (pynlpl.formats.folia.EntitiesLayer method), 684
- copychildren() (pynlpl.formats.folia.Entity method), 554
- copychildren() (pynlpl.formats.folia.Entry method), 142
- copychildren() (pynlpl.formats.folia.ErrorDetection method), 966
- copychildren() (pynlpl.formats.folia.Event method), 155
- copychildren() (pynlpl.formats.folia.Example method), 168
- copychildren() (pynlpl.formats.folia.Feature method), 864
- copychildren() (pynlpl.formats.folia.Figure method), 181
- copychildren() (pynlpl.formats.folia.Gap method), 192
- copychildren() (pynlpl.formats.folia.Head method), 205
- copychildren() (pynlpl.formats.folia.Headspace method), 789
- copychildren() (pynlpl.formats.folia.LangAnnotation method), 452
- copychildren() (pynlpl.formats.folia.LemmaAnnotation method), 463
- copychildren() (pynlpl.formats.folia.Linebreak method), 218
- copychildren() (pynlpl.formats.folia.List method), 231
- copychildren() (pynlpl.formats.folia.ListItem method), 244
- copychildren() (pynlpl.formats.folia.Metric method), 1044
- copychildren() (pynlpl.formats.folia.New method), 977
- copychildren() (pynlpl.formats.folia.Note method), 257
- copychildren() (pynlpl.formats.folia.Observation method), 566
- copychildren() (pynlpl.formats.folia.ObservationLayer method), 696
- copychildren() (pynlpl.formats.folia.Original method), 988
- copychildren() (pynlpl.formats.folia.Paragraph method), 270
- copychildren() (pynlpl.formats.folia.Part method), 283
- copychildren() (pynlpl.formats.folia.PhonContent method), 507
- copychildren() (pynlpl.formats.folia.PosAnnotation method), 441
- copychildren() (pynlpl.formats.folia.Predicate method), 578

- [copychildren\(\) \(pynlpl.formats.folia.Quote method\), 296](#)
- [copychildren\(\) \(pynlpl.formats.folia.Reference method\), 309](#)
- [copychildren\(\) \(pynlpl.formats.folia.Row method\), 322](#)
- [copychildren\(\) \(pynlpl.formats.folia.SemanticRole method\), 624](#)
- [copychildren\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 743](#)
- [copychildren\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 474](#)
- [copychildren\(\) \(pynlpl.formats.folia.Sentence method\), 336](#)
- [copychildren\(\) \(pynlpl.formats.folia.Sentiment method\), 589](#)
- [copychildren\(\) \(pynlpl.formats.folia.SentimentLayer method\), 707](#)
- [copychildren\(\) \(pynlpl.formats.folia.Statement method\), 601](#)
- [copychildren\(\) \(pynlpl.formats.folia.StatementLayer method\), 719](#)
- [copychildren\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 485](#)
- [copychildren\(\) \(pynlpl.formats.folia.Suggestion method\), 999](#)
- [copychildren\(\) \(pynlpl.formats.folia.SynsetFeature method\), 875](#)
- [copychildren\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 613](#)
- [copychildren\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 731](#)
- [copychildren\(\) \(pynlpl.formats.folia.Table method\), 350](#)
- [copychildren\(\) \(pynlpl.formats.folia.TableHead method\), 376](#)
- [copychildren\(\) \(pynlpl.formats.folia.Term method\), 363](#)
- [copychildren\(\) \(pynlpl.formats.folia.Text method\), 389](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextContent method\), 497](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextMarkupCorrection method\), 842](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextMarkupError method\), 852](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextMarkupGap method\), 810](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextMarkupString method\), 820](#)
- [copychildren\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 831](#)
- [copychildren\(\) \(pynlpl.formats.folia.TimeSegment method\), 636](#)
- [copychildren\(\) \(pynlpl.formats.folia.TimingLayer method\), 754](#)
- [copychildren\(\) \(pynlpl.formats.folia.Whitespace method\), 402](#)
- [copychildren\(\) \(pynlpl.formats.folia.Word method\), 416](#)
- [CoreferenceChain \(class in pynlpl.formats.folia\), 527](#)
- [CoreferenceLayer \(class in pynlpl.formats.folia\), 656](#)
- [CoreferenceLink \(class in pynlpl.formats.folia\), 762](#)
- [Corpus \(class in pynlpl.formats.sonar\), 1054](#)
- [CorpusDocument \(class in pynlpl.formats.sonar\), 1054](#)
- [CorpusDocumentX \(class in pynlpl.formats.sonar\), 1054](#)
- [CorpusFiles \(class in pynlpl.formats.sonar\), 1055](#)
- [CorpusX \(class in pynlpl.formats.sonar\), 1055](#)
- [correct\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 77](#)
- [correct\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 54](#)
- [correct\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 40](#)
- [correct\(\) \(pynlpl.formats.folia.AllowTokenAnnotation method\), 49](#)
- [correct\(\) \(pynlpl.formats.folia.Alternative method\), 921](#)
- [correct\(\) \(pynlpl.formats.folia.Cell method\), 103](#)
- [correct\(\) \(pynlpl.formats.folia.Chunk method\), 520](#)
- [correct\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 648](#)
- [correct\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 531](#)
- [correct\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 660](#)
- [correct\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 766](#)
- [correct\(\) \(pynlpl.formats.folia.Correction method\), 945](#)
- [correct\(\) \(pynlpl.formats.folia.Current method\), 955](#)
- [correct\(\) \(pynlpl.formats.folia.Definition method\), 116](#)
- [correct\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 672](#)
- [correct\(\) \(pynlpl.formats.folia.Dependency method\), 543](#)
- [correct\(\) \(pynlpl.formats.folia.DependencyDependent method\), 778](#)
- [correct\(\) \(pynlpl.formats.folia.Division method\), 129](#)
- [correct\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 684](#)
- [correct\(\) \(pynlpl.formats.folia.Entity method\), 555](#)
- [correct\(\) \(pynlpl.formats.folia.Entry method\), 142](#)
- [correct\(\) \(pynlpl.formats.folia.Event method\), 155](#)
- [correct\(\) \(pynlpl.formats.folia.Example method\), 168](#)
- [correct\(\) \(pynlpl.formats.folia.Figure method\), 181](#)
- [correct\(\) \(pynlpl.formats.folia.Head method\), 205](#)
- [correct\(\) \(pynlpl.formats.folia.Headspan method\), 790](#)
- [correct\(\) \(pynlpl.formats.folia.Linebreak method\), 218](#)
- [correct\(\) \(pynlpl.formats.folia.List method\), 231](#)
- [correct\(\) \(pynlpl.formats.folia.ListItem method\), 244](#)
- [correct\(\) \(pynlpl.formats.folia.New method\), 977](#)
- [correct\(\) \(pynlpl.formats.folia.Note method\), 257](#)
- [correct\(\) \(pynlpl.formats.folia.Observation method\), 566](#)
- [correct\(\) \(pynlpl.formats.folia.ObservationLayer method\), 696](#)
- [correct\(\) \(pynlpl.formats.folia.Paragraph method\), 270](#)



- `correct()` (`pynlpl.formats.folia.Part` method), 283
- `correct()` (`pynlpl.formats.folia.Predicate` method), 578
- `correct()` (`pynlpl.formats.folia.Quote` method), 296
- `correct()` (`pynlpl.formats.folia.Reference` method), 309
- `correct()` (`pynlpl.formats.folia.Row` method), 322
- `correct()` (`pynlpl.formats.folia.SemanticRole` method), 625
- `correct()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 743
- `correct()` (`pynlpl.formats.folia.Sentence` method), 336
- `correct()` (`pynlpl.formats.folia.Sentiment` method), 590
- `correct()` (`pynlpl.formats.folia.SentimentLayer` method), 707
- `correct()` (`pynlpl.formats.folia.Statement` method), 601
- `correct()` (`pynlpl.formats.folia.StatementLayer` method), 719
- `correct()` (`pynlpl.formats.folia.SyntacticUnit` method), 613
- `correct()` (`pynlpl.formats.folia.SyntaxLayer` method), 731
- `correct()` (`pynlpl.formats.folia.Table` method), 350
- `correct()` (`pynlpl.formats.folia.TableHead` method), 376
- `correct()` (`pynlpl.formats.folia.Term` method), 363
- `correct()` (`pynlpl.formats.folia.Text` method), 389
- `correct()` (`pynlpl.formats.folia.TimeSegment` method), 636
- `correct()` (`pynlpl.formats.folia.TimingLayer` method), 754
- `correct()` (`pynlpl.formats.folia.Whitespace` method), 402
- `correct()` (`pynlpl.formats.folia.Word` method), 416
- `Correction` (class in `pynlpl.formats.folia`), 941
- `corrections()` (`pynlpl.formats.folia.Sentence` method), 336
- `correctwords()` (`pynlpl.formats.folia.Sentence` method), 336
- `count()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77
- `count()` (`pynlpl.formats.folia.AbstractElement` method), 27
- `count()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 54
- `count()` (`pynlpl.formats.folia.AbstractStructureElement` method), 40
- `count()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 88
- `count()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 65
- `count()` (`pynlpl.formats.folia.ActorFeature` method), 886
- `count()` (`pynlpl.formats.folia.Alignment` method), 1010
- `count()` (`pynlpl.formats.folia.AlignReference` method), 1022
- `count()` (`pynlpl.formats.folia.Alternative` method), 922
- `count()` (`pynlpl.formats.folia.AlternativeLayers` method), 933
- `count()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 897
- `count()` (`pynlpl.formats.folia.Cell` method), 103
- `count()` (`pynlpl.formats.folia.Chunk` method), 520
- `count()` (`pynlpl.formats.folia.ChunkingLayer` method), 648
- `count()` (`pynlpl.formats.folia.CoreferenceChain` method), 531
- `count()` (`pynlpl.formats.folia.CoreferenceLayer` method), 660
- `count()` (`pynlpl.formats.folia.CoreferenceLink` method), 766
- `count()` (`pynlpl.formats.folia.Correction` method), 945
- `count()` (`pynlpl.formats.folia.Current` method), 955
- `count()` (`pynlpl.formats.folia.Definition` method), 116
- `count()` (`pynlpl.formats.folia.DependenciesLayer` method), 672
- `count()` (`pynlpl.formats.folia.Dependency` method), 543
- `count()` (`pynlpl.formats.folia.DependencyDependent` method), 778
- `count()` (`pynlpl.formats.folia.Description` method), 1033
- `count()` (`pynlpl.formats.folia.Division` method), 129
- `count()` (`pynlpl.formats.folia.Document` method), 17
- `count()` (`pynlpl.formats.folia.DomainAnnotation` method), 430
- `count()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 908
- `count()` (`pynlpl.formats.folia.EntitiesLayer` method), 684
- `count()` (`pynlpl.formats.folia.Entity` method), 555
- `count()` (`pynlpl.formats.folia.Entry` method), 142
- `count()` (`pynlpl.formats.folia.ErrorDetection` method), 966
- `count()` (`pynlpl.formats.folia.Event` method), 155
- `count()` (`pynlpl.formats.folia.Example` method), 168
- `count()` (`pynlpl.formats.folia.Feature` method), 864
- `count()` (`pynlpl.formats.folia.Figure` method), 181
- `count()` (`pynlpl.formats.folia.Gap` method), 192
- `count()` (`pynlpl.formats.folia.Head` method), 205
- `count()` (`pynlpl.formats.folia.Headspan` method), 790
- `count()` (`pynlpl.formats.folia.LangAnnotation` method), 452
- `count()` (`pynlpl.formats.folia.LemmaAnnotation` method), 463
- `count()` (`pynlpl.formats.folia.Linebreak` method), 218
- `count()` (`pynlpl.formats.folia.List` method), 231
- `count()` (`pynlpl.formats.folia.ListItem` method), 244
- `count()` (`pynlpl.formats.folia.Metric` method), 1044
- `count()` (`pynlpl.formats.folia.New` method), 977
- `count()` (`pynlpl.formats.folia.Note` method), 257
- `count()` (`pynlpl.formats.folia.Observation` method), 566
- `count()` (`pynlpl.formats.folia.ObservationLayer` method), 696
- `count()` (`pynlpl.formats.folia.Original` method), 988
- `count()` (`pynlpl.formats.folia.Paragraph` method), 270
- `count()` (`pynlpl.formats.folia.Part` method), 283
- `count()` (`pynlpl.formats.folia.PhonContent` method), 507

`count()` (`pynlpl.formats.folia.PosAnnotation` method), 441  
`count()` (`pynlpl.formats.folia.Predicate` method), 578  
`count()` (`pynlpl.formats.folia.Quote` method), 296  
`count()` (`pynlpl.formats.folia.Reference` method), 309  
`count()` (`pynlpl.formats.folia.Row` method), 322  
`count()` (`pynlpl.formats.folia.SemanticRole` method), 625  
`count()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 743  
`count()` (`pynlpl.formats.folia.SenseAnnotation` method), 474  
`count()` (`pynlpl.formats.folia.Sentence` method), 336  
`count()` (`pynlpl.formats.folia.Sentiment` method), 590  
`count()` (`pynlpl.formats.folia.SentimentLayer` method), 707  
`count()` (`pynlpl.formats.folia.Statement` method), 601  
`count()` (`pynlpl.formats.folia.StatementLayer` method), 719  
`count()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 485  
`count()` (`pynlpl.formats.folia.Suggestion` method), 999  
`count()` (`pynlpl.formats.folia.SynsetFeature` method), 875  
`count()` (`pynlpl.formats.folia.SyntacticUnit` method), 613  
`count()` (`pynlpl.formats.folia.SyntaxLayer` method), 731  
`count()` (`pynlpl.formats.folia.Table` method), 350  
`count()` (`pynlpl.formats.folia.TableHead` method), 376  
`count()` (`pynlpl.formats.folia.Term` method), 363  
`count()` (`pynlpl.formats.folia.Text` method), 389  
`count()` (`pynlpl.formats.folia.TextContent` method), 497  
`count()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 842  
`count()` (`pynlpl.formats.folia.TextMarkupError` method), 853  
`count()` (`pynlpl.formats.folia.TextMarkupGap` method), 810  
`count()` (`pynlpl.formats.folia.TextMarkupString` method), 821  
`count()` (`pynlpl.formats.folia.TextMarkupStyle` method), 831  
`count()` (`pynlpl.formats.folia.TimeSegment` method), 636  
`count()` (`pynlpl.formats.folia.TimingLayer` method), 754  
`count()` (`pynlpl.formats.folia.Whitespace` method), 402  
`count()` (`pynlpl.formats.folia.Word` method), 416  
`count()` (`pynlpl.statistics.FrequencyList` method), 1063  
`create()` (`pynlpl.formats.folia.Document` method), 17  
`crude_tokenizer()` (in module `pynlpl.textprocessors`), 1066  
`Current` (class in `pynlpl.formats.folia`), 951  
`current()` (`pynlpl.formats.folia.Correction` method), 946  
`deepvalidation()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77  
`deepvalidation()` (`pynlpl.formats.folia.AbstractElement` method), 27  
`deepvalidation()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 54  
`deepvalidation()` (`pynlpl.formats.folia.AbstractStructureElement` method), 40  
`deepvalidation()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 88  
`deepvalidation()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 65  
`deepvalidation()` (`pynlpl.formats.folia.ActorFeature` method), 887  
`deepvalidation()` (`pynlpl.formats.folia.Alignment` method), 1010  
`deepvalidation()` (`pynlpl.formats.folia.AlignReference` method), 1022  
`deepvalidation()` (`pynlpl.formats.folia.Alternative` method), 922  
`deepvalidation()` (`pynlpl.formats.folia.AlternativeLayers` method), 933  
`deepvalidation()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 898  
`deepvalidation()` (`pynlpl.formats.folia.Cell` method), 103  
`deepvalidation()` (`pynlpl.formats.folia.Chunk` method), 520  
`deepvalidation()` (`pynlpl.formats.folia.ChunkingLayer` method), 649  
`deepvalidation()` (`pynlpl.formats.folia.CoreferenceChain` method), 531  
`deepvalidation()` (`pynlpl.formats.folia.CoreferenceLayer` method), 660  
`deepvalidation()` (`pynlpl.formats.folia.CoreferenceLink` method), 766  
`deepvalidation()` (`pynlpl.formats.folia.Correction` method), 946  
`deepvalidation()` (`pynlpl.formats.folia.Current` method), 955  
`deepvalidation()` (`pynlpl.formats.folia.Definition` method), 116  
`deepvalidation()` (`pynlpl.formats.folia.DependenciesLayer` method), 672  
`deepvalidation()` (`pynlpl.formats.folia.Dependency` method), 543  
`deepvalidation()` (`pynlpl.formats.folia.DependencyDependent` method), 778  
`deepvalidation()` (`pynlpl.formats.folia.Description` method), 1033  
`deepvalidation()` (`pynlpl.formats.folia.Division` method), 129  
`deepvalidation()` (`pynlpl.formats.folia.DomainAnnotation` method), 430  
`deepvalidation()` (`pynlpl.formats.folia.EnddatetimeFeature`

## D

`date()` (`pynlpl.formats.folia.Document` method), 17  
`declare()` (`pynlpl.formats.folia.Document` method), 17  
`declared()` (`pynlpl.formats.folia.Document` method), 18

- method), 909
- deepvalidation() (pynlpl.formats.folia.EntitiesLayer method), 684
- deepvalidation() (pynlpl.formats.folia.Entity method), 555
- deepvalidation() (pynlpl.formats.folia.Entry method), 142
- deepvalidation() (pynlpl.formats.folia.ErrorDetection method), 966
- deepvalidation() (pynlpl.formats.folia.Event method), 155
- deepvalidation() (pynlpl.formats.folia.Example method), 168
- deepvalidation() (pynlpl.formats.folia.Feature method), 864
- deepvalidation() (pynlpl.formats.folia.Figure method), 181
- deepvalidation() (pynlpl.formats.folia.Gap method), 193
- deepvalidation() (pynlpl.formats.folia.Head method), 205
- deepvalidation() (pynlpl.formats.folia.Headspace method), 790
- deepvalidation() (pynlpl.formats.folia.LangAnnotation method), 452
- deepvalidation() (pynlpl.formats.folia.LemmaAnnotation method), 463
- deepvalidation() (pynlpl.formats.folia.Linebreak method), 218
- deepvalidation() (pynlpl.formats.folia.List method), 231
- deepvalidation() (pynlpl.formats.folia.ListItem method), 244
- deepvalidation() (pynlpl.formats.folia.Metric method), 1044
- deepvalidation() (pynlpl.formats.folia.New method), 977
- deepvalidation() (pynlpl.formats.folia.Note method), 257
- deepvalidation() (pynlpl.formats.folia.Observation method), 566
- deepvalidation() (pynlpl.formats.folia.ObservationLayer method), 696
- deepvalidation() (pynlpl.formats.folia.Original method), 988
- deepvalidation() (pynlpl.formats.folia.Paragraph method), 270
- deepvalidation() (pynlpl.formats.folia.Part method), 283
- deepvalidation() (pynlpl.formats.folia.PhonContent method), 507
- deepvalidation() (pynlpl.formats.folia.PosAnnotation method), 441
- deepvalidation() (pynlpl.formats.folia.Predicate method), 578
- deepvalidation() (pynlpl.formats.folia.Quote method), 296
- deepvalidation() (pynlpl.formats.folia.Reference method), 309
- deepvalidation() (pynlpl.formats.folia.Row method), 322
- deepvalidation() (pynlpl.formats.folia.SemanticRole method), 625
- deepvalidation() (pynlpl.formats.folia.SemanticRolesLayer method), 743
- deepvalidation() (pynlpl.formats.folia.SenseAnnotation method), 474
- deepvalidation() (pynlpl.formats.folia.Sentence method), 336
- deepvalidation() (pynlpl.formats.folia.Sentiment method), 590
- deepvalidation() (pynlpl.formats.folia.SentimentLayer method), 707
- deepvalidation() (pynlpl.formats.folia.Statement method), 601
- deepvalidation() (pynlpl.formats.folia.StatementLayer method), 719
- deepvalidation() (pynlpl.formats.folia.SubjectivityAnnotation method), 485
- deepvalidation() (pynlpl.formats.folia.Suggestion method), 999
- deepvalidation() (pynlpl.formats.folia.SynsetFeature method), 875
- deepvalidation() (pynlpl.formats.folia.SyntacticUnit method), 613
- deepvalidation() (pynlpl.formats.folia.SyntaxLayer method), 731
- deepvalidation() (pynlpl.formats.folia.Table method), 350
- deepvalidation() (pynlpl.formats.folia.TableHead method), 376
- deepvalidation() (pynlpl.formats.folia.Term method), 363
- deepvalidation() (pynlpl.formats.folia.Text method), 389
- deepvalidation() (pynlpl.formats.folia.TextContent method), 497
- deepvalidation() (pynlpl.formats.folia.TextMarkupCorrection method), 842
- deepvalidation() (pynlpl.formats.folia.TextMarkupError method), 853
- deepvalidation() (pynlpl.formats.folia.TextMarkupGap method), 810
- deepvalidation() (pynlpl.formats.folia.TextMarkupString method), 821
- deepvalidation() (pynlpl.formats.folia.TextMarkupStyle method), 831
- deepvalidation() (pynlpl.formats.folia.TimeSegment method), 636
- deepvalidation() (pynlpl.formats.folia.TimingLayer method), 754
- deepvalidation() (pynlpl.formats.folia.Whitespace method), 402
- deepvalidation() (pynlpl.formats.folia.Word method), 416
- defaultannotator() (pynlpl.formats.folia.Document method), 18
- defaultannotatortype() (pynlpl.formats.folia.Document method), 18
- defaultdatetime() (pynlpl.formats.folia.Document

- method), 19
- defaultparameters() (pynlpl.evaluation.AbstractExperiment method), 9
- defaultset() (pynlpl.formats.folia.Document method), 19
- Definition (class in pynlpl.formats.folia), 111
- delete() (pynlpl.evaluation.AbstractExperiment method), 9
- deleteword() (pynlpl.formats.folia.Sentence method), 336
- DependenciesLayer (class in pynlpl.formats.folia), 667
- Dependency (class in pynlpl.formats.folia), 538
- DependencyDependent (class in pynlpl.formats.folia), 773
- dependent() (pynlpl.formats.folia.Dependency method), 543
- depth() (pynlpl.datatypes.Trie method), 6
- depth() (pynlpl.search.AbstractSearchState method), 1059
- DepthFirstSearch (class in pynlpl.search), 1060
- Description (class in pynlpl.formats.folia), 1029
- description() (pynlpl.formats.folia.AbstractAnnotationLayer method), 77
- description() (pynlpl.formats.folia.AbstractElement method), 27
- description() (pynlpl.formats.folia.AbstractSpanAnnotation method), 54
- description() (pynlpl.formats.folia.AbstractStructureElement method), 40
- description() (pynlpl.formats.folia.AbstractTextMarkup method), 88
- description() (pynlpl.formats.folia.AbstractTokenAnnotation method), 65
- description() (pynlpl.formats.folia.ActorFeature method), 887
- description() (pynlpl.formats.folia.Alignment method), 1011
- description() (pynlpl.formats.folia.AlignReference method), 1022
- description() (pynlpl.formats.folia.Alternative method), 922
- description() (pynlpl.formats.folia.AlternativeLayers method), 933
- description() (pynlpl.formats.folia.BegindatetimeFeature method), 898
- description() (pynlpl.formats.folia.Cell method), 103
- description() (pynlpl.formats.folia.Chunk method), 520
- description() (pynlpl.formats.folia.ChunkingLayer method), 649
- description() (pynlpl.formats.folia.CoreferenceChain method), 531
- description() (pynlpl.formats.folia.CoreferenceLayer method), 660
- description() (pynlpl.formats.folia.CoreferenceLink method), 766
- description() (pynlpl.formats.folia.Correction method), 946
- description() (pynlpl.formats.folia.Current method), 955
- description() (pynlpl.formats.folia.Definition method), 116
- description() (pynlpl.formats.folia.DependenciesLayer method), 672
- description() (pynlpl.formats.folia.Dependency method), 543
- description() (pynlpl.formats.folia.DependencyDependent method), 778
- description() (pynlpl.formats.folia.Description method), 1033
- description() (pynlpl.formats.folia.Division method), 129
- description() (pynlpl.formats.folia.DomainAnnotation method), 430
- description() (pynlpl.formats.folia.EnddatetimeFeature method), 909
- description() (pynlpl.formats.folia.EntitiesLayer method), 684
- description() (pynlpl.formats.folia.Entity method), 555
- description() (pynlpl.formats.folia.Entry method), 142
- description() (pynlpl.formats.folia.ErrorDetection method), 966
- description() (pynlpl.formats.folia.Event method), 155
- description() (pynlpl.formats.folia.Example method), 168
- description() (pynlpl.formats.folia.Feature method), 864
- description() (pynlpl.formats.folia.Figure method), 181
- description() (pynlpl.formats.folia.Gap method), 193
- description() (pynlpl.formats.folia.Head method), 205
- description() (pynlpl.formats.folia.Headspan method), 790
- description() (pynlpl.formats.folia.LangAnnotation method), 452
- description() (pynlpl.formats.folia.LemmaAnnotation method), 463
- description() (pynlpl.formats.folia.Linebreak method), 218
- description() (pynlpl.formats.folia.List method), 231
- description() (pynlpl.formats.folia.ListItem method), 244
- description() (pynlpl.formats.folia.Metric method), 1044
- description() (pynlpl.formats.folia.New method), 977
- description() (pynlpl.formats.folia.Note method), 257
- description() (pynlpl.formats.folia.Observation method), 566
- description() (pynlpl.formats.folia.ObservationLayer method), 696
- description() (pynlpl.formats.folia.Original method), 988
- description() (pynlpl.formats.folia.Paragraph method), 270
- description() (pynlpl.formats.folia.Part method), 283
- description() (pynlpl.formats.folia.PhonContent method), 507
- description() (pynlpl.formats.folia.PosAnnotation method), 441

- `description()` (`pynlpl.formats.folia.Predicate` method), 578
  - `description()` (`pynlpl.formats.folia.Quote` method), 296
  - `description()` (`pynlpl.formats.folia.Reference` method), 309
  - `description()` (`pynlpl.formats.folia.Row` method), 322
  - `description()` (`pynlpl.formats.folia.SemanticRole` method), 625
  - `description()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 743
  - `description()` (`pynlpl.formats.folia.SenseAnnotation` method), 474
  - `description()` (`pynlpl.formats.folia.Sentence` method), 336
  - `description()` (`pynlpl.formats.folia.Sentiment` method), 590
  - `description()` (`pynlpl.formats.folia.SentimentLayer` method), 707
  - `description()` (`pynlpl.formats.folia.Statement` method), 601
  - `description()` (`pynlpl.formats.folia.StatementLayer` method), 719
  - `description()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 485
  - `description()` (`pynlpl.formats.folia.Suggestion` method), 999
  - `description()` (`pynlpl.formats.folia.SynsetFeature` method), 875
  - `description()` (`pynlpl.formats.folia.SyntacticUnit` method), 613
  - `description()` (`pynlpl.formats.folia.SyntaxLayer` method), 731
  - `description()` (`pynlpl.formats.folia.Table` method), 350
  - `description()` (`pynlpl.formats.folia.TableHead` method), 376
  - `description()` (`pynlpl.formats.folia.Term` method), 363
  - `description()` (`pynlpl.formats.folia.Text` method), 389
  - `description()` (`pynlpl.formats.folia.TextContent` method), 497
  - `description()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 842
  - `description()` (`pynlpl.formats.folia.TextMarkupError` method), 853
  - `description()` (`pynlpl.formats.folia.TextMarkupGap` method), 810
  - `description()` (`pynlpl.formats.folia.TextMarkupString` method), 821
  - `description()` (`pynlpl.formats.folia.TextMarkupStyle` method), 831
  - `description()` (`pynlpl.formats.folia.TimeSegment` method), 636
  - `description()` (`pynlpl.formats.folia.TimingLayer` method), 754
  - `description()` (`pynlpl.formats.folia.Whitespace` method), 402
  - `description()` (`pynlpl.formats.folia.Word` method), 416
  - `dict()` (`pynlpl.statistics.FrequencyList` method), 1063
  - `Distribution` (class in `pynlpl.statistics`), 1062
  - `Division` (class in `pynlpl.formats.folia`), 123
  - `division()` (`pynlpl.formats.folia.Sentence` method), 336
  - `division()` (`pynlpl.formats.folia.Word` method), 416
  - `Document` (class in `pynlpl.formats.folia`), 14
  - `domain()` (`pynlpl.formats.folia.Word` method), 416
  - `DomainAnnotation` (class in `pynlpl.formats.folia`), 426
  - `done()` (`pynlpl.evaluation.AbstractExperiment` method), 9
  - `dotproduct()` (in module `pynlpl.statistics`), 1064
  - `duration()` (`pynlpl.evaluation.AbstractExperiment` method), 9
- ## E
- `EarlyEagerBeamSearch` (class in `pynlpl.search`), 1060
  - `EnddatetimeFeature` (class in `pynlpl.formats.folia`), 904
  - `EntitiesLayer` (class in `pynlpl.formats.folia`), 679
  - `Entity` (class in `pynlpl.formats.folia`), 550
  - `entropy()` (`pynlpl.statistics.Distribution` method), 1062
  - `Entry` (class in `pynlpl.formats.folia`), 136
  - `Enum()` (in module `pynlpl.common`), 3
  - `ErrorDetection` (class in `pynlpl.formats.folia`), 962
  - `Event` (class in `pynlpl.formats.folia`), 149
  - `Example` (class in `pynlpl.formats.folia`), 162
  - `expand()` (`pynlpl.search.AbstractSearchState` method), 1060
  - `ExperimentPool` (class in `pynlpl.evaluation`), 10
  - `extend()` (`pynlpl.datatypes.FIFOQueue` method), 5
  - `extend()` (`pynlpl.datatypes.Queue` method), 6
- ## F
- `feat()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77
  - `feat()` (`pynlpl.formats.folia.AbstractElement` method), 28
  - `feat()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 54
  - `feat()` (`pynlpl.formats.folia.AbstractStructureElement` method), 40
  - `feat()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 88
  - `feat()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 65
  - `feat()` (`pynlpl.formats.folia.ActorFeature` method), 887
  - `feat()` (`pynlpl.formats.folia.Alignment` method), 1011
  - `feat()` (`pynlpl.formats.folia.AlignReference` method), 1022
  - `feat()` (`pynlpl.formats.folia.Alternative` method), 922
  - `feat()` (`pynlpl.formats.folia.AlternativeLayers` method), 933
  - `feat()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 898
  - `feat()` (`pynlpl.formats.folia.Cell` method), 103
  - `feat()` (`pynlpl.formats.folia.Chunk` method), 520



- feat() (pynlpl.formats.folia.ChunkingLayer method), 649
- feat() (pynlpl.formats.folia.CoreferenceChain method), 531
- feat() (pynlpl.formats.folia.CoreferenceLayer method), 660
- feat() (pynlpl.formats.folia.CoreferenceLink method), 766
- feat() (pynlpl.formats.folia.Correction method), 946
- feat() (pynlpl.formats.folia.Current method), 955
- feat() (pynlpl.formats.folia.Definition method), 116
- feat() (pynlpl.formats.folia.DependenciesLayer method), 672
- feat() (pynlpl.formats.folia.Dependency method), 543
- feat() (pynlpl.formats.folia.DependencyDependent method), 778
- feat() (pynlpl.formats.folia.Description method), 1033
- feat() (pynlpl.formats.folia.Division method), 129
- feat() (pynlpl.formats.folia.DomainAnnotation method), 430
- feat() (pynlpl.formats.folia.EnddatetimeFeature method), 909
- feat() (pynlpl.formats.folia.EntitiesLayer method), 684
- feat() (pynlpl.formats.folia.Entity method), 555
- feat() (pynlpl.formats.folia.Entry method), 142
- feat() (pynlpl.formats.folia.ErrorDetection method), 966
- feat() (pynlpl.formats.folia.Event method), 155
- feat() (pynlpl.formats.folia.Example method), 168
- feat() (pynlpl.formats.folia.Feature method), 864
- feat() (pynlpl.formats.folia.Figure method), 181
- feat() (pynlpl.formats.folia.Gap method), 193
- feat() (pynlpl.formats.folia.Head method), 205
- feat() (pynlpl.formats.folia.Headspan method), 790
- feat() (pynlpl.formats.folia.LangAnnotation method), 452
- feat() (pynlpl.formats.folia.LemmaAnnotation method), 463
- feat() (pynlpl.formats.folia.Linebreak method), 218
- feat() (pynlpl.formats.folia.List method), 231
- feat() (pynlpl.formats.folia.ListItem method), 244
- feat() (pynlpl.formats.folia.Metric method), 1044
- feat() (pynlpl.formats.folia.New method), 977
- feat() (pynlpl.formats.folia.Note method), 257
- feat() (pynlpl.formats.folia.Observation method), 566
- feat() (pynlpl.formats.folia.ObservationLayer method), 696
- feat() (pynlpl.formats.folia.Original method), 988
- feat() (pynlpl.formats.folia.Paragraph method), 270
- feat() (pynlpl.formats.folia.Part method), 283
- feat() (pynlpl.formats.folia.PhonContent method), 507
- feat() (pynlpl.formats.folia.PosAnnotation method), 441
- feat() (pynlpl.formats.folia.Predicate method), 578
- feat() (pynlpl.formats.folia.Quote method), 296
- feat() (pynlpl.formats.folia.Reference method), 309
- feat() (pynlpl.formats.folia.Row method), 322
- feat() (pynlpl.formats.folia.SemanticRole method), 625
- feat() (pynlpl.formats.folia.SemanticRolesLayer method), 743
- feat() (pynlpl.formats.folia.SenseAnnotation method), 474
- feat() (pynlpl.formats.folia.Sentence method), 336
- feat() (pynlpl.formats.folia.Sentiment method), 590
- feat() (pynlpl.formats.folia.SentimentLayer method), 708
- feat() (pynlpl.formats.folia.Statement method), 601
- feat() (pynlpl.formats.folia.StatementLayer method), 719
- feat() (pynlpl.formats.folia.SubjectivityAnnotation method), 485
- feat() (pynlpl.formats.folia.Suggestion method), 999
- feat() (pynlpl.formats.folia.SynsetFeature method), 875
- feat() (pynlpl.formats.folia.SyntacticUnit method), 613
- feat() (pynlpl.formats.folia.SyntaxLayer method), 731
- feat() (pynlpl.formats.folia.Table method), 350
- feat() (pynlpl.formats.folia.TableHead method), 376
- feat() (pynlpl.formats.folia.Term method), 363
- feat() (pynlpl.formats.folia.Text method), 389
- feat() (pynlpl.formats.folia.TextContent method), 497
- feat() (pynlpl.formats.folia.TextMarkupCorrection method), 842
- feat() (pynlpl.formats.folia.TextMarkupError method), 853
- feat() (pynlpl.formats.folia.TextMarkupGap method), 810
- feat() (pynlpl.formats.folia.TextMarkupString method), 821
- feat() (pynlpl.formats.folia.TextMarkupStyle method), 831
- feat() (pynlpl.formats.folia.TimeSegment method), 636
- feat() (pynlpl.formats.folia.TimingLayer method), 755
- feat() (pynlpl.formats.folia.Whitespace method), 402
- feat() (pynlpl.formats.folia.Word method), 416
- Feature (class in pynlpl.formats.folia), 860
- FIFOQueue (class in pynlpl.datatypes), 5
- Figure (class in pynlpl.formats.folia), 175
- filesampler() (in module pynlpl.evaluation), 11
- find() (pynlpl.datatypes.Trie method), 6
- find\_keyword\_in\_context() (in module pynlpl.textprocessors), 1066
- findcorrectionhandling() (pynlpl.formats.folia.AbstractAnnotationLayer method), 77
- findcorrectionhandling() (pynlpl.formats.folia.AbstractElement method), 28
- findcorrectionhandling() (pynlpl.formats.folia.AbstractSpanAnnotation method), 54
- findcorrectionhandling() (pynlpl.formats.folia.AbstractStructureElement method), 40
- findcorrectionhandling() (pynlpl.formats.folia.AbstractTextMarkup method), 88
- findcorrectionhandling() (pynlpl.formats.folia.AbstractTokenAnnotation method), 65
- findcorrectionhandling() (pynlpl.formats.folia.ActorFeature method), 887

<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Alignment method), 1011	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Feature method), 864
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.AlignReference method), 1022	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Figure method), 181
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Alternative method), 922	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Gap method), 193
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.AlternativeLayer method), 933	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Head method), 205
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.BegindatetimeFeature method), 898	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Headspace method), 790
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Cell method), 103	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.LangAnnotation method), 452
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Chunk method), 520	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.LemmaAnnotation method), 463
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.ChunkingLayer method), 649	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Linebreak method), 218
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.CoreferenceChain method), 532	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.List method), 231
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.CoreferenceLayer method), 661	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.ListItem method), 244
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.CoreferenceLink method), 766	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Metric method), 1044
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Correction method), 946	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.New method), 977
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Current method), 955	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Note method), 257
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Definition method), 116	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Observation method), 567
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.DependenciesLayer method), 672	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.ObservationLayer method), 696
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Dependency method), 543	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Original method), 988
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.DependencyDependency method), 778	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Paragraph method), 270
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Description method), 1033	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Part method), 283
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Division method), 129	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.PhonContent method), 508
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.DomainAnnotation method), 430	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.PosAnnotation method), 441
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.EnddatetimeFeature method), 909	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Predicate method), 578
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.EntitiesLayer method), 684	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Quote method), 296
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Entity method), 555	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Reference method), 309
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Entry method), 142	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Row method), 322
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.ErrorDetection method), 966	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SemanticRole method), 625
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Event method), 155	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SemanticRolesLayer method), 743
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Example method), 168	<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SenseAnnotation method), 474

<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Sentence method), 337	<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractElement class method), 28
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Sentiment method), 590	<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractSpanAnnotation class method), 54
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SentimentLayer method), 708	<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractStructureElement class method), 40
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Statement method), 602	<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractTextMarkup class method), 88
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.StatementLayer method), 719	<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractTokenAnnotation class method), 65
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SubjectivityAnalysis method), 485	<code>findreplaceables()</code> (pynlpl.formats.folia.ActorFeature class method), 887
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Suggestion method), 999	<code>findreplaceables()</code> (pynlpl.formats.folia.Alignment class method), 1011
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SynsetFeature method), 876	<code>findreplaceables()</code> (pynlpl.formats.folia.AlignReference class method), 1022
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SyntacticUnit method), 613	<code>findreplaceables()</code> (pynlpl.formats.folia.Alternative class method), 922
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.SyntaxLayer method), 731	<code>findreplaceables()</code> (pynlpl.formats.folia.AlternativeLayers class method), 933
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Table method), 350	<code>findreplaceables()</code> (pynlpl.formats.folia.BegindatetimeFeature class method), 898
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TableHead method), 376	<code>findreplaceables()</code> (pynlpl.formats.folia.Cell class method), 104
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Term method), 363	<code>findreplaceables()</code> (pynlpl.formats.folia.Chunk class method), 520
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Text method), 389	<code>findreplaceables()</code> (pynlpl.formats.folia.ChunkingLayer class method), 649
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextContent method), 497	<code>findreplaceables()</code> (pynlpl.formats.folia.CoreferenceChain class method), 532
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextMarkupContent method), 842	<code>findreplaceables()</code> (pynlpl.formats.folia.CoreferenceLayer class method), 661
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextMarkupElement method), 853	<code>findreplaceables()</code> (pynlpl.formats.folia.CoreferenceLink class method), 767
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextMarkupGroup method), 810	<code>findreplaceables()</code> (pynlpl.formats.folia.Correction class method), 946
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextMarkupStyle method), 821	<code>findreplaceables()</code> (pynlpl.formats.folia.Current class method), 955
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TextMarkupStyle method), 832	<code>findreplaceables()</code> (pynlpl.formats.folia.Definition class method), 116
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TimeSegment method), 637	<code>findreplaceables()</code> (pynlpl.formats.folia.DependenciesLayer class method), 672
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.TimingLayer method), 755	<code>findreplaceables()</code> (pynlpl.formats.folia.Dependency class method), 543
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Whitespace method), 402	<code>findreplaceables()</code> (pynlpl.formats.folia.DependencyDependent class method), 778
<code>findcorrectionhandling()</code> (pynlpl.formats.folia.Word method), 416	<code>findreplaceables()</code> (pynlpl.formats.folia.Description class method), 1033
<code>finddefaultreference()</code> (pynlpl.formats.folia.PhonContent method), 508	<code>findreplaceables()</code> (pynlpl.formats.folia.Division class method), 129
<code>finddefaultreference()</code> (pynlpl.formats.folia.TextContent method), 497	<code>findreplaceables()</code> (pynlpl.formats.folia.DomainAnnotation class method), 431
<code>findreplaceables()</code> (pynlpl.formats.folia.AbstractAnnotation class method), 77	<code>findreplaceables()</code> (pynlpl.formats.folia.EnddatetimeFeature class method), 909



<code>findreplaceables()</code> (pynlpl.formats.folia.EntitiesLayer class method), 684	<code>findreplaceables()</code> (pynlpl.formats.folia.Quote class method), 296
<code>findreplaceables()</code> (pynlpl.formats.folia.Entity class method), 555	<code>findreplaceables()</code> (pynlpl.formats.folia.Reference class method), 309
<code>findreplaceables()</code> (pynlpl.formats.folia.Entry class method), 142	<code>findreplaceables()</code> (pynlpl.formats.folia.Row class method), 322
<code>findreplaceables()</code> (pynlpl.formats.folia.ErrorDetection class method), 967	<code>findreplaceables()</code> (pynlpl.formats.folia.SemanticRole class method), 625
<code>findreplaceables()</code> (pynlpl.formats.folia.Event class method), 155	<code>findreplaceables()</code> (pynlpl.formats.folia.SemanticRolesLayer class method), 743
<code>findreplaceables()</code> (pynlpl.formats.folia.Example class method), 168	<code>findreplaceables()</code> (pynlpl.formats.folia.SenseAnnotation class method), 475
<code>findreplaceables()</code> (pynlpl.formats.folia.Feature class method), 865	<code>findreplaceables()</code> (pynlpl.formats.folia.Sentence class method), 337
<code>findreplaceables()</code> (pynlpl.formats.folia.Figure class method), 181	<code>findreplaceables()</code> (pynlpl.formats.folia.Sentiment class method), 590
<code>findreplaceables()</code> (pynlpl.formats.folia.Gap class method), 193	<code>findreplaceables()</code> (pynlpl.formats.folia.SentimentLayer class method), 708
<code>findreplaceables()</code> (pynlpl.formats.folia.Head class method), 205	<code>findreplaceables()</code> (pynlpl.formats.folia.Statement class method), 602
<code>findreplaceables()</code> (pynlpl.formats.folia.Headspace class method), 790	<code>findreplaceables()</code> (pynlpl.formats.folia.StatementLayer class method), 719
<code>findreplaceables()</code> (pynlpl.formats.folia.LangAnnotation class method), 453	<code>findreplaceables()</code> (pynlpl.formats.folia.SubjectivityAnnotation class method), 486
<code>findreplaceables()</code> (pynlpl.formats.folia.LemmaAnnotation class method), 464	<code>findreplaceables()</code> (pynlpl.formats.folia.Suggestion class method), 1000
<code>findreplaceables()</code> (pynlpl.formats.folia.Linebreak class method), 218	<code>findreplaceables()</code> (pynlpl.formats.folia.SynsetFeature class method), 876
<code>findreplaceables()</code> (pynlpl.formats.folia.List class method), 231	<code>findreplaceables()</code> (pynlpl.formats.folia.SyntacticUnit class method), 613
<code>findreplaceables()</code> (pynlpl.formats.folia.ListItem class method), 244	<code>findreplaceables()</code> (pynlpl.formats.folia.SyntaxLayer class method), 731
<code>findreplaceables()</code> (pynlpl.formats.folia.Metric class method), 1045	<code>findreplaceables()</code> (pynlpl.formats.folia.Table class method), 350
<code>findreplaceables()</code> (pynlpl.formats.folia.New class method), 978	<code>findreplaceables()</code> (pynlpl.formats.folia.TableHead class method), 376
<code>findreplaceables()</code> (pynlpl.formats.folia.Note class method), 257	<code>findreplaceables()</code> (pynlpl.formats.folia.Term class method), 363
<code>findreplaceables()</code> (pynlpl.formats.folia.Observation class method), 567	<code>findreplaceables()</code> (pynlpl.formats.folia.Text class method), 389
<code>findreplaceables()</code> (pynlpl.formats.folia.ObservationLayer class method), 696	<code>findreplaceables()</code> (pynlpl.formats.folia.TextContent class method), 497
<code>findreplaceables()</code> (pynlpl.formats.folia.Original class method), 989	<code>findreplaceables()</code> (pynlpl.formats.folia.TextMarkupCorrection class method), 842
<code>findreplaceables()</code> (pynlpl.formats.folia.Paragraph class method), 270	<code>findreplaceables()</code> (pynlpl.formats.folia.TextMarkupError class method), 853
<code>findreplaceables()</code> (pynlpl.formats.folia.Part class method), 283	<code>findreplaceables()</code> (pynlpl.formats.folia.TextMarkupGap class method), 810
<code>findreplaceables()</code> (pynlpl.formats.folia.PhonContent class method), 508	<code>findreplaceables()</code> (pynlpl.formats.folia.TextMarkupString class method), 821
<code>findreplaceables()</code> (pynlpl.formats.folia.PosAnnotation class method), 442	<code>findreplaceables()</code> (pynlpl.formats.folia.TextMarkupStyle class method), 832
<code>findreplaceables()</code> (pynlpl.formats.folia.Predicate class method), 578	<code>findreplaceables()</code> (pynlpl.formats.folia.TimeSegment class method), 637

`findreplaceables()` (`pynlpl.formats.folia.TimingLayer` class method), 755

`findreplaceables()` (`pynlpl.formats.folia.Whitespace` class method), 402

`findreplaceables()` (`pynlpl.formats.folia.Word` class method), 416

`findspan()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77

`findspan()` (`pynlpl.formats.folia.ChunkingLayer` method), 649

`findspan()` (`pynlpl.formats.folia.CoreferenceLayer` method), 661

`findspan()` (`pynlpl.formats.folia.DependenciesLayer` method), 673

`findspan()` (`pynlpl.formats.folia.EntitiesLayer` method), 684

`findspan()` (`pynlpl.formats.folia.ObservationLayer` method), 696

`findspan()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 743

`findspan()` (`pynlpl.formats.folia.SentimentLayer` method), 708

`findspan()` (`pynlpl.formats.folia.StatementLayer` method), 720

`findspan()` (`pynlpl.formats.folia.SyntaxLayer` method), 731

`findspan()` (`pynlpl.formats.folia.TimingLayer` method), 755

`findspans()` (`pynlpl.formats.folia.Word` method), 417

`findwords()` (`pynlpl.formats.folia.Document` method), 19

`findwords()` (`pynlpl.formats.folia.Reader` method), 805

`fp_rate()` (`pynlpl.evaluation.ClassEvaluation` method), 9

`FrequencyList` (class in `pynlpl.statistics`), 1062

`fromstring()` (`pynlpl.datatypes.Pattern` static method), 5

`fscore()` (`pynlpl.evaluation.ClassEvaluation` method), 9

## G

`Gap` (class in `pynlpl.formats.folia`), 188

`generate_id()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77

`generate_id()` (`pynlpl.formats.folia.AbstractSpanAnnotationLayer` method), 54

`generate_id()` (`pynlpl.formats.folia.AbstractStructureElement` method), 40

`generate_id()` (`pynlpl.formats.folia.AbstractTokenAnnotationLayer` method), 65

`generate_id()` (`pynlpl.formats.folia.Alternative` method), 922

`generate_id()` (`pynlpl.formats.folia.Cell` method), 104

`generate_id()` (`pynlpl.formats.folia.Chunk` method), 520

`generate_id()` (`pynlpl.formats.folia.ChunkingLayer` method), 649

`generate_id()` (`pynlpl.formats.folia.CoreferenceChain` method), 532

`generate_id()` (`pynlpl.formats.folia.CoreferenceLayer` method), 661

`generate_id()` (`pynlpl.formats.folia.CoreferenceLink` method), 767

`generate_id()` (`pynlpl.formats.folia.Correction` method), 946

`generate_id()` (`pynlpl.formats.folia.Current` method), 955

`generate_id()` (`pynlpl.formats.folia.Definition` method), 117

`generate_id()` (`pynlpl.formats.folia.DependenciesLayer` method), 673

`generate_id()` (`pynlpl.formats.folia.Dependency` method), 544

`generate_id()` (`pynlpl.formats.folia.DependencyDependent` method), 778

`generate_id()` (`pynlpl.formats.folia.Division` method), 129

`generate_id()` (`pynlpl.formats.folia.DomainAnnotation` method), 431

`generate_id()` (`pynlpl.formats.folia.EntitiesLayer` method), 685

`generate_id()` (`pynlpl.formats.folia.Entity` method), 555

`generate_id()` (`pynlpl.formats.folia.Entry` method), 142

`generate_id()` (`pynlpl.formats.folia.ErrorDetection` method), 967

`generate_id()` (`pynlpl.formats.folia.Event` method), 155

`generate_id()` (`pynlpl.formats.folia.Example` method), 168

`generate_id()` (`pynlpl.formats.folia.Figure` method), 182

`generate_id()` (`pynlpl.formats.folia.Head` method), 206

`generate_id()` (`pynlpl.formats.folia.Headspan` method), 790

`generate_id()` (`pynlpl.formats.folia.LangAnnotation` method), 453

`generate_id()` (`pynlpl.formats.folia.LemmaAnnotation` method), 464

`generate_id()` (`pynlpl.formats.folia.Linebreak` method), 219

`generate_id()` (`pynlpl.formats.folia.List` method), 231

`generate_id()` (`pynlpl.formats.folia.ListItem` method), 244

`generate_id()` (`pynlpl.formats.folia.New` method), 978

`generate_id()` (`pynlpl.formats.folia.Note` method), 257

`generate_id()` (`pynlpl.formats.folia.Observation` method), 567

`generate_id()` (`pynlpl.formats.folia.ObservationLayer` method), 696

`generate_id()` (`pynlpl.formats.folia.Original` method), 989

`generate_id()` (`pynlpl.formats.folia.Paragraph` method), 270

`generate_id()` (`pynlpl.formats.folia.Part` method), 283

`generate_id()` (`pynlpl.formats.folia.PosAnnotation` method), 442

`generate_id()` (`pynlpl.formats.folia.Predicate` method), 578

`generate_id()` (`pynlpl.formats.folia.Quote` method), 296

[generate\\_id\(\) \(pynlpl.formats.folia.Reference method\), 309](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Row method\), 322](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SemanticRole method\), 625](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 743](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 475](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Sentence method\), 337](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Sentiment method\), 590](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SentimentLayer method\), 708](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Statement method\), 602](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.StatementLayer method\), 720](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 486](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Suggestion method\), 1000](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 613](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 732](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Table method\), 350](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.TableHead method\), 376](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Term method\), 363](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Text method\), 389](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.TimeSegment method\), 637](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.TimingLayer method\), 755](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Whitespace method\), 402](#)  
[generate\\_id\(\) \(pynlpl.formats.folia.Word method\), 417](#)  
[getalignedtarget\(\) \(pynlpl.formats.giza.GizaSentenceAlignment method\), 1053](#)  
[getcorrection\(\) \(pynlpl.formats.folia.Word method\), 417](#)  
[getcorrections\(\) \(pynlpl.formats.folia.Word method\), 417](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 77](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractElement method\), 28](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 54](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 40](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractTextMarkup method\), 88](#)  
[getindex\(\) \(pynlpl.formats.folia.AbstractTokenAnnotation method\), 65](#)  
[getindex\(\) \(pynlpl.formats.folia.ActorFeature method\), 887](#)  
[getindex\(\) \(pynlpl.formats.folia.Alignment method\), 1011](#)  
[getindex\(\) \(pynlpl.formats.folia.AlignReference method\), 1022](#)  
[getindex\(\) \(pynlpl.formats.folia.Alternative method\), 922](#)  
[getindex\(\) \(pynlpl.formats.folia.AlternativeLayers method\), 933](#)  
[getindex\(\) \(pynlpl.formats.folia.BegindatetimeFeature method\), 898](#)  
[getindex\(\) \(pynlpl.formats.folia.Cell method\), 104](#)  
[getindex\(\) \(pynlpl.formats.folia.Chunk method\), 520](#)  
[getindex\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 649](#)  
[getindex\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 532](#)  
[getindex\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 661](#)  
[getindex\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 767](#)  
[getindex\(\) \(pynlpl.formats.folia.Correction method\), 946](#)  
[getindex\(\) \(pynlpl.formats.folia.Current method\), 956](#)  
[getindex\(\) \(pynlpl.formats.folia.Definition method\), 117](#)  
[getindex\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 673](#)  
[getindex\(\) \(pynlpl.formats.folia.Dependency method\), 544](#)  
[getindex\(\) \(pynlpl.formats.folia.DependencyDependent method\), 778](#)  
[getindex\(\) \(pynlpl.formats.folia.Description method\), 1034](#)  
[getindex\(\) \(pynlpl.formats.folia.Division method\), 130](#)  
[getindex\(\) \(pynlpl.formats.folia.DomainAnnotation method\), 431](#)  
[getindex\(\) \(pynlpl.formats.folia.EnddatetimeFeature method\), 909](#)  
[getindex\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 685](#)  
[getindex\(\) \(pynlpl.formats.folia.Entity method\), 555](#)  
[getindex\(\) \(pynlpl.formats.folia.Entry method\), 142](#)  
[getindex\(\) \(pynlpl.formats.folia.ErrorDetection method\), 967](#)  
[getindex\(\) \(pynlpl.formats.folia.Event method\), 156](#)  
[getindex\(\) \(pynlpl.formats.folia.Example method\), 168](#)  
[getindex\(\) \(pynlpl.formats.folia.Feature method\), 865](#)  
[getindex\(\) \(pynlpl.formats.folia.Figure method\), 182](#)  
[getindex\(\) \(pynlpl.formats.folia.Gap method\), 193](#)  
[getindex\(\) \(pynlpl.formats.folia.Head method\), 206](#)  
[getindex\(\) \(pynlpl.formats.folia.Headspan method\), 790](#)  
[getindex\(\) \(pynlpl.formats.folia.LangAnnotation method\), 453](#)  
[getindex\(\) \(pynlpl.formats.folia.LemmaAnnotation](#)

method), 464  
getindex() (pynlpl.formats.folia.Linebreak method), 219  
getindex() (pynlpl.formats.folia.List method), 231  
getindex() (pynlpl.formats.folia.ListItem method), 244  
getindex() (pynlpl.formats.folia.Metric method), 1045  
getindex() (pynlpl.formats.folia.New method), 978  
getindex() (pynlpl.formats.folia.Note method), 257  
getindex() (pynlpl.formats.folia.Observation method), 567  
getindex() (pynlpl.formats.folia.ObservationLayer method), 696  
getindex() (pynlpl.formats.folia.Original method), 989  
getindex() (pynlpl.formats.folia.Paragraph method), 271  
getindex() (pynlpl.formats.folia.Part method), 284  
getindex() (pynlpl.formats.folia.PhonContent method), 508  
getindex() (pynlpl.formats.folia.PosAnnotation method), 442  
getindex() (pynlpl.formats.folia.Predicate method), 578  
getindex() (pynlpl.formats.folia.Quote method), 296  
getindex() (pynlpl.formats.folia.Reference method), 310  
getindex() (pynlpl.formats.folia.Row method), 322  
getindex() (pynlpl.formats.folia.SemanticRole method), 625  
getindex() (pynlpl.formats.folia.SemanticRolesLayer method), 743  
getindex() (pynlpl.formats.folia.SenseAnnotation method), 475  
getindex() (pynlpl.formats.folia.Sentence method), 337  
getindex() (pynlpl.formats.folia.Sentiment method), 590  
getindex() (pynlpl.formats.folia.SentimentLayer method), 708  
getindex() (pynlpl.formats.folia.Statement method), 602  
getindex() (pynlpl.formats.folia.StatementLayer method), 720  
getindex() (pynlpl.formats.folia.SubjectivityAnnotation method), 486  
getindex() (pynlpl.formats.folia.Suggestion method), 1000  
getindex() (pynlpl.formats.folia.SynsetFeature method), 876  
getindex() (pynlpl.formats.folia.SyntacticUnit method), 613  
getindex() (pynlpl.formats.folia.SyntaxLayer method), 732  
getindex() (pynlpl.formats.folia.Table method), 350  
getindex() (pynlpl.formats.folia.TableHead method), 376  
getindex() (pynlpl.formats.folia.Term method), 363  
getindex() (pynlpl.formats.folia.Text method), 389  
getindex() (pynlpl.formats.folia.TextContent method), 497  
getindex() (pynlpl.formats.folia.TextMarkupCorrection method), 842  
getindex() (pynlpl.formats.folia.TextMarkupError method), 853  
getindex() (pynlpl.formats.folia.TextMarkupGap method), 810  
getindex() (pynlpl.formats.folia.TextMarkupString method), 821  
getindex() (pynlpl.formats.folia.TextMarkupStyle method), 832  
getindex() (pynlpl.formats.folia.TimeSegment method), 637  
getindex() (pynlpl.formats.folia.TimingLayer method), 755  
getindex() (pynlpl.formats.folia.Whitespace method), 402  
getindex() (pynlpl.formats.folia.Word method), 417  
getmetadata() (pynlpl.formats.folia.AbstractAnnotationLayer method), 77  
getmetadata() (pynlpl.formats.folia.AbstractElement method), 28  
getmetadata() (pynlpl.formats.folia.AbstractSpanAnnotation method), 54  
getmetadata() (pynlpl.formats.folia.AbstractStructureElement method), 41  
getmetadata() (pynlpl.formats.folia.AbstractTextMarkup method), 88  
getmetadata() (pynlpl.formats.folia.AbstractTokenAnnotation method), 65  
getmetadata() (pynlpl.formats.folia.ActorFeature method), 887  
getmetadata() (pynlpl.formats.folia.Alignment method), 1011  
getmetadata() (pynlpl.formats.folia.AlignReference method), 1022  
getmetadata() (pynlpl.formats.folia.Alternative method), 922  
getmetadata() (pynlpl.formats.folia.AlternativeLayers method), 933  
getmetadata() (pynlpl.formats.folia.BeginatetimeFeature method), 898  
getmetadata() (pynlpl.formats.folia.Cell method), 104  
getmetadata() (pynlpl.formats.folia.Chunk method), 520  
getmetadata() (pynlpl.formats.folia.ChunkingLayer method), 649  
getmetadata() (pynlpl.formats.folia.CoreferenceChain method), 532  
getmetadata() (pynlpl.formats.folia.CoreferenceLayer method), 661  
getmetadata() (pynlpl.formats.folia.CoreferenceLink method), 767  
getmetadata() (pynlpl.formats.folia.Correction method), 946  
getmetadata() (pynlpl.formats.folia.Current method), 956  
getmetadata() (pynlpl.formats.folia.Definition method), 117  
getmetadata() (pynlpl.formats.folia.DependenciesLayer method), 673

- getmetadata() (pynlpl.formats.folia.Dependency method), 544
- getmetadata() (pynlpl.formats.folia.DependencyDependent method), 778
- getmetadata() (pynlpl.formats.folia.Description method), 1034
- getmetadata() (pynlpl.formats.folia.Division method), 130
- getmetadata() (pynlpl.formats.folia.DomainAnnotation method), 431
- getmetadata() (pynlpl.formats.folia.EnddatetimeFeature method), 909
- getmetadata() (pynlpl.formats.folia.EntitiesLayer method), 685
- getmetadata() (pynlpl.formats.folia.Entity method), 555
- getmetadata() (pynlpl.formats.folia.Entry method), 143
- getmetadata() (pynlpl.formats.folia.ErrorDetection method), 967
- getmetadata() (pynlpl.formats.folia.Event method), 156
- getmetadata() (pynlpl.formats.folia.Example method), 169
- getmetadata() (pynlpl.formats.folia.Feature method), 865
- getmetadata() (pynlpl.formats.folia.Figure method), 182
- getmetadata() (pynlpl.formats.folia.Gap method), 193
- getmetadata() (pynlpl.formats.folia.Head method), 206
- getmetadata() (pynlpl.formats.folia.Headspan method), 790
- getmetadata() (pynlpl.formats.folia.LangAnnotation method), 453
- getmetadata() (pynlpl.formats.folia.LemmaAnnotation method), 464
- getmetadata() (pynlpl.formats.folia.Linebreak method), 219
- getmetadata() (pynlpl.formats.folia.List method), 232
- getmetadata() (pynlpl.formats.folia.ListItem method), 245
- getmetadata() (pynlpl.formats.folia.Metric method), 1045
- getmetadata() (pynlpl.formats.folia.New method), 978
- getmetadata() (pynlpl.formats.folia.Note method), 258
- getmetadata() (pynlpl.formats.folia.Observation method), 567
- getmetadata() (pynlpl.formats.folia.ObservationLayer method), 696
- getmetadata() (pynlpl.formats.folia.Original method), 989
- getmetadata() (pynlpl.formats.folia.Paragraph method), 271
- getmetadata() (pynlpl.formats.folia.Part method), 284
- getmetadata() (pynlpl.formats.folia.PhonContent method), 508
- getmetadata() (pynlpl.formats.folia.PosAnnotation method), 442
- getmetadata() (pynlpl.formats.folia.Predicate method), 578
- getmetadata() (pynlpl.formats.folia.Quote method), 297
- getmetadata() (pynlpl.formats.folia.Reference method), 310
- getmetadata() (pynlpl.formats.folia.Row method), 323
- getmetadata() (pynlpl.formats.folia.SemanticRole method), 625
- getmetadata() (pynlpl.formats.folia.SemanticRolesLayer method), 743
- getmetadata() (pynlpl.formats.folia.SenseAnnotation method), 475
- getmetadata() (pynlpl.formats.folia.Sentence method), 337
- getmetadata() (pynlpl.formats.folia.Sentiment method), 590
- getmetadata() (pynlpl.formats.folia.SentimentLayer method), 708
- getmetadata() (pynlpl.formats.folia.Statement method), 602
- getmetadata() (pynlpl.formats.folia.StatementLayer method), 720
- getmetadata() (pynlpl.formats.folia.SubjectivityAnnotation method), 486
- getmetadata() (pynlpl.formats.folia.Suggestion method), 1000
- getmetadata() (pynlpl.formats.folia.SynsetFeature method), 876
- getmetadata() (pynlpl.formats.folia.SyntacticUnit method), 613
- getmetadata() (pynlpl.formats.folia.SyntaxLayer method), 732
- getmetadata() (pynlpl.formats.folia.Table method), 350
- getmetadata() (pynlpl.formats.folia.TableHead method), 376
- getmetadata() (pynlpl.formats.folia.Term method), 363
- getmetadata() (pynlpl.formats.folia.Text method), 389
- getmetadata() (pynlpl.formats.folia.TextContent method), 497
- getmetadata() (pynlpl.formats.folia.TextMarkupCorrection method), 842
- getmetadata() (pynlpl.formats.folia.TextMarkupError method), 853
- getmetadata() (pynlpl.formats.folia.TextMarkupGap method), 810
- getmetadata() (pynlpl.formats.folia.TextMarkupString method), 821
- getmetadata() (pynlpl.formats.folia.TextMarkupStyle method), 832
- getmetadata() (pynlpl.formats.folia.TimeSegment method), 637
- getmetadata() (pynlpl.formats.folia.TimingLayer method), 755
- getmetadata() (pynlpl.formats.folia.Whitespace method), 402
- getmetadata() (pynlpl.formats.folia.Word method), 417



`getreference()` (`pynlpl.formats.folia.PhonContent` method), 508

`getreference()` (`pynlpl.formats.folia.TextContent` method), 497

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 77

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractElement` method), 28

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 54

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractStructureElement` method), 41

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 88

`gettextdelimiter()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 66

`gettextdelimiter()` (`pynlpl.formats.folia.ActorFeature` method), 887

`gettextdelimiter()` (`pynlpl.formats.folia.Alignment` method), 1011

`gettextdelimiter()` (`pynlpl.formats.folia.AlignReference` method), 1022

`gettextdelimiter()` (`pynlpl.formats.folia.Alternative` method), 922

`gettextdelimiter()` (`pynlpl.formats.folia.AlternativeLayers` method), 933

`gettextdelimiter()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 898

`gettextdelimiter()` (`pynlpl.formats.folia.Cell` method), 104

`gettextdelimiter()` (`pynlpl.formats.folia.Chunk` method), 520

`gettextdelimiter()` (`pynlpl.formats.folia.ChunkingLayer` method), 649

`gettextdelimiter()` (`pynlpl.formats.folia.CoreferenceChain` method), 532

`gettextdelimiter()` (`pynlpl.formats.folia.CoreferenceLayer` method), 661

`gettextdelimiter()` (`pynlpl.formats.folia.CoreferenceLink` method), 767

`gettextdelimiter()` (`pynlpl.formats.folia.Correction` method), 946

`gettextdelimiter()` (`pynlpl.formats.folia.Current` method), 956

`gettextdelimiter()` (`pynlpl.formats.folia.Definition` method), 117

`gettextdelimiter()` (`pynlpl.formats.folia.DependenciesLayer` method), 673

`gettextdelimiter()` (`pynlpl.formats.folia.Dependency` method), 544

`gettextdelimiter()` (`pynlpl.formats.folia.DependencyDependency` method), 779

`gettextdelimiter()` (`pynlpl.formats.folia.Description` method), 1034

`gettextdelimiter()` (`pynlpl.formats.folia.Division` method), 130

`gettextdelimiter()` (`pynlpl.formats.folia.DomainAnnotation` method), 431

`gettextdelimiter()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 909

`gettextdelimiter()` (`pynlpl.formats.folia.EntitiesLayer` method), 685

`gettextdelimiter()` (`pynlpl.formats.folia.Entity` method), 555

`gettextdelimiter()` (`pynlpl.formats.folia.Entry` method), 143

`gettextdelimiter()` (`pynlpl.formats.folia.ErrorDetection` method), 967

`gettextdelimiter()` (`pynlpl.formats.folia.Event` method), 156

`gettextdelimiter()` (`pynlpl.formats.folia.Example` method), 169

`gettextdelimiter()` (`pynlpl.formats.folia.Feature` method), 865

`gettextdelimiter()` (`pynlpl.formats.folia.Figure` method), 182

`gettextdelimiter()` (`pynlpl.formats.folia.Gap` method), 193

`gettextdelimiter()` (`pynlpl.formats.folia.Head` method), 206

`gettextdelimiter()` (`pynlpl.formats.folia.Headspan` method), 790

`gettextdelimiter()` (`pynlpl.formats.folia.LangAnnotation` method), 453

`gettextdelimiter()` (`pynlpl.formats.folia.LemmaAnnotation` method), 464

`gettextdelimiter()` (`pynlpl.formats.folia.Linebreak` method), 219

`gettextdelimiter()` (`pynlpl.formats.folia.List` method), 232

`gettextdelimiter()` (`pynlpl.formats.folia.ListItem` method), 245

`gettextdelimiter()` (`pynlpl.formats.folia.Metric` method), 1045

`gettextdelimiter()` (`pynlpl.formats.folia.New` method), 978

`gettextdelimiter()` (`pynlpl.formats.folia.Note` method), 258

`gettextdelimiter()` (`pynlpl.formats.folia.Observation` method), 567

`gettextdelimiter()` (`pynlpl.formats.folia.ObservationLayer` method), 696

`gettextdelimiter()` (`pynlpl.formats.folia.Original` method), 989

`gettextdelimiter()` (`pynlpl.formats.folia.Paragraph` method), 271

`gettextdelimiter()` (`pynlpl.formats.folia.Part` method), 284

`gettextdelimiter()` (`pynlpl.formats.folia.PhonContent` method), 508

`gettextdelimiter()` (`pynlpl.formats.folia.PosAnnotation` method), 442

- [gettextdelimiter\(\) \(pynlpl.formats.folia.Predicate method\), 579](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Quote method\), 297](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Reference method\), 310](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Row method\), 323](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SemanticRole method\), 625](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 743](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 475](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Sentence method\), 337](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Sentiment method\), 590](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SentimentLayer method\), 708](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Statement method\), 602](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.StatementLayer method\), 720](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 486](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Suggestion method\), 1000](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SynsetFeature method\), 876](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 614](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 732](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Table method\), 350](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TableHead method\), 376](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Term method\), 363](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Text method\), 389](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextContent method\), 497](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextMarkupCorrection method\), 843](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextMarkupError method\), 853](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextMarkupGap method\), 811](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextMarkupString method\), 821](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 832](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TimeSegment method\), 637](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.TimingLayer method\), 755](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Whitespace method\), 402](#)  
[gettextdelimiter\(\) \(pynlpl.formats.folia.Word method\), 417](#)  
[GizaModel \(class in pynlpl.formats.giza\), 1053](#)  
[GizaSentenceAlignment \(class in pynlpl.formats.giza\), 1053](#)
- ## H
- [hasannotation\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 77](#)  
[hasannotation\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 54](#)  
[hasannotation\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 41](#)  
[hasannotation\(\) \(pynlpl.formats.folia.AllowTokenAnnotation method\), 49](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Alternative method\), 922](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Cell method\), 104](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Chunk method\), 520](#)  
[hasannotation\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 649](#)  
[hasannotation\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 532](#)  
[hasannotation\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 661](#)  
[hasannotation\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 767](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Definition method\), 117](#)  
[hasannotation\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 673](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Dependency method\), 544](#)  
[hasannotation\(\) \(pynlpl.formats.folia.DependencyDependent method\), 779](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Division method\), 130](#)  
[hasannotation\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 685](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Entity method\), 555](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Entry method\), 143](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Event method\), 156](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Example method\), 169](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Figure method\), 182](#)  
[hasannotation\(\) \(pynlpl.formats.folia.Head method\), 206](#)

`hasannotation()` (`pynlpl.formats.folia.HeadsSpan` method), 790

`hasannotation()` (`pynlpl.formats.folia.Linebreak` method), 219

`hasannotation()` (`pynlpl.formats.folia.List` method), 232

`hasannotation()` (`pynlpl.formats.folia.ListItem` method), 245

`hasannotation()` (`pynlpl.formats.folia.Note` method), 258

`hasannotation()` (`pynlpl.formats.folia.Observation` method), 567

`hasannotation()` (`pynlpl.formats.folia.ObservationLayer` method), 696

`hasannotation()` (`pynlpl.formats.folia.Paragraph` method), 271

`hasannotation()` (`pynlpl.formats.folia.Part` method), 284

`hasannotation()` (`pynlpl.formats.folia.Predicate` method), 579

`hasannotation()` (`pynlpl.formats.folia.Quote` method), 297

`hasannotation()` (`pynlpl.formats.folia.Reference` method), 310

`hasannotation()` (`pynlpl.formats.folia.Row` method), 323

`hasannotation()` (`pynlpl.formats.folia.SemanticRole` method), 625

`hasannotation()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 743

`hasannotation()` (`pynlpl.formats.folia.Sentence` method), 337

`hasannotation()` (`pynlpl.formats.folia.Sentiment` method), 590

`hasannotation()` (`pynlpl.formats.folia.SentimentLayer` method), 708

`hasannotation()` (`pynlpl.formats.folia.Statement` method), 602

`hasannotation()` (`pynlpl.formats.folia.StatementLayer` method), 720

`hasannotation()` (`pynlpl.formats.folia.SyntacticUnit` method), 614

`hasannotation()` (`pynlpl.formats.folia.SyntaxLayer` method), 732

`hasannotation()` (`pynlpl.formats.folia.Table` method), 350

`hasannotation()` (`pynlpl.formats.folia.TableHead` method), 376

`hasannotation()` (`pynlpl.formats.folia.Term` method), 363

`hasannotation()` (`pynlpl.formats.folia.Text` method), 390

`hasannotation()` (`pynlpl.formats.folia.TimeSegment` method), 637

`hasannotation()` (`pynlpl.formats.folia.TimingLayer` method), 755

`hasannotation()` (`pynlpl.formats.folia.Whitespace` method), 403

`hasannotation()` (`pynlpl.formats.folia.Word` method), 417

`hasannotationlayer()` (`pynlpl.formats.folia.AbstractStructureElement` method), 41

`hasannotationlayer()` (`pynlpl.formats.folia.Cell` method), 104

`hasannotationlayer()` (`pynlpl.formats.folia.Definition` method), 117

`hasannotationlayer()` (`pynlpl.formats.folia.Division` method), 130

`hasannotationlayer()` (`pynlpl.formats.folia.Entry` method), 143

`hasannotationlayer()` (`pynlpl.formats.folia.Event` method), 156

`hasannotationlayer()` (`pynlpl.formats.folia.Example` method), 169

`hasannotationlayer()` (`pynlpl.formats.folia.Figure` method), 182

`hasannotationlayer()` (`pynlpl.formats.folia.Head` method), 206

`hasannotationlayer()` (`pynlpl.formats.folia.Linebreak` method), 219

`hasannotationlayer()` (`pynlpl.formats.folia.List` method), 232

`hasannotationlayer()` (`pynlpl.formats.folia.ListItem` method), 245

`hasannotationlayer()` (`pynlpl.formats.folia.Note` method), 258

`hasannotationlayer()` (`pynlpl.formats.folia.Paragraph` method), 271

`hasannotationlayer()` (`pynlpl.formats.folia.Part` method), 284

`hasannotationlayer()` (`pynlpl.formats.folia.Quote` method), 297

`hasannotationlayer()` (`pynlpl.formats.folia.Reference` method), 310

`hasannotationlayer()` (`pynlpl.formats.folia.Row` method), 323

`hasannotationlayer()` (`pynlpl.formats.folia.Sentence` method), 337

`hasannotationlayer()` (`pynlpl.formats.folia.Table` method), 351

`hasannotationlayer()` (`pynlpl.formats.folia.TableHead` method), 377

`hasannotationlayer()` (`pynlpl.formats.folia.Term` method), 364

`hasannotationlayer()` (`pynlpl.formats.folia.Text` method), 390

`hasannotationlayer()` (`pynlpl.formats.folia.Whitespace` method), 403

`hasannotationlayer()` (`pynlpl.formats.folia.Word` method), 417

`hascurrent()` (`pynlpl.formats.folia.Correction` method), 946

`hasnew()` (`pynlpl.formats.folia.Correction` method), 946

`hasoriginal()` (`pynlpl.formats.folia.Correction` method), 946

`hasphon()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 78



- hasphon() (pynlpl.formats.folia.AbstractElement method), 28
- hasphon() (pynlpl.formats.folia.AbstractSpanAnnotation method), 55
- hasphon() (pynlpl.formats.folia.AbstractStructureElement method), 41
- hasphon() (pynlpl.formats.folia.AbstractTextMarkup method), 88
- hasphon() (pynlpl.formats.folia.AbstractTokenAnnotation method), 66
- hasphon() (pynlpl.formats.folia.ActorFeature method), 887
- hasphon() (pynlpl.formats.folia.Alignment method), 1011
- hasphon() (pynlpl.formats.folia.AlignReference method), 1022
- hasphon() (pynlpl.formats.folia.Alternative method), 922
- hasphon() (pynlpl.formats.folia.AlternativeLayers method), 933
- hasphon() (pynlpl.formats.folia.BegindatetimeFeature method), 898
- hasphon() (pynlpl.formats.folia.Cell method), 104
- hasphon() (pynlpl.formats.folia.Chunk method), 521
- hasphon() (pynlpl.formats.folia.ChunkingLayer method), 649
- hasphon() (pynlpl.formats.folia.CoreferenceChain method), 532
- hasphon() (pynlpl.formats.folia.CoreferenceLayer method), 661
- hasphon() (pynlpl.formats.folia.CoreferenceLink method), 767
- hasphon() (pynlpl.formats.folia.Correction method), 947
- hasphon() (pynlpl.formats.folia.Current method), 956
- hasphon() (pynlpl.formats.folia.Definition method), 117
- hasphon() (pynlpl.formats.folia.DependenciesLayer method), 673
- hasphon() (pynlpl.formats.folia.Dependency method), 544
- hasphon() (pynlpl.formats.folia.DependencyDependent method), 779
- hasphon() (pynlpl.formats.folia.Description method), 1034
- hasphon() (pynlpl.formats.folia.Division method), 130
- hasphon() (pynlpl.formats.folia.DomainAnnotation method), 431
- hasphon() (pynlpl.formats.folia.EnddatetimeFeature method), 909
- hasphon() (pynlpl.formats.folia.EntitiesLayer method), 685
- hasphon() (pynlpl.formats.folia.Entity method), 555
- hasphon() (pynlpl.formats.folia.Entry method), 143
- hasphon() (pynlpl.formats.folia.ErrorDetection method), 967
- hasphon() (pynlpl.formats.folia.Event method), 156
- hasphon() (pynlpl.formats.folia.Example method), 169
- hasphon() (pynlpl.formats.folia.Feature method), 865
- hasphon() (pynlpl.formats.folia.Figure method), 182
- hasphon() (pynlpl.formats.folia.Gap method), 193
- hasphon() (pynlpl.formats.folia.Head method), 206
- hasphon() (pynlpl.formats.folia.Headspace method), 790
- hasphon() (pynlpl.formats.folia.LangAnnotation method), 453
- hasphon() (pynlpl.formats.folia.LemmaAnnotation method), 464
- hasphon() (pynlpl.formats.folia.Linebreak method), 219
- hasphon() (pynlpl.formats.folia.List method), 232
- hasphon() (pynlpl.formats.folia.ListItem method), 245
- hasphon() (pynlpl.formats.folia.Metric method), 1045
- hasphon() (pynlpl.formats.folia.New method), 978
- hasphon() (pynlpl.formats.folia.Note method), 258
- hasphon() (pynlpl.formats.folia.Observation method), 567
- hasphon() (pynlpl.formats.folia.ObservationLayer method), 697
- hasphon() (pynlpl.formats.folia.Original method), 989
- hasphon() (pynlpl.formats.folia.Paragraph method), 271
- hasphon() (pynlpl.formats.folia.Part method), 284
- hasphon() (pynlpl.formats.folia.PhonContent method), 508
- hasphon() (pynlpl.formats.folia.PosAnnotation method), 442
- hasphon() (pynlpl.formats.folia.Predicate method), 579
- hasphon() (pynlpl.formats.folia.Quote method), 297
- hasphon() (pynlpl.formats.folia.Reference method), 310
- hasphon() (pynlpl.formats.folia.Row method), 323
- hasphon() (pynlpl.formats.folia.SemanticRole method), 625
- hasphon() (pynlpl.formats.folia.SemanticRolesLayer method), 744
- hasphon() (pynlpl.formats.folia.SenseAnnotation method), 475
- hasphon() (pynlpl.formats.folia.Sentence method), 337
- hasphon() (pynlpl.formats.folia.Sentiment method), 590
- hasphon() (pynlpl.formats.folia.SentimentLayer method), 708
- hasphon() (pynlpl.formats.folia.Statement method), 602
- hasphon() (pynlpl.formats.folia.StatementLayer method), 720
- hasphon() (pynlpl.formats.folia.SubjectivityAnnotation method), 486
- hasphon() (pynlpl.formats.folia.Suggestion method), 1000
- hasphon() (pynlpl.formats.folia.SynsetFeature method), 876
- hasphon() (pynlpl.formats.folia.SyntacticUnit method), 614
- hasphon() (pynlpl.formats.folia.SyntaxLayer method), 732
- hasphon() (pynlpl.formats.folia.Table method), 351

`hasphon()` (`pynlpl.formats.folia.TableHead` method), 377  
`hasphon()` (`pynlpl.formats.folia.Term` method), 364  
`hasphon()` (`pynlpl.formats.folia.Text` method), 390  
`hasphon()` (`pynlpl.formats.folia.TextContent` method), 498  
`hasphon()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 843  
`hasphon()` (`pynlpl.formats.folia.TextMarkupError` method), 853  
`hasphon()` (`pynlpl.formats.folia.TextMarkupGap` method), 811  
`hasphon()` (`pynlpl.formats.folia.TextMarkupString` method), 821  
`hasphon()` (`pynlpl.formats.folia.TextMarkupStyle` method), 832  
`hasphon()` (`pynlpl.formats.folia.TimeSegment` method), 637  
`hasphon()` (`pynlpl.formats.folia.TimingLayer` method), 755  
`hasphon()` (`pynlpl.formats.folia.Whitespace` method), 403  
`hasphon()` (`pynlpl.formats.folia.Word` method), 417  
`hassuggestions()` (`pynlpl.formats.folia.Correction` method), 947  
`hastext()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 78  
`hastext()` (`pynlpl.formats.folia.AbstractElement` method), 28  
`hastext()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 55  
`hastext()` (`pynlpl.formats.folia.AbstractStructureElement` method), 41  
`hastext()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 89  
`hastext()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 66  
`hastext()` (`pynlpl.formats.folia.ActorFeature` method), 888  
`hastext()` (`pynlpl.formats.folia.Alignment` method), 1011  
`hastext()` (`pynlpl.formats.folia.AlignReference` method), 1023  
`hastext()` (`pynlpl.formats.folia.Alternative` method), 923  
`hastext()` (`pynlpl.formats.folia.AlternativeLayers` method), 934  
`hastext()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 899  
`hastext()` (`pynlpl.formats.folia.Cell` method), 104  
`hastext()` (`pynlpl.formats.folia.Chunk` method), 521  
`hastext()` (`pynlpl.formats.folia.ChunkingLayer` method), 650  
`hastext()` (`pynlpl.formats.folia.CoreferenceChain` method), 532  
`hastext()` (`pynlpl.formats.folia.CoreferenceLayer` method), 662  
`hastext()` (`pynlpl.formats.folia.CoreferenceLink` method), 767  
`hastext()` (`pynlpl.formats.folia.Correction` method), 947  
`hastext()` (`pynlpl.formats.folia.Current` method), 956  
`hastext()` (`pynlpl.formats.folia.Definition` method), 117  
`hastext()` (`pynlpl.formats.folia.DependenciesLayer` method), 673  
`hastext()` (`pynlpl.formats.folia.Dependency` method), 544  
`hastext()` (`pynlpl.formats.folia.DependencyDependent` method), 779  
`hastext()` (`pynlpl.formats.folia.Description` method), 1034  
`hastext()` (`pynlpl.formats.folia.Division` method), 130  
`hastext()` (`pynlpl.formats.folia.DomainAnnotation` method), 431  
`hastext()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 910  
`hastext()` (`pynlpl.formats.folia.EntitiesLayer` method), 685  
`hastext()` (`pynlpl.formats.folia.Entity` method), 556  
`hastext()` (`pynlpl.formats.folia.Entry` method), 143  
`hastext()` (`pynlpl.formats.folia.ErrorDetection` method), 967  
`hastext()` (`pynlpl.formats.folia.Event` method), 156  
`hastext()` (`pynlpl.formats.folia.Example` method), 169  
`hastext()` (`pynlpl.formats.folia.Feature` method), 865  
`hastext()` (`pynlpl.formats.folia.Figure` method), 182  
`hastext()` (`pynlpl.formats.folia.Gap` method), 194  
`hastext()` (`pynlpl.formats.folia.Head` method), 206  
`hastext()` (`pynlpl.formats.folia.Headspan` method), 791  
`hastext()` (`pynlpl.formats.folia.LangAnnotation` method), 453  
`hastext()` (`pynlpl.formats.folia.LemmaAnnotation` method), 464  
`hastext()` (`pynlpl.formats.folia.Linebreak` method), 219  
`hastext()` (`pynlpl.formats.folia.List` method), 232  
`hastext()` (`pynlpl.formats.folia.ListItem` method), 245  
`hastext()` (`pynlpl.formats.folia.Metric` method), 1045  
`hastext()` (`pynlpl.formats.folia.New` method), 978  
`hastext()` (`pynlpl.formats.folia.Note` method), 258  
`hastext()` (`pynlpl.formats.folia.Observation` method), 567  
`hastext()` (`pynlpl.formats.folia.ObservationLayer` method), 697  
`hastext()` (`pynlpl.formats.folia.Original` method), 989  
`hastext()` (`pynlpl.formats.folia.Paragraph` method), 271  
`hastext()` (`pynlpl.formats.folia.Part` method), 284  
`hastext()` (`pynlpl.formats.folia.PhonContent` method), 508  
`hastext()` (`pynlpl.formats.folia.PosAnnotation` method), 442  
`hastext()` (`pynlpl.formats.folia.Predicate` method), 579  
`hastext()` (`pynlpl.formats.folia.Quote` method), 297  
`hastext()` (`pynlpl.formats.folia.Reference` method), 310  
`hastext()` (`pynlpl.formats.folia.Row` method), 323  
`hastext()` (`pynlpl.formats.folia.SemanticRole` method), 626  
`hastext()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 744

- [hastext\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 475](#)  
[hastext\(\) \(pynlpl.formats.folia.Sentence method\), 337](#)  
[hastext\(\) \(pynlpl.formats.folia.Sentiment method\), 591](#)  
[hastext\(\) \(pynlpl.formats.folia.SentimentLayer method\), 709](#)  
[hastext\(\) \(pynlpl.formats.folia.Statement method\), 602](#)  
[hastext\(\) \(pynlpl.formats.folia.StatementLayer method\), 720](#)  
[hastext\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 486](#)  
[hastext\(\) \(pynlpl.formats.folia.Suggestion method\), 1000](#)  
[hastext\(\) \(pynlpl.formats.folia.SynsetFeature method\), 876](#)  
[hastext\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 614](#)  
[hastext\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 732](#)  
[hastext\(\) \(pynlpl.formats.folia.Table method\), 351](#)  
[hastext\(\) \(pynlpl.formats.folia.TableHead method\), 377](#)  
[hastext\(\) \(pynlpl.formats.folia.Term method\), 364](#)  
[hastext\(\) \(pynlpl.formats.folia.Text method\), 390](#)  
[hastext\(\) \(pynlpl.formats.folia.TextContent method\), 498](#)  
[hastext\(\) \(pynlpl.formats.folia.TextMarkupCorrection method\), 843](#)  
[hastext\(\) \(pynlpl.formats.folia.TextMarkupError method\), 854](#)  
[hastext\(\) \(pynlpl.formats.folia.TextMarkupGap method\), 811](#)  
[hastext\(\) \(pynlpl.formats.folia.TextMarkupString method\), 822](#)  
[hastext\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 832](#)  
[hastext\(\) \(pynlpl.formats.folia.TimeSegment method\), 637](#)  
[hastext\(\) \(pynlpl.formats.folia.TimingLayer method\), 756](#)  
[hastext\(\) \(pynlpl.formats.folia.Whitespace method\), 403](#)  
[hastext\(\) \(pynlpl.formats.folia.Word method\), 418](#)  
[Head \(class in pynlpl.formats.folia\), 200](#)  
[head\(\) \(pynlpl.formats.folia.Dependency method\), 544](#)  
[head\(\) \(pynlpl.formats.folia.Division method\), 130](#)  
[Headspan \(class in pynlpl.formats.folia\), 785](#)  
[HiddenMarkovModel \(class in pynlpl.statistics\), 1063](#)  
[HillClimbingSearch \(class in pynlpl.search\), 1060](#)  
[histogram\(\) \(in module pynlpl.statistics\), 1064](#)
- I**
- [incorection\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 78](#)  
[incorection\(\) \(pynlpl.formats.folia.AbstractElement method\), 29](#)  
[incorection\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 55](#)  
[incorection\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 41](#)  
[incorection\(\) \(pynlpl.formats.folia.AbstractTextMarkup method\), 89](#)  
[incorection\(\) \(pynlpl.formats.folia.AbstractTokenAnnotation method\), 66](#)  
[incorection\(\) \(pynlpl.formats.folia.ActorFeature method\), 888](#)  
[incorection\(\) \(pynlpl.formats.folia.Alignment method\), 1012](#)  
[incorection\(\) \(pynlpl.formats.folia.AlignReference method\), 1023](#)  
[incorection\(\) \(pynlpl.formats.folia.Alternative method\), 923](#)  
[incorection\(\) \(pynlpl.formats.folia.AlternativeLayers method\), 934](#)  
[incorection\(\) \(pynlpl.formats.folia.BegindatetimeFeature method\), 899](#)  
[incorection\(\) \(pynlpl.formats.folia.Cell method\), 105](#)  
[incorection\(\) \(pynlpl.formats.folia.Chunk method\), 521](#)  
[incorection\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 650](#)  
[incorection\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 533](#)  
[incorection\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 662](#)  
[incorection\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 768](#)  
[incorection\(\) \(pynlpl.formats.folia.Correction method\), 947](#)  
[incorection\(\) \(pynlpl.formats.folia.Current method\), 956](#)  
[incorection\(\) \(pynlpl.formats.folia.Definition method\), 118](#)  
[incorection\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 674](#)  
[incorection\(\) \(pynlpl.formats.folia.Dependency method\), 545](#)  
[incorection\(\) \(pynlpl.formats.folia.DependencyDependent method\), 779](#)  
[incorection\(\) \(pynlpl.formats.folia.Description method\), 1034](#)  
[incorection\(\) \(pynlpl.formats.folia.Division method\), 131](#)  
[incorection\(\) \(pynlpl.formats.folia.DomainAnnotation method\), 431](#)  
[incorection\(\) \(pynlpl.formats.folia.EnddatetimeFeature method\), 910](#)  
[incorection\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 686](#)  
[incorection\(\) \(pynlpl.formats.folia.Entity method\), 556](#)  
[incorection\(\) \(pynlpl.formats.folia.Entry method\), 143](#)  
[incorection\(\) \(pynlpl.formats.folia.ErrorDetection method\), 967](#)  
[incorection\(\) \(pynlpl.formats.folia.Event method\), 156](#)  
[incorection\(\) \(pynlpl.formats.folia.Example method\), 169](#)

`incorection()` (`pynlpl.formats.folia.Feature` method), 865  
`incorection()` (`pynlpl.formats.folia.Figure` method), 183  
`incorection()` (`pynlpl.formats.folia.Gap` method), 194  
`incorection()` (`pynlpl.formats.folia.Head` method), 207  
`incorection()` (`pynlpl.formats.folia.Heads�an` method), 791  
`incorection()` (`pynlpl.formats.folia.LangAnnotation` method), 453  
`incorection()` (`pynlpl.formats.folia.LemmaAnnotation` method), 464  
`incorection()` (`pynlpl.formats.folia.Linebreak` method), 220  
`incorection()` (`pynlpl.formats.folia.List` method), 232  
`incorection()` (`pynlpl.formats.folia.ListItem` method), 245  
`incorection()` (`pynlpl.formats.folia.Metric` method), 1045  
`incorection()` (`pynlpl.formats.folia.New` method), 978  
`incorection()` (`pynlpl.formats.folia.Note` method), 258  
`incorection()` (`pynlpl.formats.folia.Observation` method), 568  
`incorection()` (`pynlpl.formats.folia.ObservationLayer` method), 697  
`incorection()` (`pynlpl.formats.folia.Original` method), 989  
`incorection()` (`pynlpl.formats.folia.Paragraph` method), 271  
`incorection()` (`pynlpl.formats.folia.Part` method), 284  
`incorection()` (`pynlpl.formats.folia.PhonContent` method), 509  
`incorection()` (`pynlpl.formats.folia.PosAnnotation` method), 442  
`incorection()` (`pynlpl.formats.folia.Predicate` method), 579  
`incorection()` (`pynlpl.formats.folia.Quote` method), 297  
`incorection()` (`pynlpl.formats.folia.Reference` method), 310  
`incorection()` (`pynlpl.formats.folia.Row` method), 323  
`incorection()` (`pynlpl.formats.folia.SemanticRole` method), 626  
`incorection()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 744  
`incorection()` (`pynlpl.formats.folia.SenseAnnotation` method), 475  
`incorection()` (`pynlpl.formats.folia.Sentence` method), 338  
`incorection()` (`pynlpl.formats.folia.Sentiment` method), 591  
`incorection()` (`pynlpl.formats.folia.SentimentLayer` method), 709  
`incorection()` (`pynlpl.formats.folia.Statement` method), 603  
`incorection()` (`pynlpl.formats.folia.StatementLayer` method), 721  
`incorection()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 486  
`incorection()` (`pynlpl.formats.folia.Suggestion` method), 1000  
`incorection()` (`pynlpl.formats.folia.SynsetFeature` method), 877  
`incorection()` (`pynlpl.formats.folia.SyntacticUnit` method), 614  
`incorection()` (`pynlpl.formats.folia.SyntaxLayer` method), 733  
`incorection()` (`pynlpl.formats.folia.Table` method), 351  
`incorection()` (`pynlpl.formats.folia.TableHead` method), 377  
`incorection()` (`pynlpl.formats.folia.Term` method), 364  
`incorection()` (`pynlpl.formats.folia.Text` method), 390  
`incorection()` (`pynlpl.formats.folia.TextContent` method), 498  
`incorection()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 843  
`incorection()` (`pynlpl.formats.folia.TextMarkupError` method), 854  
`incorection()` (`pynlpl.formats.folia.TextMarkupGap` method), 811  
`incorection()` (`pynlpl.formats.folia.TextMarkupString` method), 822  
`incorection()` (`pynlpl.formats.folia.TextMarkupStyle` method), 833  
`incorection()` (`pynlpl.formats.folia.TimeSegment` method), 638  
`incorection()` (`pynlpl.formats.folia.TimingLayer` method), 756  
`incorection()` (`pynlpl.formats.folia.Whitespace` method), 403  
`incorection()` (`pynlpl.formats.folia.Word` method), 418  
`information()` (`pynlpl.statistics.Distribution` method), 1062  
`initdoc()` (`pynlpl.formats.folia.Reader` method), 805  
`insert()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 78  
`insert()` (`pynlpl.formats.folia.AbstractElement` method), 29  
`insert()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 55  
`insert()` (`pynlpl.formats.folia.AbstractStructureElement` method), 42  
`insert()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 89  
`insert()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 66  
`insert()` (`pynlpl.formats.folia.ActorFeature` method), 888  
`insert()` (`pynlpl.formats.folia.Alignment` method), 1012  
`insert()` (`pynlpl.formats.folia.AlignReference` method), 1023  
`insert()` (`pynlpl.formats.folia.Alternative` method), 923  
`insert()` (`pynlpl.formats.folia.AlternativeLayers` method),

- 934
- `insert()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 899
- `insert()` (`pynlpl.formats.folia.Cell` method), 105
- `insert()` (`pynlpl.formats.folia.Chunk` method), 521
- `insert()` (`pynlpl.formats.folia.ChunkingLayer` method), 650
- `insert()` (`pynlpl.formats.folia.CoreferenceChain` method), 533
- `insert()` (`pynlpl.formats.folia.CoreferenceLayer` method), 662
- `insert()` (`pynlpl.formats.folia.CoreferenceLink` method), 768
- `insert()` (`pynlpl.formats.folia.Correction` method), 947
- `insert()` (`pynlpl.formats.folia.Current` method), 956
- `insert()` (`pynlpl.formats.folia.Definition` method), 118
- `insert()` (`pynlpl.formats.folia.DependenciesLayer` method), 674
- `insert()` (`pynlpl.formats.folia.Dependency` method), 545
- `insert()` (`pynlpl.formats.folia.DependencyDependent` method), 779
- `insert()` (`pynlpl.formats.folia.Description` method), 1034
- `insert()` (`pynlpl.formats.folia.Division` method), 131
- `insert()` (`pynlpl.formats.folia.DomainAnnotation` method), 432
- `insert()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 910
- `insert()` (`pynlpl.formats.folia.EntitiesLayer` method), 686
- `insert()` (`pynlpl.formats.folia.Entity` method), 556
- `insert()` (`pynlpl.formats.folia.Entry` method), 144
- `insert()` (`pynlpl.formats.folia.ErrorDetection` method), 968
- `insert()` (`pynlpl.formats.folia.Event` method), 157
- `insert()` (`pynlpl.formats.folia.Example` method), 170
- `insert()` (`pynlpl.formats.folia.Feature` method), 866
- `insert()` (`pynlpl.formats.folia.Figure` method), 183
- `insert()` (`pynlpl.formats.folia.Gap` method), 194
- `insert()` (`pynlpl.formats.folia.Head` method), 207
- `insert()` (`pynlpl.formats.folia.Headspan` method), 791
- `insert()` (`pynlpl.formats.folia.LangAnnotation` method), 454
- `insert()` (`pynlpl.formats.folia.LemmaAnnotation` method), 465
- `insert()` (`pynlpl.formats.folia.Linebreak` method), 220
- `insert()` (`pynlpl.formats.folia.List` method), 233
- `insert()` (`pynlpl.formats.folia.ListItem` method), 246
- `insert()` (`pynlpl.formats.folia.Metric` method), 1046
- `insert()` (`pynlpl.formats.folia.New` method), 979
- `insert()` (`pynlpl.formats.folia.Note` method), 259
- `insert()` (`pynlpl.formats.folia.Observation` method), 568
- `insert()` (`pynlpl.formats.folia.ObservationLayer` method), 697
- `insert()` (`pynlpl.formats.folia.Original` method), 990
- `insert()` (`pynlpl.formats.folia.Paragraph` method), 272
- `insert()` (`pynlpl.formats.folia.Part` method), 285
- `insert()` (`pynlpl.formats.folia.PhonContent` method), 509
- `insert()` (`pynlpl.formats.folia.PosAnnotation` method), 443
- `insert()` (`pynlpl.formats.folia.Predicate` method), 579
- `insert()` (`pynlpl.formats.folia.Quote` method), 298
- `insert()` (`pynlpl.formats.folia.Reference` method), 311
- `insert()` (`pynlpl.formats.folia.Row` method), 324
- `insert()` (`pynlpl.formats.folia.SemanticRole` method), 626
- `insert()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 744
- `insert()` (`pynlpl.formats.folia.SenseAnnotation` method), 476
- `insert()` (`pynlpl.formats.folia.Sentence` method), 338
- `insert()` (`pynlpl.formats.folia.Sentiment` method), 591
- `insert()` (`pynlpl.formats.folia.SentimentLayer` method), 709
- `insert()` (`pynlpl.formats.folia.Statement` method), 603
- `insert()` (`pynlpl.formats.folia.StatementLayer` method), 721
- `insert()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 487
- `insert()` (`pynlpl.formats.folia.Suggestion` method), 1001
- `insert()` (`pynlpl.formats.folia.SynsetFeature` method), 877
- `insert()` (`pynlpl.formats.folia.SyntacticUnit` method), 614
- `insert()` (`pynlpl.formats.folia.SyntaxLayer` method), 733
- `insert()` (`pynlpl.formats.folia.Table` method), 351
- `insert()` (`pynlpl.formats.folia.TableHead` method), 377
- `insert()` (`pynlpl.formats.folia.Term` method), 364
- `insert()` (`pynlpl.formats.folia.Text` method), 390
- `insert()` (`pynlpl.formats.folia.TextContent` method), 498
- `insert()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 843
- `insert()` (`pynlpl.formats.folia.TextMarkupError` method), 854
- `insert()` (`pynlpl.formats.folia.TextMarkupGap` method), 811
- `insert()` (`pynlpl.formats.folia.TextMarkupString` method), 822
- `insert()` (`pynlpl.formats.folia.TextMarkupStyle` method), 833
- `insert()` (`pynlpl.formats.folia.TimeSegment` method), 638
- `insert()` (`pynlpl.formats.folia.TimingLayer` method), 756
- `insert()` (`pynlpl.formats.folia.Whitespace` method), 403
- `insert()` (`pynlpl.formats.folia.Word` method), 418
- `insertword()` (`pynlpl.formats.folia.Sentence` method), 338
- `insertwordleft()` (`pynlpl.formats.folia.Sentence` method), 338
- `intersect()` (`pynlpl.formats.giza.GizaSentenceAlignment` method), 1053
- `IntersectionAlignment` (class in `pynlpl.formats.giza`), 1053
- `InvalidFeatureException`, 1053
- `InvalidTagException`, 1053



`is_end_of_sentence()` (in module `pynlpl.textprocessors`), 1066

`isstring()` (in module `pynlpl.common`), 3

`items()` (`pynlpl.datatypes.PatternMap` method), 5

`items()` (`pynlpl.datatypes.Trie` method), 6

`items()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 78

`items()` (`pynlpl.formats.folia.AbstractElement` method), 29

`items()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 55

`items()` (`pynlpl.formats.folia.AbstractStructureElement` method), 42

`items()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 89

`items()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 66

`items()` (`pynlpl.formats.folia.ActorFeature` method), 888

`items()` (`pynlpl.formats.folia.Alignment` method), 1012

`items()` (`pynlpl.formats.folia.AlignReference` method), 1023

`items()` (`pynlpl.formats.folia.Alternative` method), 923

`items()` (`pynlpl.formats.folia.AlternativeLayers` method), 934

`items()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 899

`items()` (`pynlpl.formats.folia.Cell` method), 105

`items()` (`pynlpl.formats.folia.Chunk` method), 521

`items()` (`pynlpl.formats.folia.ChunkingLayer` method), 650

`items()` (`pynlpl.formats.folia.CoreferenceChain` method), 533

`items()` (`pynlpl.formats.folia.CoreferenceLayer` method), 662

`items()` (`pynlpl.formats.folia.CoreferenceLink` method), 768

`items()` (`pynlpl.formats.folia.Correction` method), 947

`items()` (`pynlpl.formats.folia.Current` method), 956

`items()` (`pynlpl.formats.folia.Definition` method), 118

`items()` (`pynlpl.formats.folia.DependenciesLayer` method), 674

`items()` (`pynlpl.formats.folia.Dependency` method), 545

`items()` (`pynlpl.formats.folia.DependencyDependent` method), 779

`items()` (`pynlpl.formats.folia.Description` method), 1034

`items()` (`pynlpl.formats.folia.Division` method), 131

`items()` (`pynlpl.formats.folia.Document` method), 19

`items()` (`pynlpl.formats.folia.DomainAnnotation` method), 432

`items()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 910

`items()` (`pynlpl.formats.folia.EntitiesLayer` method), 686

`items()` (`pynlpl.formats.folia.Entity` method), 556

`items()` (`pynlpl.formats.folia.Entry` method), 144

`items()` (`pynlpl.formats.folia.ErrorDetection` method), 968

`items()` (`pynlpl.formats.folia.Event` method), 157

`items()` (`pynlpl.formats.folia.Example` method), 170

`items()` (`pynlpl.formats.folia.Feature` method), 866

`items()` (`pynlpl.formats.folia.Figure` method), 183

`items()` (`pynlpl.formats.folia.Gap` method), 194

`items()` (`pynlpl.formats.folia.Head` method), 207

`items()` (`pynlpl.formats.folia.Headspace` method), 791

`items()` (`pynlpl.formats.folia.LangAnnotation` method), 454

`items()` (`pynlpl.formats.folia.LemmaAnnotation` method), 465

`items()` (`pynlpl.formats.folia.Linebreak` method), 220

`items()` (`pynlpl.formats.folia.List` method), 233

`items()` (`pynlpl.formats.folia.ListItem` method), 246

`items()` (`pynlpl.formats.folia.Metric` method), 1046

`items()` (`pynlpl.formats.folia.New` method), 979

`items()` (`pynlpl.formats.folia.Note` method), 259

`items()` (`pynlpl.formats.folia.Observation` method), 568

`items()` (`pynlpl.formats.folia.ObservationLayer` method), 697

`items()` (`pynlpl.formats.folia.Original` method), 990

`items()` (`pynlpl.formats.folia.Paragraph` method), 272

`items()` (`pynlpl.formats.folia.Part` method), 285

`items()` (`pynlpl.formats.folia.PhonContent` method), 509

`items()` (`pynlpl.formats.folia.PosAnnotation` method), 443

`items()` (`pynlpl.formats.folia.Predicate` method), 579

`items()` (`pynlpl.formats.folia.Quote` method), 298

`items()` (`pynlpl.formats.folia.Reference` method), 311

`items()` (`pynlpl.formats.folia.Row` method), 324

`items()` (`pynlpl.formats.folia.SemanticRole` method), 626

`items()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 744

`items()` (`pynlpl.formats.folia.SenseAnnotation` method), 476

`items()` (`pynlpl.formats.folia.Sentence` method), 338

`items()` (`pynlpl.formats.folia.Sentiment` method), 591

`items()` (`pynlpl.formats.folia.SentimentLayer` method), 709

`items()` (`pynlpl.formats.folia.Statement` method), 603

`items()` (`pynlpl.formats.folia.StatementLayer` method), 721

`items()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 487

`items()` (`pynlpl.formats.folia.Suggestion` method), 1001

`items()` (`pynlpl.formats.folia.SynsetFeature` method), 877

`items()` (`pynlpl.formats.folia.SyntacticUnit` method), 614

`items()` (`pynlpl.formats.folia.SyntaxLayer` method), 733

`items()` (`pynlpl.formats.folia.Table` method), 351

`items()` (`pynlpl.formats.folia.TableHead` method), 377

`items()` (`pynlpl.formats.folia.Term` method), 364

`items()` (`pynlpl.formats.folia.Text` method), 390

`items()` (`pynlpl.formats.folia.TextContent` method), 498

- items() (pynlpl.formats.folia.TextMarkupCorrection method), 843
  - items() (pynlpl.formats.folia.TextMarkupError method), 854
  - items() (pynlpl.formats.folia.TextMarkupGap method), 811
  - items() (pynlpl.formats.folia.TextMarkupString method), 822
  - items() (pynlpl.formats.folia.TextMarkupStyle method), 833
  - items() (pynlpl.formats.folia.TimeSegment method), 638
  - items() (pynlpl.formats.folia.TimingLayer method), 756
  - items() (pynlpl.formats.folia.Whitespace method), 403
  - items() (pynlpl.formats.folia.Word method), 418
  - items() (pynlpl.statistics.Distribution method), 1062
  - items() (pynlpl.statistics.FrequencyList method), 1063
  - IterativeDeepening (class in pynlpl.search), 1060
  - iterbytes() (pynlpl.datatypes.Pattern method), 5
- ## J
- json() (pynlpl.formats.folia.AbstractAnnotationLayer method), 78
  - json() (pynlpl.formats.folia.AbstractElement method), 29
  - json() (pynlpl.formats.folia.AbstractSpanAnnotation method), 55
  - json() (pynlpl.formats.folia.AbstractStructureElement method), 42
  - json() (pynlpl.formats.folia.AbstractTextMarkup method), 89
  - json() (pynlpl.formats.folia.AbstractTokenAnnotation method), 66
  - json() (pynlpl.formats.folia.ActorFeature method), 888
  - json() (pynlpl.formats.folia.Alignment method), 1012
  - json() (pynlpl.formats.folia.AlignReference method), 1023
  - json() (pynlpl.formats.folia.Alternative method), 923
  - json() (pynlpl.formats.folia.AlternativeLayers method), 934
  - json() (pynlpl.formats.folia.BegindatetimeFeature method), 899
  - json() (pynlpl.formats.folia.Cell method), 105
  - json() (pynlpl.formats.folia.Chunk method), 521
  - json() (pynlpl.formats.folia.ChunkingLayer method), 650
  - json() (pynlpl.formats.folia.CoreferenceChain method), 533
  - json() (pynlpl.formats.folia.CoreferenceLayer method), 662
  - json() (pynlpl.formats.folia.CoreferenceLink method), 768
  - json() (pynlpl.formats.folia.Correction method), 947
  - json() (pynlpl.formats.folia.Current method), 956
  - json() (pynlpl.formats.folia.Definition method), 118
  - json() (pynlpl.formats.folia.DependenciesLayer method), 674
  - json() (pynlpl.formats.folia.Dependency method), 545
  - json() (pynlpl.formats.folia.DependencyDependent method), 779
  - json() (pynlpl.formats.folia.Description method), 1035
  - json() (pynlpl.formats.folia.Division method), 131
  - json() (pynlpl.formats.folia.Document method), 19
  - json() (pynlpl.formats.folia.DomainAnnotation method), 432
  - json() (pynlpl.formats.folia.EnddatetimeFeature method), 910
  - json() (pynlpl.formats.folia.EntitiesLayer method), 686
  - json() (pynlpl.formats.folia.Entity method), 556
  - json() (pynlpl.formats.folia.Entry method), 144
  - json() (pynlpl.formats.folia.ErrorDetection method), 968
  - json() (pynlpl.formats.folia.Event method), 157
  - json() (pynlpl.formats.folia.Example method), 170
  - json() (pynlpl.formats.folia.Feature method), 866
  - json() (pynlpl.formats.folia.Figure method), 183
  - json() (pynlpl.formats.folia.Gap method), 194
  - json() (pynlpl.formats.folia.Head method), 207
  - json() (pynlpl.formats.folia.Heads�an method), 791
  - json() (pynlpl.formats.folia.LangAnnotation method), 454
  - json() (pynlpl.formats.folia.LemmaAnnotation method), 465
  - json() (pynlpl.formats.folia.Linebreak method), 220
  - json() (pynlpl.formats.folia.List method), 233
  - json() (pynlpl.formats.folia.ListItem method), 246
  - json() (pynlpl.formats.folia.Metric method), 1046
  - json() (pynlpl.formats.folia.New method), 979
  - json() (pynlpl.formats.folia.Note method), 259
  - json() (pynlpl.formats.folia.Observation method), 568
  - json() (pynlpl.formats.folia.ObservationLayer method), 697
  - json() (pynlpl.formats.folia.Original method), 990
  - json() (pynlpl.formats.folia.Paragraph method), 272
  - json() (pynlpl.formats.folia.Part method), 285
  - json() (pynlpl.formats.folia.PhonContent method), 509
  - json() (pynlpl.formats.folia.PosAnnotation method), 443
  - json() (pynlpl.formats.folia.Predicate method), 579
  - json() (pynlpl.formats.folia.Quote method), 298
  - json() (pynlpl.formats.folia.Reference method), 311
  - json() (pynlpl.formats.folia.Row method), 324
  - json() (pynlpl.formats.folia.SemanticRole method), 626
  - json() (pynlpl.formats.folia.SemanticRolesLayer method), 744
  - json() (pynlpl.formats.folia.SenseAnnotation method), 476
  - json() (pynlpl.formats.folia.Sentence method), 338
  - json() (pynlpl.formats.folia.Sentiment method), 591
  - json() (pynlpl.formats.folia.SentimentLayer method), 709
  - json() (pynlpl.formats.folia.Statement method), 603
  - json() (pynlpl.formats.folia.StatementLayer method), 721

- `json()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 487
- `json()` (`pynlpl.formats.folia.Suggestion` method), 1001
- `json()` (`pynlpl.formats.folia.SynsetFeature` method), 877
- `json()` (`pynlpl.formats.folia.SyntacticUnit` method), 614
- `json()` (`pynlpl.formats.folia.SyntaxLayer` method), 733
- `json()` (`pynlpl.formats.folia.Table` method), 351
- `json()` (`pynlpl.formats.folia.TableHead` method), 377
- `json()` (`pynlpl.formats.folia.Term` method), 364
- `json()` (`pynlpl.formats.folia.Text` method), 390
- `json()` (`pynlpl.formats.folia.TextContent` method), 498
- `json()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 843
- `json()` (`pynlpl.formats.folia.TextMarkupError` method), 854
- `json()` (`pynlpl.formats.folia.TextMarkupGap` method), 811
- `json()` (`pynlpl.formats.folia.TextMarkupString` method), 822
- `json()` (`pynlpl.formats.folia.TextMarkupStyle` method), 833
- `json()` (`pynlpl.formats.folia.TimeSegment` method), 638
- `json()` (`pynlpl.formats.folia.TimingLayer` method), 756
- `json()` (`pynlpl.formats.folia.Whitespace` method), 403
- `json()` (`pynlpl.formats.folia.Word` method), 418
- `jsondeclarations()` (`pynlpl.formats.folia.Document` method), 19
- K**
- `keys()` (`pynlpl.statistics.Distribution` method), 1062
- L**
- `LABEL` (`pynlpl.formats.folia.ActorFeature` attribute), 885
- `LABEL` (`pynlpl.formats.folia.Alignment` attribute), 1009
- `LABEL` (`pynlpl.formats.folia.Alternative` attribute), 919
- `LABEL` (`pynlpl.formats.folia.AlternativeLayers` attribute), 931
- `LABEL` (`pynlpl.formats.folia.BegindatetimeFeature` attribute), 896
- `LABEL` (`pynlpl.formats.folia.Cell` attribute), 100
- `LABEL` (`pynlpl.formats.folia.Chunk` attribute), 518
- `LABEL` (`pynlpl.formats.folia.CoreferenceChain` attribute), 529
- `LABEL` (`pynlpl.formats.folia.CoreferenceLink` attribute), 764
- `LABEL` (`pynlpl.formats.folia.Correction` attribute), 944
- `LABEL` (`pynlpl.formats.folia.Definition` attribute), 113
- `LABEL` (`pynlpl.formats.folia.Dependency` attribute), 541
- `LABEL` (`pynlpl.formats.folia.DependencyDependent` attribute), 776
- `LABEL` (`pynlpl.formats.folia.Description` attribute), 1031
- `LABEL` (`pynlpl.formats.folia.Division` attribute), 126
- `LABEL` (`pynlpl.formats.folia.DomainAnnotation` attribute), 428
- `LABEL` (`pynlpl.formats.folia.EnddatetimeFeature` attribute), 907
- `LABEL` (`pynlpl.formats.folia.Entity` attribute), 553
- `LABEL` (`pynlpl.formats.folia.Entry` attribute), 139
- `LABEL` (`pynlpl.formats.folia.ErrorDetection` attribute), 964
- `LABEL` (`pynlpl.formats.folia.Event` attribute), 152
- `LABEL` (`pynlpl.formats.folia.Example` attribute), 165
- `LABEL` (`pynlpl.formats.folia.Feature` attribute), 862
- `LABEL` (`pynlpl.formats.folia.Figure` attribute), 178
- `LABEL` (`pynlpl.formats.folia.Gap` attribute), 191
- `LABEL` (`pynlpl.formats.folia.Head` attribute), 202
- `LABEL` (`pynlpl.formats.folia.Headspan` attribute), 788
- `LABEL` (`pynlpl.formats.folia.LangAnnotation` attribute), 450
- `LABEL` (`pynlpl.formats.folia.LemmaAnnotation` attribute), 461
- `LABEL` (`pynlpl.formats.folia.Linebreak` attribute), 215
- `LABEL` (`pynlpl.formats.folia.List` attribute), 228
- `LABEL` (`pynlpl.formats.folia.ListItem` attribute), 241
- `LABEL` (`pynlpl.formats.folia.Metric` attribute), 1042
- `LABEL` (`pynlpl.formats.folia.Note` attribute), 254
- `LABEL` (`pynlpl.formats.folia.Observation` attribute), 564
- `LABEL` (`pynlpl.formats.folia.Paragraph` attribute), 267
- `LABEL` (`pynlpl.formats.folia.Part` attribute), 280
- `LABEL` (`pynlpl.formats.folia.PhonContent` attribute), 505
- `LABEL` (`pynlpl.formats.folia.PosAnnotation` attribute), 439
- `LABEL` (`pynlpl.formats.folia.Predicate` attribute), 576
- `LABEL` (`pynlpl.formats.folia.Quote` attribute), 293
- `LABEL` (`pynlpl.formats.folia.Reference` attribute), 306
- `LABEL` (`pynlpl.formats.folia.Row` attribute), 319
- `LABEL` (`pynlpl.formats.folia.SemanticRole` attribute), 623
- `LABEL` (`pynlpl.formats.folia.SenseAnnotation` attribute), 472
- `LABEL` (`pynlpl.formats.folia.Sentence` attribute), 332
- `LABEL` (`pynlpl.formats.folia.Sentiment` attribute), 588
- `LABEL` (`pynlpl.formats.folia.Statement` attribute), 599
- `LABEL` (`pynlpl.formats.folia.SubjectivityAnnotation` attribute), 483
- `LABEL` (`pynlpl.formats.folia.SynsetFeature` attribute), 873
- `LABEL` (`pynlpl.formats.folia.SyntacticUnit` attribute), 611
- `LABEL` (`pynlpl.formats.folia.Table` attribute), 347
- `LABEL` (`pynlpl.formats.folia.TableHead` attribute), 373
- `LABEL` (`pynlpl.formats.folia.Term` attribute), 360
- `LABEL` (`pynlpl.formats.folia.Text` attribute), 386
- `LABEL` (`pynlpl.formats.folia.TextContent` attribute), 495
- `LABEL` (`pynlpl.formats.folia.TimeSegment` attribute), 634



- LABEL (pynlpl.formats.folia.Whitespace attribute), 399  
 LABEL (pynlpl.formats.folia.Word attribute), 412  
 LangAnnotation (class in pynlpl.formats.folia), 448  
 language() (pynlpl.formats.folia.Document method), 19  
 layers() (pynlpl.formats.folia.AbstractStructureElement method), 42  
 layers() (pynlpl.formats.folia.Cell method), 105  
 layers() (pynlpl.formats.folia.Definition method), 118  
 layers() (pynlpl.formats.folia.Division method), 131  
 layers() (pynlpl.formats.folia.Entry method), 144  
 layers() (pynlpl.formats.folia.Event method), 157  
 layers() (pynlpl.formats.folia.Example method), 170  
 layers() (pynlpl.formats.folia.Figure method), 183  
 layers() (pynlpl.formats.folia.Head method), 207  
 layers() (pynlpl.formats.folia.Linebreak method), 220  
 layers() (pynlpl.formats.folia.List method), 233  
 layers() (pynlpl.formats.folia.ListItem method), 246  
 layers() (pynlpl.formats.folia.Note method), 259  
 layers() (pynlpl.formats.folia.Paragraph method), 272  
 layers() (pynlpl.formats.folia.Part method), 285  
 layers() (pynlpl.formats.folia.Quote method), 298  
 layers() (pynlpl.formats.folia.Reference method), 311  
 layers() (pynlpl.formats.folia.Row method), 324  
 layers() (pynlpl.formats.folia.Sentence method), 338  
 layers() (pynlpl.formats.folia.Table method), 352  
 layers() (pynlpl.formats.folia.TableHead method), 378  
 layers() (pynlpl.formats.folia.Term method), 365  
 layers() (pynlpl.formats.folia.Text method), 391  
 layers() (pynlpl.formats.folia.Whitespace method), 404  
 layers() (pynlpl.formats.folia.Word method), 418  
 leaf() (pynlpl.datatypes.Tree method), 6  
 leaf() (pynlpl.datatypes.Trie method), 6  
 leftcontext() (pynlpl.formats.folia.AbstractAnnotationLayer method), 79  
 leftcontext() (pynlpl.formats.folia.AbstractElement method), 29  
 leftcontext() (pynlpl.formats.folia.AbstractSpanAnnotation method), 56  
 leftcontext() (pynlpl.formats.folia.AbstractStructureElement method), 42  
 leftcontext() (pynlpl.formats.folia.AbstractTextMarkup method), 89  
 leftcontext() (pynlpl.formats.folia.AbstractTokenAnnotation method), 67  
 leftcontext() (pynlpl.formats.folia.ActorFeature method), 888  
 leftcontext() (pynlpl.formats.folia.Alignment method), 1012  
 leftcontext() (pynlpl.formats.folia.AlignReference method), 1023  
 leftcontext() (pynlpl.formats.folia.Alternative method), 923  
 leftcontext() (pynlpl.formats.folia.AlternativeLayers method), 934  
 leftcontext() (pynlpl.formats.folia.BegindatetimeFeature method), 899  
 leftcontext() (pynlpl.formats.folia.Cell method), 105  
 leftcontext() (pynlpl.formats.folia.Chunk method), 522  
 leftcontext() (pynlpl.formats.folia.ChunkingLayer method), 650  
 leftcontext() (pynlpl.formats.folia.CoreferenceChain method), 533  
 leftcontext() (pynlpl.formats.folia.CoreferenceLayer method), 662  
 leftcontext() (pynlpl.formats.folia.CoreferenceLink method), 768  
 leftcontext() (pynlpl.formats.folia.Correction method), 947  
 leftcontext() (pynlpl.formats.folia.Current method), 957  
 leftcontext() (pynlpl.formats.folia.Definition method), 118  
 leftcontext() (pynlpl.formats.folia.DependenciesLayer method), 674  
 leftcontext() (pynlpl.formats.folia.Dependency method), 545  
 leftcontext() (pynlpl.formats.folia.DependencyDependent method), 780  
 leftcontext() (pynlpl.formats.folia.Description method), 1035  
 leftcontext() (pynlpl.formats.folia.Division method), 131  
 leftcontext() (pynlpl.formats.folia.DomainAnnotation method), 432  
 leftcontext() (pynlpl.formats.folia.EnddatetimeFeature method), 910  
 leftcontext() (pynlpl.formats.folia.EntitiesLayer method), 686  
 leftcontext() (pynlpl.formats.folia.Entity method), 556  
 leftcontext() (pynlpl.formats.folia.Entry method), 144  
 leftcontext() (pynlpl.formats.folia.ErrorDetection method), 968  
 leftcontext() (pynlpl.formats.folia.Event method), 157  
 leftcontext() (pynlpl.formats.folia.Example method), 170  
 leftcontext() (pynlpl.formats.folia.Feature method), 866  
 leftcontext() (pynlpl.formats.folia.Figure method), 183  
 leftcontext() (pynlpl.formats.folia.Gap method), 194  
 leftcontext() (pynlpl.formats.folia.Head method), 207  
 leftcontext() (pynlpl.formats.folia.Headspan method), 791  
 leftcontext() (pynlpl.formats.folia.LangAnnotation method), 454  
 leftcontext() (pynlpl.formats.folia.LemmaAnnotation method), 465  
 leftcontext() (pynlpl.formats.folia.Linebreak method), 220  
 leftcontext() (pynlpl.formats.folia.List method), 233  
 leftcontext() (pynlpl.formats.folia.ListItem method), 246  
 leftcontext() (pynlpl.formats.folia.Metric method), 1046  
 leftcontext() (pynlpl.formats.folia.New method), 979

`leftcontext()` (`pynlpl.formats.folia.Note` method), 259  
`leftcontext()` (`pynlpl.formats.folia.Observation` method), 568  
`leftcontext()` (`pynlpl.formats.folia.ObservationLayer` method), 698  
`leftcontext()` (`pynlpl.formats.folia.Original` method), 990  
`leftcontext()` (`pynlpl.formats.folia.Paragraph` method), 272  
`leftcontext()` (`pynlpl.formats.folia.Part` method), 285  
`leftcontext()` (`pynlpl.formats.folia.PhonContent` method), 509  
`leftcontext()` (`pynlpl.formats.folia.PosAnnotation` method), 443  
`leftcontext()` (`pynlpl.formats.folia.Predicate` method), 580  
`leftcontext()` (`pynlpl.formats.folia.Quote` method), 298  
`leftcontext()` (`pynlpl.formats.folia.Reference` method), 311  
`leftcontext()` (`pynlpl.formats.folia.Row` method), 324  
`leftcontext()` (`pynlpl.formats.folia.SemanticRole` method), 626  
`leftcontext()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 745  
`leftcontext()` (`pynlpl.formats.folia.SenseAnnotation` method), 476  
`leftcontext()` (`pynlpl.formats.folia.Sentence` method), 338  
`leftcontext()` (`pynlpl.formats.folia.Sentiment` method), 591  
`leftcontext()` (`pynlpl.formats.folia.SentimentLayer` method), 709  
`leftcontext()` (`pynlpl.formats.folia.Statement` method), 603  
`leftcontext()` (`pynlpl.formats.folia.StatementLayer` method), 721  
`leftcontext()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 487  
`leftcontext()` (`pynlpl.formats.folia.Suggestion` method), 1001  
`leftcontext()` (`pynlpl.formats.folia.SynsetFeature` method), 877  
`leftcontext()` (`pynlpl.formats.folia.SyntacticUnit` method), 615  
`leftcontext()` (`pynlpl.formats.folia.SyntaxLayer` method), 733  
`leftcontext()` (`pynlpl.formats.folia.Table` method), 352  
`leftcontext()` (`pynlpl.formats.folia.TableHead` method), 378  
`leftcontext()` (`pynlpl.formats.folia.Term` method), 365  
`leftcontext()` (`pynlpl.formats.folia.Text` method), 391  
`leftcontext()` (`pynlpl.formats.folia.TextContent` method), 498  
`leftcontext()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 843  
`leftcontext()` (`pynlpl.formats.folia.TextMarkupError` method), 854

`leftcontext()` (`pynlpl.formats.folia.TextMarkupGap` method), 811  
`leftcontext()` (`pynlpl.formats.folia.TextMarkupString` method), 822  
`leftcontext()` (`pynlpl.formats.folia.TextMarkupStyle` method), 833  
`leftcontext()` (`pynlpl.formats.folia.TimeSegment` method), 638  
`leftcontext()` (`pynlpl.formats.folia.TimingLayer` method), 756  
`leftcontext()` (`pynlpl.formats.folia.Whitespace` method), 404  
`leftcontext()` (`pynlpl.formats.folia.Word` method), 418  
`lemma()` (`pynlpl.formats.folia.Word` method), 418  
`LemmaAnnotation` (class in `pynlpl.formats.folia`), 459  
`levenshtein()` (in module `pynlpl.statistics`), 1064  
`license()` (`pynlpl.formats.folia.Document` method), 19  
`Linebreak` (class in `pynlpl.formats.folia`), 212  
`List` (class in `pynlpl.formats.folia`), 225  
`ListItem` (class in `pynlpl.formats.folia`), 238  
`LMClient` (class in `pynlpl.lm.client`), 1058  
`load()` (`pynlpl.formats.folia.Document` method), 19  
`load()` (`pynlpl.lm.lm.SimpleLanguageModel` method), 1057  
`load()` (`pynlpl.statistics.FrequencyList` method), 1063  
`log()` (in module `pynlpl.common`), 3  
`log2()` (in module `pynlpl.statistics`), 1064  
`logscore()` (`pynlpl.lm.srilm.SRILM` method), 1057

## M

`mae()` (in module `pynlpl.evaluation`), 11  
`mae()` (`pynlpl.evaluation.OrdinalEvaluation` method), 10  
`MarkovChain` (class in `pynlpl.statistics`), 1063  
`maxentropy()` (`pynlpl.statistics.Distribution` method), 1062  
`mean()` (in module `pynlpl.statistics`), 1064  
`median()` (in module `pynlpl.statistics`), 1064  
`mergewords()` (`pynlpl.formats.folia.Sentence` method), 338  
`Metric` (class in `pynlpl.formats.folia`), 1040  
`mode()` (in module `pynlpl.statistics`), 1064  
`mode()` (`pynlpl.statistics.Distribution` method), 1062  
`mode()` (`pynlpl.statistics.FrequencyList` method), 1063  
`morpheme()` (`pynlpl.formats.folia.Word` method), 418  
`morphemes()` (`pynlpl.formats.folia.Word` method), 418  
`MultiWindower` (class in `pynlpl.textprocessors`), 1066  
`MultiWordAlignment` (class in `pynlpl.formats.giza`), 1053

## N

`New` (class in `pynlpl.formats.folia`), 973  
`new()` (`pynlpl.formats.folia.Correction` method), 947  
`next()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 79  
`next()` (`pynlpl.formats.folia.AbstractElement` method), 29

- next() (pynlpl.formats.folia.AbstractSpanAnnotation method), 56
- next() (pynlpl.formats.folia.AbstractStructureElement method), 42
- next() (pynlpl.formats.folia.AbstractTextMarkup method), 89
- next() (pynlpl.formats.folia.AbstractTokenAnnotation method), 67
- next() (pynlpl.formats.folia.ActorFeature method), 888
- next() (pynlpl.formats.folia.Alignment method), 1012
- next() (pynlpl.formats.folia.AlignReference method), 1023
- next() (pynlpl.formats.folia.Alternative method), 923
- next() (pynlpl.formats.folia.AlternativeLayers method), 935
- next() (pynlpl.formats.folia.BegindatetimeFeature method), 899
- next() (pynlpl.formats.folia.Cell method), 105
- next() (pynlpl.formats.folia.Chunk method), 522
- next() (pynlpl.formats.folia.ChunkingLayer method), 650
- next() (pynlpl.formats.folia.CoreferenceChain method), 533
- next() (pynlpl.formats.folia.CoreferenceLayer method), 662
- next() (pynlpl.formats.folia.CoreferenceLink method), 768
- next() (pynlpl.formats.folia.Correction method), 947
- next() (pynlpl.formats.folia.Current method), 957
- next() (pynlpl.formats.folia.Definition method), 118
- next() (pynlpl.formats.folia.DependenciesLayer method), 674
- next() (pynlpl.formats.folia.Dependency method), 545
- next() (pynlpl.formats.folia.DependencyDependent method), 780
- next() (pynlpl.formats.folia.Description method), 1035
- next() (pynlpl.formats.folia.Division method), 131
- next() (pynlpl.formats.folia.DomainAnnotation method), 432
- next() (pynlpl.formats.folia.EnddatetimeFeature method), 910
- next() (pynlpl.formats.folia.EntitiesLayer method), 686
- next() (pynlpl.formats.folia.Entity method), 556
- next() (pynlpl.formats.folia.Entry method), 144
- next() (pynlpl.formats.folia.ErrorDetection method), 968
- next() (pynlpl.formats.folia.Event method), 157
- next() (pynlpl.formats.folia.Example method), 170
- next() (pynlpl.formats.folia.Feature method), 866
- next() (pynlpl.formats.folia.Figure method), 183
- next() (pynlpl.formats.folia.Gap method), 194
- next() (pynlpl.formats.folia.Head method), 207
- next() (pynlpl.formats.folia.Heads�an method), 791
- next() (pynlpl.formats.folia.LangAnnotation method), 454
- next() (pynlpl.formats.folia.LemmaAnnotation method), 465
- next() (pynlpl.formats.folia.Linebreak method), 220
- next() (pynlpl.formats.folia.List method), 233
- next() (pynlpl.formats.folia.ListItem method), 246
- next() (pynlpl.formats.folia.Metric method), 1046
- next() (pynlpl.formats.folia.New method), 979
- next() (pynlpl.formats.folia.Note method), 259
- next() (pynlpl.formats.folia.Observation method), 568
- next() (pynlpl.formats.folia.ObservationLayer method), 698
- next() (pynlpl.formats.folia.Original method), 990
- next() (pynlpl.formats.folia.Paragraph method), 272
- next() (pynlpl.formats.folia.Part method), 285
- next() (pynlpl.formats.folia.PhonContent method), 509
- next() (pynlpl.formats.folia.PosAnnotation method), 443
- next() (pynlpl.formats.folia.Predicate method), 580
- next() (pynlpl.formats.folia.Quote method), 298
- next() (pynlpl.formats.folia.Reference method), 311
- next() (pynlpl.formats.folia.Row method), 324
- next() (pynlpl.formats.folia.SemanticRole method), 626
- next() (pynlpl.formats.folia.SemanticRolesLayer method), 745
- next() (pynlpl.formats.folia.SenseAnnotation method), 476
- next() (pynlpl.formats.folia.Sentence method), 339
- next() (pynlpl.formats.folia.Sentiment method), 591
- next() (pynlpl.formats.folia.SentimentLayer method), 709
- next() (pynlpl.formats.folia.Statement method), 603
- next() (pynlpl.formats.folia.StatementLayer method), 721
- next() (pynlpl.formats.folia.SubjectivityAnnotation method), 487
- next() (pynlpl.formats.folia.Suggestion method), 1001
- next() (pynlpl.formats.folia.SynsetFeature method), 877
- next() (pynlpl.formats.folia.SyntacticUnit method), 615
- next() (pynlpl.formats.folia.SyntaxLayer method), 733
- next() (pynlpl.formats.folia.Table method), 352
- next() (pynlpl.formats.folia.TableHead method), 378
- next() (pynlpl.formats.folia.Term method), 365
- next() (pynlpl.formats.folia.Text method), 391
- next() (pynlpl.formats.folia.TextContent method), 498
- next() (pynlpl.formats.folia.TextMarkupCorrection method), 844
- next() (pynlpl.formats.folia.TextMarkupError method), 854
- next() (pynlpl.formats.folia.TextMarkupGap method), 812
- next() (pynlpl.formats.folia.TextMarkupString method), 822
- next() (pynlpl.formats.folia.TextMarkupStyle method), 833
- next() (pynlpl.formats.folia.TimeSegment method), 638
- next() (pynlpl.formats.folia.TimingLayer method), 756
- next() (pynlpl.formats.folia.Whitespace method), 404
- next() (pynlpl.formats.folia.Word method), 419

next() (pynlpl.formats.taggedata.Taggedata method), 1055

normalize() (in module pynlpl.statistics), 1064

Note (class in pynlpl.formats.folia), 251

ns() (in module pynlpl.formats.sonar), 1055

## O

Observation (class in pynlpl.formats.folia), 562

ObservationLayer (class in pynlpl.formats.folia), 691

OCCURRENCES (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74

OCCURRENCES (pynlpl.formats.folia.AbstractElement attribute), 26

OCCURRENCES (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52

OCCURRENCES (pynlpl.formats.folia.AbstractStructureElement attribute), 37

OCCURRENCES (pynlpl.formats.folia.AbstractTextMarkup attribute), 86

OCCURRENCES (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63

OCCURRENCES (pynlpl.formats.folia.ActorFeature attribute), 885

OCCURRENCES (pynlpl.formats.folia.Alignment attribute), 1009

OCCURRENCES (pynlpl.formats.folia.AlignReference attribute), 1020

OCCURRENCES (pynlpl.formats.folia.Alternative attribute), 919

OCCURRENCES (pynlpl.formats.folia.AlternativeLayers attribute), 931

OCCURRENCES (pynlpl.formats.folia.BegindatetimeFeature attribute), 896

OCCURRENCES (pynlpl.formats.folia.Cell attribute), 100

OCCURRENCES (pynlpl.formats.folia.Chunk attribute), 518

OCCURRENCES (pynlpl.formats.folia.ChunkingLayer attribute), 646

OCCURRENCES (pynlpl.formats.folia.CoreferenceChain attribute), 529

OCCURRENCES (pynlpl.formats.folia.CoreferenceLayer attribute), 658

OCCURRENCES (pynlpl.formats.folia.CoreferenceLink attribute), 764

OCCURRENCES (pynlpl.formats.folia.Correction attribute), 944

OCCURRENCES (pynlpl.formats.folia.Current attribute), 953

OCCURRENCES (pynlpl.formats.folia.Definition attribute), 113

OCCURRENCES (pynlpl.formats.folia.DependenciesLayer attribute), 670

OCCURRENCES (pynlpl.formats.folia.Dependency attribute), 541

OCCURRENCES (pynlpl.formats.folia.DependencyDependent attribute), 776

OCCURRENCES (pynlpl.formats.folia.Description attribute), 1031

OCCURRENCES (pynlpl.formats.folia.Division attribute), 126

OCCURRENCES (pynlpl.formats.folia.DomainAnnotation attribute), 428

OCCURRENCES (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

OCCURRENCES (pynlpl.formats.folia.EntitiesLayer attribute), 682

OCCURRENCES (pynlpl.formats.folia.Entity attribute), 553

OCCURRENCES (pynlpl.formats.folia.Entry attribute), 139

OCCURRENCES (pynlpl.formats.folia.ErrorDetection attribute), 964

OCCURRENCES (pynlpl.formats.folia.Event attribute), 152

OCCURRENCES (pynlpl.formats.folia.Example attribute), 165

OCCURRENCES (pynlpl.formats.folia.Feature attribute), 862

OCCURRENCES (pynlpl.formats.folia.Figure attribute), 178

OCCURRENCES (pynlpl.formats.folia.Gap attribute), 191

OCCURRENCES (pynlpl.formats.folia.Head attribute), 202

OCCURRENCES (pynlpl.formats.folia.Headspan attribute), 788

OCCURRENCES (pynlpl.formats.folia.LangAnnotation attribute), 450

OCCURRENCES (pynlpl.formats.folia.LemmaAnnotation attribute), 461

OCCURRENCES (pynlpl.formats.folia.Linebreak attribute), 215

OCCURRENCES (pynlpl.formats.folia.List attribute), 228

OCCURRENCES (pynlpl.formats.folia.ListItem attribute), 241

OCCURRENCES (pynlpl.formats.folia.Metric attribute), 1042

OCCURRENCES (pynlpl.formats.folia.New attribute), 975

OCCURRENCES (pynlpl.formats.folia.Note attribute), 254

OCCURRENCES (pynlpl.formats.folia.Observation attribute), 564

OCCURRENCES (pynlpl.formats.folia.ObservationLayer attribute), 693

- OCCURRENCES (pynlpl.formats.folia.Original attribute), 986
- OCCURRENCES (pynlpl.formats.folia.Paragraph attribute), 267
- OCCURRENCES (pynlpl.formats.folia.Part attribute), 280
- OCCURRENCES (pynlpl.formats.folia.PhonContent attribute), 505
- OCCURRENCES (pynlpl.formats.folia.PosAnnotation attribute), 439
- OCCURRENCES (pynlpl.formats.folia.Predicate attribute), 576
- OCCURRENCES (pynlpl.formats.folia.Quote attribute), 293
- OCCURRENCES (pynlpl.formats.folia.Reference attribute), 306
- OCCURRENCES (pynlpl.formats.folia.Row attribute), 319
- OCCURRENCES (pynlpl.formats.folia.SemanticRole attribute), 623
- OCCURRENCES (pynlpl.formats.folia.SemanticRolesLayer attribute), 740
- OCCURRENCES (pynlpl.formats.folia.SenseAnnotation attribute), 472
- OCCURRENCES (pynlpl.formats.folia.Sentence attribute), 332
- OCCURRENCES (pynlpl.formats.folia.Sentiment attribute), 588
- OCCURRENCES (pynlpl.formats.folia.SentimentLayer attribute), 705
- OCCURRENCES (pynlpl.formats.folia.Statement attribute), 599
- OCCURRENCES (pynlpl.formats.folia.StatementLayer attribute), 717
- OCCURRENCES (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- OCCURRENCES (pynlpl.formats.folia.Suggestion attribute), 997
- OCCURRENCES (pynlpl.formats.folia.SynsetFeature attribute), 873
- OCCURRENCES (pynlpl.formats.folia.SyntacticUnit attribute), 611
- OCCURRENCES (pynlpl.formats.folia.SyntaxLayer attribute), 729
- OCCURRENCES (pynlpl.formats.folia.Table attribute), 347
- OCCURRENCES (pynlpl.formats.folia.TableHead attribute), 373
- OCCURRENCES (pynlpl.formats.folia.Term attribute), 360
- OCCURRENCES (pynlpl.formats.folia.Text attribute), 386
- OCCURRENCES (pynlpl.formats.folia.TextContent attribute), 495
- OCCURRENCES (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- OCCURRENCES (pynlpl.formats.folia.TextMarkupError attribute), 851
- OCCURRENCES (pynlpl.formats.folia.TextMarkupGap attribute), 808
- OCCURRENCES (pynlpl.formats.folia.TextMarkupString attribute), 819
- OCCURRENCES (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- OCCURRENCES (pynlpl.formats.folia.TimeSegment attribute), 634
- OCCURRENCES (pynlpl.formats.folia.TimingLayer attribute), 752
- OCCURRENCES (pynlpl.formats.folia.Whitespace attribute), 399
- OCCURRENCES (pynlpl.formats.folia.Word attribute), 413
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractElement attribute), 26
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractStructureElement attribute), 37
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractTextMarkup attribute), 86
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.ActorFeature attribute), 885
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.Alignment attribute), 1009
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AlignReference attribute), 1020
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.Alternative attribute), 919
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.AlternativeLayers attribute), 931
- OCCURRENCES\_PER\_SET (pynlpl.formats.folia.BegindatetimeFeature attribute), 896



OCCURRENCES_PER_SET (pynlpl.formats.folia.Cell attribute), 100	964	OCCURRENCES_PER_SET (pynlpl.formats.folia.Event attribute), 152
OCCURRENCES_PER_SET (pynlpl.formats.folia.Chunk attribute), 518		OCCURRENCES_PER_SET (pynlpl.formats.folia.Example attribute), 165
OCCURRENCES_PER_SET (pynlpl.formats.folia.ChunkingLayer attribute), 646		OCCURRENCES_PER_SET (pynlpl.formats.folia.Feature attribute), 862
OCCURRENCES_PER_SET (pynlpl.formats.folia.CoreferenceChain attribute), 529	at-	OCCURRENCES_PER_SET (pynlpl.formats.folia.Figure attribute), 178
OCCURRENCES_PER_SET (pynlpl.formats.folia.CoreferenceLayer attribute), 658	at-	OCCURRENCES_PER_SET (pynlpl.formats.folia.Gap attribute), 191
OCCURRENCES_PER_SET (pynlpl.formats.folia.CoreferenceLink attribute), 764	at-	OCCURRENCES_PER_SET (pynlpl.formats.folia.Head attribute), 202
OCCURRENCES_PER_SET (pynlpl.formats.folia.Correction attribute), 944		OCCURRENCES_PER_SET (pynlpl.formats.folia.Headspan attribute), 788
OCCURRENCES_PER_SET (pynlpl.formats.folia.Current attribute), 953		OCCURRENCES_PER_SET (pynlpl.formats.folia.LangAnnotation attribute), 450
OCCURRENCES_PER_SET (pynlpl.formats.folia.Definition attribute), 113		OCCURRENCES_PER_SET (pynlpl.formats.folia.LemmaAnnotation attribute), 461
OCCURRENCES_PER_SET (pynlpl.formats.folia.DependenciesLayer attribute), 670		OCCURRENCES_PER_SET (pynlpl.formats.folia.Linebreak attribute), 215
OCCURRENCES_PER_SET (pynlpl.formats.folia.Dependency attribute), 541		OCCURRENCES_PER_SET (pynlpl.formats.folia.List attribute), 228
OCCURRENCES_PER_SET (pynlpl.formats.folia.DependencyDependent attribute), 776		OCCURRENCES_PER_SET (pynlpl.formats.folia.ListItem attribute), 241
OCCURRENCES_PER_SET (pynlpl.formats.folia.Description attribute), 1031		OCCURRENCES_PER_SET (pynlpl.formats.folia.Metric attribute), 1042
OCCURRENCES_PER_SET (pynlpl.formats.folia.Division attribute), 126		OCCURRENCES_PER_SET (pynlpl.formats.folia.New attribute), 975
OCCURRENCES_PER_SET (pynlpl.formats.folia.DomainAnnotation attribute), 428		OCCURRENCES_PER_SET (pynlpl.formats.folia.Note attribute), 254
OCCURRENCES_PER_SET (pynlpl.formats.folia.EnddatetimeFeature attribute), 907		OCCURRENCES_PER_SET (pynlpl.formats.folia.Observation attribute), 564
OCCURRENCES_PER_SET (pynlpl.formats.folia.EntitiesLayer attribute), 682		OCCURRENCES_PER_SET (pynlpl.formats.folia.ObservationLayer attribute), 694
OCCURRENCES_PER_SET (pynlpl.formats.folia.Entity attribute), 553		OCCURRENCES_PER_SET (pynlpl.formats.folia.Original attribute), 986
OCCURRENCES_PER_SET (pynlpl.formats.folia.Entry attribute), 139		OCCURRENCES_PER_SET (pynlpl.formats.folia.Paragraph attribute), 267
OCCURRENCES_PER_SET (pynlpl.formats.folia.ErrorDetection attribute),		OCCURRENCES_PER_SET (pynlpl.formats.folia.Part attribute), 280
		OCCURRENCES_PER_SET (pynlpl.formats.folia.PhonContent attribute), 505

OCCURRENCES_PER_SET (pynlpl.formats.folia.PosAnnotation attribute), 439	OCCURRENCES_PER_SET (pynlpl.formats.folia.TableHead attribute), 373
OCCURRENCES_PER_SET (pynlpl.formats.folia.Predicate attribute), 576	OCCURRENCES_PER_SET (pynlpl.formats.folia.Term attribute), 360
OCCURRENCES_PER_SET (pynlpl.formats.folia.Quote attribute), 293	OCCURRENCES_PER_SET (pynlpl.formats.folia.Text attribute), 386
OCCURRENCES_PER_SET (pynlpl.formats.folia.Reference attribute), 306	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextContent attribute), 495
OCCURRENCES_PER_SET (pynlpl.formats.folia.Row attribute), 319	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
OCCURRENCES_PER_SET (pynlpl.formats.folia.SemanticRole attribute), 623	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextMarkupError attribute), 851
OCCURRENCES_PER_SET (pynlpl.formats.folia.SemanticRolesLayer attribute), 741	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextMarkupGap attribute), 808
OCCURRENCES_PER_SET (pynlpl.formats.folia.SenseAnnotation attribute), 472	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextMarkupString attribute), 819
OCCURRENCES_PER_SET (pynlpl.formats.folia.Sentence attribute), 332	OCCURRENCES_PER_SET (pynlpl.formats.folia.TextMarkupStyle attribute), 830
OCCURRENCES_PER_SET (pynlpl.formats.folia.Sentiment attribute), 588	OCCURRENCES_PER_SET (pynlpl.formats.folia.TimeSegment attribute), 634
OCCURRENCES_PER_SET (pynlpl.formats.folia.SentimentLayer attribute), 705	OCCURRENCES_PER_SET (pynlpl.formats.folia.TimingLayer attribute), 752
OCCURRENCES_PER_SET (pynlpl.formats.folia.Statement attribute), 599	OCCURRENCES_PER_SET (pynlpl.formats.folia.Whitespace attribute), 399
OCCURRENCES_PER_SET (pynlpl.formats.folia.StatementLayer attribute), 717	OCCURRENCES_PER_SET (pynlpl.formats.folia.Word attribute), 413
OCCURRENCES_PER_SET (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74
OCCURRENCES_PER_SET (pynlpl.formats.folia.Suggestion attribute), 997	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractElement attribute), 26
OCCURRENCES_PER_SET (pynlpl.formats.folia.SynsetFeature attribute), 873	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52
OCCURRENCES_PER_SET (pynlpl.formats.folia.SyntacticUnit attribute), 611	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractStructureElement attribute), 37
OCCURRENCES_PER_SET (pynlpl.formats.folia.SyntaxLayer attribute), 729	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractTextMarkup attribute), 86
OCCURRENCES_PER_SET (pynlpl.formats.folia.Table attribute), 347	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63
	OPTIONAL_ATTRIBS (pynlpl.formats.folia.ActorFeature attribute), 885
	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Alignment attribute), 1009
	OPTIONAL_ATTRIBS (pynlpl.formats.folia.AlignReference attribute), 1020

OPTIONAL_ATTRIBS (pynlpl.formats.folia.Alternative attribute), 919	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Gap attribute), 191
OPTIONAL_ATTRIBS (pynlpl.formats.folia.AlternativeLayer attribute), 931	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Head attribute), 202
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Begindatetime attribute), 896	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Headspan attribute), 788
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Cell attribute), 100	OPTIONAL_ATTRIBS (pynlpl.formats.folia.LangAnnotation attribute), 450
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Chunk attribute), 518	OPTIONAL_ATTRIBS (pynlpl.formats.folia.LemmaAnnotation attribute), 461
OPTIONAL_ATTRIBS (pynlpl.formats.folia.ChunkingLayer attribute), 646	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Linebreak attribute), 215
OPTIONAL_ATTRIBS (pynlpl.formats.folia.CoreferenceChain attribute), 529	OPTIONAL_ATTRIBS (pynlpl.formats.folia.List attribute), 228
OPTIONAL_ATTRIBS (pynlpl.formats.folia.CoreferenceLayer attribute), 658	OPTIONAL_ATTRIBS (pynlpl.formats.folia.ListItem attribute), 241
OPTIONAL_ATTRIBS (pynlpl.formats.folia.CoreferenceLink attribute), 764	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Metric attribute), 1042
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Correction attribute), 944	OPTIONAL_ATTRIBS (pynlpl.formats.folia.New attribute), 975
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Current attribute), 953	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Note attribute), 254
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Definition attribute), 113	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Observation attribute), 564
OPTIONAL_ATTRIBS (pynlpl.formats.folia.DependenciesLayer attribute), 670	OPTIONAL_ATTRIBS (pynlpl.formats.folia.ObservationLayer attribute), 694
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Dependency attribute), 541	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Original attribute), 986
OPTIONAL_ATTRIBS (pynlpl.formats.folia.DependencyDependency attribute), 776	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Paragraph attribute), 267
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Description attribute), 1031	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Part attribute), 280
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Division attribute), 126	OPTIONAL_ATTRIBS (pynlpl.formats.folia.PhonContent attribute), 506
OPTIONAL_ATTRIBS (pynlpl.formats.folia.DomainAnnotation attribute), 428	OPTIONAL_ATTRIBS (pynlpl.formats.folia.PosAnnotation attribute), 439
OPTIONAL_ATTRIBS (pynlpl.formats.folia.EnddatetimeFeature attribute), 907	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Predicate attribute), 576
OPTIONAL_ATTRIBS (pynlpl.formats.folia.EntitiesLayer attribute), 682	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Quote attribute), 293
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Entity attribute), 553	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Reference attribute), 306
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Entry attribute), 139	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Row attribute), 319
OPTIONAL_ATTRIBS (pynlpl.formats.folia.ErrorDetection attribute), 964	OPTIONAL_ATTRIBS (pynlpl.formats.folia.SemanticRole attribute), 623
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Event attribute), 152	OPTIONAL_ATTRIBS (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Example attribute), 165	OPTIONAL_ATTRIBS (pynlpl.formats.folia.SenseAnnotation attribute), 472
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Feature attribute), 862	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Sentence attribute), 332
OPTIONAL_ATTRIBS (pynlpl.formats.folia.Figure attribute), 178	OPTIONAL_ATTRIBS (pynlpl.formats.folia.Sentiment attribute), 588



OPTIONAL\_ATTRIBS (pynlpl.formats.folia.SentimentLayer attribute), 705  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Statement attribute), 599  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.StatementLayer attribute), 717  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Suggestion attribute), 997  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.SynsetFeature attribute), 873  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.SyntacticUnit attribute), 611  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.SyntaxLayer attribute), 729  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Table attribute), 347  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TableHead attribute), 373  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Term attribute), 360  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Text attribute), 386  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextContent attribute), 495  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextMarkupCorrection attribute), 840  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextMarkupError attribute), 851  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextMarkupGap attribute), 808  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextMarkupString attribute), 819  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TextMarkupStyle attribute), 830  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TimeSegment attribute), 634  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.TimingLayer attribute), 752  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Whitespace attribute), 399  
 OPTIONAL\_ATTRIBS (pynlpl.formats.folia.Word attribute), 413  
 OrdinalEvaluation (class in pynlpl.evaluation), 10  
 Original (class in pynlpl.formats.folia), 984  
 original() (pynlpl.formats.folia.Correction method), 947  
 originaltext() (pynlpl.formats.folia.AbstractAnnotationLayer method), 79  
 originaltext() (pynlpl.formats.folia.AbstractElement method), 29  
 originaltext() (pynlpl.formats.folia.AbstractSpanAnnotation method), 56  
 originaltext() (pynlpl.formats.folia.AbstractStructureElement method), 42  
 originaltext() (pynlpl.formats.folia.AbstractTextMarkup method), 90  
 originaltext() (pynlpl.formats.folia.AbstractTokenAnnotation method), 67  
 originaltext() (pynlpl.formats.folia.ActorFeature method), 888  
 originaltext() (pynlpl.formats.folia.Alignment method), 1012  
 originaltext() (pynlpl.formats.folia.AlignReference method), 1024  
 originaltext() (pynlpl.formats.folia.Alternative method), 924  
 originaltext() (pynlpl.formats.folia.AlternativeLayers method), 935  
 originaltext() (pynlpl.formats.folia.BegindatetimeFeature method), 899  
 originaltext() (pynlpl.formats.folia.Cell method), 105  
 originaltext() (pynlpl.formats.folia.Chunk method), 522  
 originaltext() (pynlpl.formats.folia.ChunkingLayer method), 651  
 originaltext() (pynlpl.formats.folia.CoreferenceChain method), 533  
 originaltext() (pynlpl.formats.folia.CoreferenceLayer method), 662  
 originaltext() (pynlpl.formats.folia.CoreferenceLink method), 768  
 originaltext() (pynlpl.formats.folia.Correction method), 948  
 originaltext() (pynlpl.formats.folia.Current method), 957  
 originaltext() (pynlpl.formats.folia.Definition method), 118  
 originaltext() (pynlpl.formats.folia.DependenciesLayer method), 674  
 originaltext() (pynlpl.formats.folia.Dependency method), 545  
 originaltext() (pynlpl.formats.folia.DependencyDependent method), 780  
 originaltext() (pynlpl.formats.folia.Description method), 1035  
 originaltext() (pynlpl.formats.folia.Division method), 131  
 originaltext() (pynlpl.formats.folia.DomainAnnotation method), 432  
 originaltext() (pynlpl.formats.folia.EnddatetimeFeature method), 910  
 originaltext() (pynlpl.formats.folia.EntitiesLayer method), 686  
 originaltext() (pynlpl.formats.folia.Entity method), 557  
 originaltext() (pynlpl.formats.folia.Entry method), 144  
 originaltext() (pynlpl.formats.folia.ErrorDetection method), 968  
 originaltext() (pynlpl.formats.folia.Event method), 157  
 originaltext() (pynlpl.formats.folia.Example method), 170

`originaltext()` (`pynlpl.formats.folia.Feature` method), 866  
`originaltext()` (`pynlpl.formats.folia.Figure` method), 183  
`originaltext()` (`pynlpl.formats.folia.Gap` method), 194  
`originaltext()` (`pynlpl.formats.folia.Head` method), 207  
`originaltext()` (`pynlpl.formats.folia.Headspan` method), 792  
`originaltext()` (`pynlpl.formats.folia.LangAnnotation` method), 454  
`originaltext()` (`pynlpl.formats.folia.LemmaAnnotation` method), 465  
`originaltext()` (`pynlpl.formats.folia.Linebreak` method), 220  
`originaltext()` (`pynlpl.formats.folia.List` method), 233  
`originaltext()` (`pynlpl.formats.folia.ListItem` method), 246  
`originaltext()` (`pynlpl.formats.folia.Metric` method), 1046  
`originaltext()` (`pynlpl.formats.folia.New` method), 979  
`originaltext()` (`pynlpl.formats.folia.Note` method), 259  
`originaltext()` (`pynlpl.formats.folia.Observation` method), 568  
`originaltext()` (`pynlpl.formats.folia.ObservationLayer` method), 698  
`originaltext()` (`pynlpl.formats.folia.Original` method), 990  
`originaltext()` (`pynlpl.formats.folia.Paragraph` method), 272  
`originaltext()` (`pynlpl.formats.folia.Part` method), 285  
`originaltext()` (`pynlpl.formats.folia.PhonContent` method), 509  
`originaltext()` (`pynlpl.formats.folia.PosAnnotation` method), 443  
`originaltext()` (`pynlpl.formats.folia.Predicate` method), 580  
`originaltext()` (`pynlpl.formats.folia.Quote` method), 298  
`originaltext()` (`pynlpl.formats.folia.Reference` method), 311  
`originaltext()` (`pynlpl.formats.folia.Row` method), 324  
`originaltext()` (`pynlpl.formats.folia.SemanticRole` method), 627  
`originaltext()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 745  
`originaltext()` (`pynlpl.formats.folia.SenseAnnotation` method), 476  
`originaltext()` (`pynlpl.formats.folia.Sentence` method), 339  
`originaltext()` (`pynlpl.formats.folia.Sentiment` method), 592  
`originaltext()` (`pynlpl.formats.folia.SentimentLayer` method), 710  
`originaltext()` (`pynlpl.formats.folia.Statement` method), 603  
`originaltext()` (`pynlpl.formats.folia.StatementLayer` method), 721  
`originaltext()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 487  
`originaltext()` (`pynlpl.formats.folia.Suggestion` method), 1001  
`originaltext()` (`pynlpl.formats.folia.SynsetFeature` method), 877  
`originaltext()` (`pynlpl.formats.folia.SyntacticUnit` method), 615  
`originaltext()` (`pynlpl.formats.folia.SyntaxLayer` method), 733  
`originaltext()` (`pynlpl.formats.folia.Table` method), 352  
`originaltext()` (`pynlpl.formats.folia.TableHead` method), 378  
`originaltext()` (`pynlpl.formats.folia.Term` method), 365  
`originaltext()` (`pynlpl.formats.folia.Text` method), 391  
`originaltext()` (`pynlpl.formats.folia.TextContent` method), 499  
`originaltext()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 844  
`originaltext()` (`pynlpl.formats.folia.TextMarkupError` method), 854  
`originaltext()` (`pynlpl.formats.folia.TextMarkupGap` method), 812  
`originaltext()` (`pynlpl.formats.folia.TextMarkupString` method), 822  
`originaltext()` (`pynlpl.formats.folia.TextMarkupStyle` method), 833  
`originaltext()` (`pynlpl.formats.folia.TimeSegment` method), 638  
`originaltext()` (`pynlpl.formats.folia.TimingLayer` method), 757  
`originaltext()` (`pynlpl.formats.folia.Whitespace` method), 404  
`originaltext()` (`pynlpl.formats.folia.Word` method), 419  
`output()` (`pynlpl.statistics.Distribution` method), 1062  
`output()` (`pynlpl.statistics.FrequencyList` method), 1063  
`outputmetrics()` (`pynlpl.evaluation.ClassEvaluation` method), 10

## P

`p()` (`pynlpl.statistics.FrequencyList` method), 1063  
`p()` (`pynlpl.statistics.MarkovChain` method), 1063  
`Paragraph` (class in `pynlpl.formats.folia`), 264  
`paragraph()` (`pynlpl.formats.folia.Sentence` method), 339  
`paragraph()` (`pynlpl.formats.folia.Word` method), 419  
`paragraphs()` (`pynlpl.formats.folia.AbstractStructureElement` method), 42  
`paragraphs()` (`pynlpl.formats.folia.Cell` method), 105  
`paragraphs()` (`pynlpl.formats.folia.Definition` method), 118  
`paragraphs()` (`pynlpl.formats.folia.Division` method), 131  
`paragraphs()` (`pynlpl.formats.folia.Document` method), 19  
`paragraphs()` (`pynlpl.formats.folia.Entry` method), 144  
`paragraphs()` (`pynlpl.formats.folia.Event` method), 157  
`paragraphs()` (`pynlpl.formats.folia.Example` method), 170  
`paragraphs()` (`pynlpl.formats.folia.Figure` method), 183  
`paragraphs()` (`pynlpl.formats.folia.Head` method), 207

- paragraphs() (pynlpl.formats.folia.Linebreak method), 220
- paragraphs() (pynlpl.formats.folia.List method), 233
- paragraphs() (pynlpl.formats.folia.ListItem method), 246
- paragraphs() (pynlpl.formats.folia.Note method), 259
- paragraphs() (pynlpl.formats.folia.Paragraph method), 272
- paragraphs() (pynlpl.formats.folia.Part method), 285
- paragraphs() (pynlpl.formats.folia.Quote method), 298
- paragraphs() (pynlpl.formats.folia.Reference method), 311
- paragraphs() (pynlpl.formats.folia.Row method), 324
- paragraphs() (pynlpl.formats.folia.Sentence method), 339
- paragraphs() (pynlpl.formats.folia.Table method), 352
- paragraphs() (pynlpl.formats.folia.TableHead method), 378
- paragraphs() (pynlpl.formats.folia.Term method), 365
- paragraphs() (pynlpl.formats.folia.Text method), 391
- paragraphs() (pynlpl.formats.folia.Whitespace method), 404
- paragraphs() (pynlpl.formats.folia.Word method), 419
- paragraphs() (pynlpl.formats.sonar.CorpusDocument method), 1054
- paragraphs() (pynlpl.formats.sonar.CorpusDocumentX method), 1054
- ParamSearch (class in pynlpl.evaluation), 10
- parse() (pynlpl.formats.fql.Query method), 803
- parse\_cgn\_postag() (in module pynlpl.formats.cgn), 1053
- parseAlignment() (in module pynlpl.formats.giza), 1054
- parseDistribution() (pynlpl.formats.timbl.TimblOutput method), 1055
- parsemetadata() (pynlpl.formats.folia.Document method), 20
- parsesubmetadata() (pynlpl.formats.folia.Document method), 20
- parsexml() (pynlpl.formats.folia.AbstractAnnotationLayer class method), 79
- parsexml() (pynlpl.formats.folia.AbstractElement class method), 29
- parsexml() (pynlpl.formats.folia.AbstractSpanAnnotation class method), 56
- parsexml() (pynlpl.formats.folia.AbstractStructureElement class method), 42
- parsexml() (pynlpl.formats.folia.AbstractTextMarkup class method), 90
- parsexml() (pynlpl.formats.folia.AbstractTokenAnnotation class method), 67
- parsexml() (pynlpl.formats.folia.ActorFeature class method), 889
- parsexml() (pynlpl.formats.folia.Alignment class method), 1012
- parsexml() (pynlpl.formats.folia.AlignReference class method), 1024
- parsexml() (pynlpl.formats.folia.Alternative class method), 924
- parsexml() (pynlpl.formats.folia.AlternativeLayers class method), 935
- parsexml() (pynlpl.formats.folia.BegindatetimeFeature class method), 900
- parsexml() (pynlpl.formats.folia.Cell class method), 105
- parsexml() (pynlpl.formats.folia.Chunk class method), 522
- parsexml() (pynlpl.formats.folia.ChunkingLayer class method), 651
- parsexml() (pynlpl.formats.folia.CoreferenceChain class method), 533
- parsexml() (pynlpl.formats.folia.CoreferenceLayer class method), 663
- parsexml() (pynlpl.formats.folia.CoreferenceLink class method), 768
- parsexml() (pynlpl.formats.folia.Correction class method), 948
- parsexml() (pynlpl.formats.folia.Current class method), 957
- parsexml() (pynlpl.formats.folia.Definition class method), 118
- parsexml() (pynlpl.formats.folia.DependenciesLayer class method), 674
- parsexml() (pynlpl.formats.folia.Dependency class method), 545
- parsexml() (pynlpl.formats.folia.DependencyDependent class method), 780
- parsexml() (pynlpl.formats.folia.Description class method), 1035
- parsexml() (pynlpl.formats.folia.Division class method), 131
- parsexml() (pynlpl.formats.folia.Document method), 20
- parsexml() (pynlpl.formats.folia.DomainAnnotation class method), 432
- parsexml() (pynlpl.formats.folia.EnddatetimeFeature class method), 911
- parsexml() (pynlpl.formats.folia.EntitiesLayer class method), 686
- parsexml() (pynlpl.formats.folia.Entity class method), 557
- parsexml() (pynlpl.formats.folia.Entry class method), 144
- parsexml() (pynlpl.formats.folia.ErrorDetection class method), 968
- parsexml() (pynlpl.formats.folia.Event class method), 157
- parsexml() (pynlpl.formats.folia.Example class method), 170
- parsexml() (pynlpl.formats.folia.Feature class method), 866
- parsexml() (pynlpl.formats.folia.Figure class method), 183
- parsexml() (pynlpl.formats.folia.Gap class method), 195
- parsexml() (pynlpl.formats.folia.Head class method), 207
- parsexml() (pynlpl.formats.folia.Headspan class method),

- [792](#)
- `parsexml()` (`pynlpl.formats.folia.LangAnnotation` class method), [454](#)
- `parsexml()` (`pynlpl.formats.folia.LemmaAnnotation` class method), [465](#)
- `parsexml()` (`pynlpl.formats.folia.Linebreak` class method), [220](#)
- `parsexml()` (`pynlpl.formats.folia.List` class method), [233](#)
- `parsexml()` (`pynlpl.formats.folia.ListItem` class method), [246](#)
- `parsexml()` (`pynlpl.formats.folia.Metric` class method), [1046](#)
- `parsexml()` (`pynlpl.formats.folia.New` class method), [979](#)
- `parsexml()` (`pynlpl.formats.folia.Note` class method), [259](#)
- `parsexml()` (`pynlpl.formats.folia.Observation` class method), [568](#)
- `parsexml()` (`pynlpl.formats.folia.ObservationLayer` class method), [698](#)
- `parsexml()` (`pynlpl.formats.folia.Original` class method), [990](#)
- `parsexml()` (`pynlpl.formats.folia.Paragraph` class method), [272](#)
- `parsexml()` (`pynlpl.formats.folia.Part` class method), [285](#)
- `parsexml()` (`pynlpl.formats.folia.PhonContent` class method), [509](#)
- `parsexml()` (`pynlpl.formats.folia.PosAnnotation` class method), [443](#)
- `parsexml()` (`pynlpl.formats.folia.Predicate` class method), [580](#)
- `parsexml()` (`pynlpl.formats.folia.Quote` class method), [298](#)
- `parsexml()` (`pynlpl.formats.folia.Reference` class method), [311](#)
- `parsexml()` (`pynlpl.formats.folia.Row` class method), [324](#)
- `parsexml()` (`pynlpl.formats.folia.SemanticRole` class method), [627](#)
- `parsexml()` (`pynlpl.formats.folia.SemanticRolesLayer` class method), [745](#)
- `parsexml()` (`pynlpl.formats.folia.SenseAnnotation` class method), [476](#)
- `parsexml()` (`pynlpl.formats.folia.Sentence` class method), [339](#)
- `parsexml()` (`pynlpl.formats.folia.Sentiment` class method), [592](#)
- `parsexml()` (`pynlpl.formats.folia.SentimentLayer` class method), [710](#)
- `parsexml()` (`pynlpl.formats.folia.Statement` class method), [603](#)
- `parsexml()` (`pynlpl.formats.folia.StatementLayer` class method), [721](#)
- `parsexml()` (`pynlpl.formats.folia.SubjectivityAnnotation` class method), [487](#)
- `parsexml()` (`pynlpl.formats.folia.Suggestion` class method), [1001](#)
- `parsexml()` (`pynlpl.formats.folia.SynsetFeature` class method), [877](#)
- `parsexml()` (`pynlpl.formats.folia.SyntacticUnit` class method), [615](#)
- `parsexml()` (`pynlpl.formats.folia.SyntaxLayer` class method), [733](#)
- `parsexml()` (`pynlpl.formats.folia.Table` class method), [352](#)
- `parsexml()` (`pynlpl.formats.folia.TableHead` class method), [378](#)
- `parsexml()` (`pynlpl.formats.folia.Term` class method), [365](#)
- `parsexml()` (`pynlpl.formats.folia.Text` class method), [391](#)
- `parsexml()` (`pynlpl.formats.folia.TextContent` class method), [499](#)
- `parsexml()` (`pynlpl.formats.folia.TextMarkupCorrection` class method), [844](#)
- `parsexml()` (`pynlpl.formats.folia.TextMarkupError` class method), [855](#)
- `parsexml()` (`pynlpl.formats.folia.TextMarkupGap` class method), [812](#)
- `parsexml()` (`pynlpl.formats.folia.TextMarkupString` class method), [823](#)
- `parsexml()` (`pynlpl.formats.folia.TextMarkupStyle` class method), [833](#)
- `parsexml()` (`pynlpl.formats.folia.TimeSegment` class method), [638](#)
- `parsexml()` (`pynlpl.formats.folia.TimingLayer` class method), [757](#)
- `parsexml()` (`pynlpl.formats.folia.Whitespace` class method), [404](#)
- `parsexml()` (`pynlpl.formats.folia.Word` class method), [419](#)
- `parsexmldeclarations()` (`pynlpl.formats.folia.Document` method), [20](#)
- `Part` (class in `pynlpl.formats.folia`), [277](#)
- `path()` (`pynlpl.datatypes.Trie` method), [7](#)
- `path()` (`pynlpl.search.AbstractSearchState` method), [1060](#)
- `pathcost()` (`pynlpl.search.AbstractSearchState` method), [1060](#)
- `Pattern` (class in `pynlpl.datatypes`), [5](#)
- `PatternMap` (class in `pynlpl.datatypes`), [5](#)
- `PatternSet` (class in `pynlpl.datatypes`), [5](#)
- `pendingvalidation()` (`pynlpl.formats.folia.Document` method), [20](#)
- `perplexity()` (`pynlpl.statistics.Distribution` method), [1062](#)
- `phon()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), [79](#)
- `phon()` (`pynlpl.formats.folia.AbstractElement` method), [30](#)
- `phon()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), [56](#)
- `phon()` (`pynlpl.formats.folia.AbstractStructureElement` method), [42](#)
- `phon()` (`pynlpl.formats.folia.AbstractTextMarkup` method), [90](#)
- `phon()` (`pynlpl.formats.folia.AbstractTokenAnnotation`

- method), 67
- phon() (pynlpl.formats.folia.ActorFeature method), 889
- phon() (pynlpl.formats.folia.Alignment method), 1013
- phon() (pynlpl.formats.folia.AlignReference method), 1024
- phon() (pynlpl.formats.folia.Alternative method), 924
- phon() (pynlpl.formats.folia.AlternativeLayers method), 935
- phon() (pynlpl.formats.folia.BegindatetimeFeature method), 900
- phon() (pynlpl.formats.folia.Cell method), 106
- phon() (pynlpl.formats.folia.Chunk method), 522
- phon() (pynlpl.formats.folia.ChunkingLayer method), 651
- phon() (pynlpl.formats.folia.CoreferenceChain method), 534
- phon() (pynlpl.formats.folia.CoreferenceLayer method), 663
- phon() (pynlpl.formats.folia.CoreferenceLink method), 768
- phon() (pynlpl.formats.folia.Correction method), 948
- phon() (pynlpl.formats.folia.Current method), 957
- phon() (pynlpl.formats.folia.Definition method), 119
- phon() (pynlpl.formats.folia.DependenciesLayer method), 674
- phon() (pynlpl.formats.folia.Dependency method), 545
- phon() (pynlpl.formats.folia.DependencyDependent method), 780
- phon() (pynlpl.formats.folia.Description method), 1035
- phon() (pynlpl.formats.folia.Division method), 131
- phon() (pynlpl.formats.folia.DomainAnnotation method), 432
- phon() (pynlpl.formats.folia.EnddatetimeFeature method), 911
- phon() (pynlpl.formats.folia.EntitiesLayer method), 686
- phon() (pynlpl.formats.folia.Entity method), 557
- phon() (pynlpl.formats.folia.Entry method), 144
- phon() (pynlpl.formats.folia.ErrorDetection method), 968
- phon() (pynlpl.formats.folia.Event method), 157
- phon() (pynlpl.formats.folia.Example method), 170
- phon() (pynlpl.formats.folia.Feature method), 866
- phon() (pynlpl.formats.folia.Figure method), 184
- phon() (pynlpl.formats.folia.Gap method), 195
- phon() (pynlpl.formats.folia.Head method), 208
- phon() (pynlpl.formats.folia.Headspan method), 792
- phon() (pynlpl.formats.folia.LangAnnotation method), 454
- phon() (pynlpl.formats.folia.LemmaAnnotation method), 465
- phon() (pynlpl.formats.folia.Linebreak method), 220
- phon() (pynlpl.formats.folia.List method), 233
- phon() (pynlpl.formats.folia.ListItem method), 246
- phon() (pynlpl.formats.folia.Metric method), 1046
- phon() (pynlpl.formats.folia.New method), 979
- phon() (pynlpl.formats.folia.Note method), 259
- phon() (pynlpl.formats.folia.Observation method), 569
- phon() (pynlpl.formats.folia.ObservationLayer method), 698
- phon() (pynlpl.formats.folia.Original method), 990
- phon() (pynlpl.formats.folia.Paragraph method), 272
- phon() (pynlpl.formats.folia.Part method), 285
- phon() (pynlpl.formats.folia.PhonContent method), 509
- phon() (pynlpl.formats.folia.PosAnnotation method), 443
- phon() (pynlpl.formats.folia.Predicate method), 580
- phon() (pynlpl.formats.folia.Quote method), 298
- phon() (pynlpl.formats.folia.Reference method), 311
- phon() (pynlpl.formats.folia.Row method), 324
- phon() (pynlpl.formats.folia.SemanticRole method), 627
- phon() (pynlpl.formats.folia.SemanticRolesLayer method), 745
- phon() (pynlpl.formats.folia.SenseAnnotation method), 476
- phon() (pynlpl.formats.folia.Sentence method), 339
- phon() (pynlpl.formats.folia.Sentiment method), 592
- phon() (pynlpl.formats.folia.SentimentLayer method), 710
- phon() (pynlpl.formats.folia.Statement method), 603
- phon() (pynlpl.formats.folia.StatementLayer method), 721
- phon() (pynlpl.formats.folia.SubjectivityAnnotation method), 487
- phon() (pynlpl.formats.folia.Suggestion method), 1001
- phon() (pynlpl.formats.folia.SynsetFeature method), 877
- phon() (pynlpl.formats.folia.SyntacticUnit method), 615
- phon() (pynlpl.formats.folia.SyntaxLayer method), 733
- phon() (pynlpl.formats.folia.Table method), 352
- phon() (pynlpl.formats.folia.TableHead method), 378
- phon() (pynlpl.formats.folia.Term method), 365
- phon() (pynlpl.formats.folia.Text method), 391
- phon() (pynlpl.formats.folia.TextContent method), 499
- phon() (pynlpl.formats.folia.TextMarkupCorrection method), 844
- phon() (pynlpl.formats.folia.TextMarkupError method), 855
- phon() (pynlpl.formats.folia.TextMarkupGap method), 812
- phon() (pynlpl.formats.folia.TextMarkupString method), 823
- phon() (pynlpl.formats.folia.TextMarkupStyle method), 833
- phon() (pynlpl.formats.folia.TimeSegment method), 638
- phon() (pynlpl.formats.folia.TimingLayer method), 757
- phon() (pynlpl.formats.folia.Whitespace method), 404
- phon() (pynlpl.formats.folia.Word method), 419
- PHONCONTAINER (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 74
- PHONCONTAINER (pynlpl.formats.folia.AbstractElement attribute), 26



PHONCONTAINER (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52

PHONCONTAINER (pynlpl.formats.folia.AbstractStructureAnnotation attribute), 37

PHONCONTAINER (pynlpl.formats.folia.AbstractTextMarkup attribute), 86

PHONCONTAINER (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63

PHONCONTAINER (pynlpl.formats.folia.ActorFeature attribute), 885

PHONCONTAINER (pynlpl.formats.folia.Alignment attribute), 1009

PHONCONTAINER (pynlpl.formats.folia.AlignReference attribute), 1020

PHONCONTAINER (pynlpl.formats.folia.Alternative attribute), 919

PHONCONTAINER (pynlpl.formats.folia.AlternativeLayer attribute), 931

PHONCONTAINER (pynlpl.formats.folia.BegindatetimeFeature attribute), 896

PHONCONTAINER (pynlpl.formats.folia.Cell attribute), 100

PHONCONTAINER (pynlpl.formats.folia.Chunk attribute), 518

PHONCONTAINER (pynlpl.formats.folia.ChunkingLayer attribute), 646

PHONCONTAINER (pynlpl.formats.folia.CoreferenceChain attribute), 529

PHONCONTAINER (pynlpl.formats.folia.CoreferenceLayer attribute), 658

PHONCONTAINER (pynlpl.formats.folia.CoreferenceLink attribute), 764

PHONCONTAINER (pynlpl.formats.folia.Correction attribute), 944

PHONCONTAINER (pynlpl.formats.folia.Current attribute), 953

PHONCONTAINER (pynlpl.formats.folia.Definition attribute), 113

PHONCONTAINER (pynlpl.formats.folia.DependenciesLayer attribute), 670

PHONCONTAINER (pynlpl.formats.folia.Dependency attribute), 541

PHONCONTAINER (pynlpl.formats.folia.DependencyDependency attribute), 776

PHONCONTAINER (pynlpl.formats.folia.Description attribute), 1031

PHONCONTAINER (pynlpl.formats.folia.Division attribute), 126

PHONCONTAINER (pynlpl.formats.folia.DomainAnnotation attribute), 428

PHONCONTAINER (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

PHONCONTAINER (pynlpl.formats.folia.EntitiesLayer attribute), 682

PHONCONTAINER (pynlpl.formats.folia.Entity attribute), 553

PHONCONTAINER (pynlpl.formats.folia.Entry attribute), 139

PHONCONTAINER (pynlpl.formats.folia.ErrorDetection attribute), 964

PHONCONTAINER (pynlpl.formats.folia.Event attribute), 152

PHONCONTAINER (pynlpl.formats.folia.Example attribute), 165

PHONCONTAINER (pynlpl.formats.folia.Feature attribute), 862

PHONCONTAINER (pynlpl.formats.folia.Figure attribute), 178

PHONCONTAINER (pynlpl.formats.folia.Gap attribute), 191

PHONCONTAINER (pynlpl.formats.folia.Head attribute), 202

PHONCONTAINER (pynlpl.formats.folia.Headspan attribute), 788

PHONCONTAINER (pynlpl.formats.folia.LangAnnotation attribute), 450

PHONCONTAINER (pynlpl.formats.folia.LemmaAnnotation attribute), 461

PHONCONTAINER (pynlpl.formats.folia.Linebreak attribute), 215

PHONCONTAINER (pynlpl.formats.folia.List attribute), 228

PHONCONTAINER (pynlpl.formats.folia.ListItem attribute), 241

PHONCONTAINER (pynlpl.formats.folia.Metric attribute), 1042

PHONCONTAINER (pynlpl.formats.folia.New attribute), 975

PHONCONTAINER (pynlpl.formats.folia.Note attribute), 254

PHONCONTAINER (pynlpl.formats.folia.Observation attribute), 564

PHONCONTAINER (pynlpl.formats.folia.ObservationLayer attribute), 694

PHONCONTAINER (pynlpl.formats.folia.Original attribute), 986

PHONCONTAINER (pynlpl.formats.folia.Paragraph attribute), 267

PHONCONTAINER (pynlpl.formats.folia.Part attribute), 280

PHONCONTAINER (pynlpl.formats.folia.PhonContent attribute), 506

PHONCONTAINER (pynlpl.formats.folia.PosAnnotation attribute), 439

PHONCONTAINER (pynlpl.formats.folia.Predicate attribute), 576

PHONCONTAINER (pynlpl.formats.folia.Quote attribute), 293

- PHONCONTAINER (pynlpl.formats.folia.Reference attribute), 306
- PHONCONTAINER (pynlpl.formats.folia.Row attribute), 319
- PHONCONTAINER (pynlpl.formats.folia.SemanticRole attribute), 623
- PHONCONTAINER (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- PHONCONTAINER (pynlpl.formats.folia.SenseAnnotation attribute), 472
- PHONCONTAINER (pynlpl.formats.folia.Sentence attribute), 333
- PHONCONTAINER (pynlpl.formats.folia.Sentiment attribute), 588
- PHONCONTAINER (pynlpl.formats.folia.SentimentLayer attribute), 705
- PHONCONTAINER (pynlpl.formats.folia.Statement attribute), 599
- PHONCONTAINER (pynlpl.formats.folia.StatementLayer attribute), 717
- PHONCONTAINER (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483
- PHONCONTAINER (pynlpl.formats.folia.Suggestion attribute), 997
- PHONCONTAINER (pynlpl.formats.folia.SynsetFeature attribute), 873
- PHONCONTAINER (pynlpl.formats.folia.SyntacticUnit attribute), 611
- PHONCONTAINER (pynlpl.formats.folia.SyntaxLayer attribute), 729
- PHONCONTAINER (pynlpl.formats.folia.Table attribute), 347
- PHONCONTAINER (pynlpl.formats.folia.TableHead attribute), 373
- PHONCONTAINER (pynlpl.formats.folia.Term attribute), 360
- PHONCONTAINER (pynlpl.formats.folia.Text attribute), 386
- PHONCONTAINER (pynlpl.formats.folia.TextContent attribute), 495
- PHONCONTAINER (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- PHONCONTAINER (pynlpl.formats.folia.TextMarkupError attribute), 851
- PHONCONTAINER (pynlpl.formats.folia.TextMarkupGap attribute), 808
- PHONCONTAINER (pynlpl.formats.folia.TextMarkupString attribute), 819
- PHONCONTAINER (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- PHONCONTAINER (pynlpl.formats.folia.TimeSegment attribute), 634
- PHONCONTAINER (pynlpl.formats.folia.TimingLayer attribute), 752
- PHONCONTAINER (pynlpl.formats.folia.Whitespace attribute), 399
- PHONCONTAINER (pynlpl.formats.folia.Word attribute), 413
- PhonContent (class in pynlpl.formats.folia), 503
- phoncontent() (pynlpl.formats.folia.AbstractAnnotationLayer method), 80
- phoncontent() (pynlpl.formats.folia.AbstractElement method), 30
- phoncontent() (pynlpl.formats.folia.AbstractSpanAnnotation method), 57
- phoncontent() (pynlpl.formats.folia.AbstractStructureElement method), 43
- phoncontent() (pynlpl.formats.folia.AbstractTextMarkup method), 91
- phoncontent() (pynlpl.formats.folia.AbstractTokenAnnotation method), 68
- phoncontent() (pynlpl.formats.folia.ActorFeature method), 889
- phoncontent() (pynlpl.formats.folia.Alignment method), 1013
- phoncontent() (pynlpl.formats.folia.AlignReference method), 1025
- phoncontent() (pynlpl.formats.folia.Alternative method), 925
- phoncontent() (pynlpl.formats.folia.AlternativeLayers method), 936
- phoncontent() (pynlpl.formats.folia.BegindatetimeFeature method), 900
- phoncontent() (pynlpl.formats.folia.Cell method), 106
- phoncontent() (pynlpl.formats.folia.Chunk method), 523
- phoncontent() (pynlpl.formats.folia.ChunkingLayer method), 652
- phoncontent() (pynlpl.formats.folia.CoreferenceChain method), 534
- phoncontent() (pynlpl.formats.folia.CoreferenceLayer method), 663
- phoncontent() (pynlpl.formats.folia.CoreferenceLink method), 769
- phoncontent() (pynlpl.formats.folia.Correction method), 948
- phoncontent() (pynlpl.formats.folia.Current method), 958
- phoncontent() (pynlpl.formats.folia.Definition method), 119
- phoncontent() (pynlpl.formats.folia.DependenciesLayer method), 675
- phoncontent() (pynlpl.formats.folia.Dependency method), 546
- phoncontent() (pynlpl.formats.folia.DependencyDependent method), 781
- phoncontent() (pynlpl.formats.folia.Description method), 1036
- phoncontent() (pynlpl.formats.folia.Division method), 132

phoncontent() (pynlpl.formats.folia.DomainAnnotation method), 433

phoncontent() (pynlpl.formats.folia.EnddatetimeFeature method), 911

phoncontent() (pynlpl.formats.folia.EntitiesLayer method), 687

phoncontent() (pynlpl.formats.folia.Entity method), 558

phoncontent() (pynlpl.formats.folia.Entry method), 145

phoncontent() (pynlpl.formats.folia.ErrorDetection method), 969

phoncontent() (pynlpl.formats.folia.Event method), 158

phoncontent() (pynlpl.formats.folia.Example method), 171

phoncontent() (pynlpl.formats.folia.Feature method), 867

phoncontent() (pynlpl.formats.folia.Figure method), 184

phoncontent() (pynlpl.formats.folia.Gap method), 195

phoncontent() (pynlpl.formats.folia.Head method), 208

phoncontent() (pynlpl.formats.folia.Headspeed method), 793

phoncontent() (pynlpl.formats.folia.LangAnnotation method), 455

phoncontent() (pynlpl.formats.folia.LemmaAnnotation method), 466

phoncontent() (pynlpl.formats.folia.Linebreak method), 221

phoncontent() (pynlpl.formats.folia.List method), 234

phoncontent() (pynlpl.formats.folia.ListItem method), 247

phoncontent() (pynlpl.formats.folia.Metric method), 1047

phoncontent() (pynlpl.formats.folia.New method), 980

phoncontent() (pynlpl.formats.folia.Note method), 260

phoncontent() (pynlpl.formats.folia.Observation method), 569

phoncontent() (pynlpl.formats.folia.ObservationLayer method), 699

phoncontent() (pynlpl.formats.folia.Original method), 991

phoncontent() (pynlpl.formats.folia.Paragraph method), 273

phoncontent() (pynlpl.formats.folia.Part method), 286

phoncontent() (pynlpl.formats.folia.PhonContent method), 509

phoncontent() (pynlpl.formats.folia.PosAnnotation method), 444

phoncontent() (pynlpl.formats.folia.Predicate method), 581

phoncontent() (pynlpl.formats.folia.Quote method), 299

phoncontent() (pynlpl.formats.folia.Reference method), 312

phoncontent() (pynlpl.formats.folia.Row method), 325

phoncontent() (pynlpl.formats.folia.SemanticRole method), 628

phoncontent() (pynlpl.formats.folia.SemanticRolesLayer method), 746

phoncontent() (pynlpl.formats.folia.SenseAnnotation method), 477

phoncontent() (pynlpl.formats.folia.Sentence method), 340

phoncontent() (pynlpl.formats.folia.Sentiment method), 593

phoncontent() (pynlpl.formats.folia.SentimentLayer method), 710

phoncontent() (pynlpl.formats.folia.Statement method), 604

phoncontent() (pynlpl.formats.folia.StatementLayer method), 722

phoncontent() (pynlpl.formats.folia.SubjectivityAnnotation method), 488

phoncontent() (pynlpl.formats.folia.Suggestion method), 1002

phoncontent() (pynlpl.formats.folia.SynsetFeature method), 878

phoncontent() (pynlpl.formats.folia.SyntacticUnit method), 616

phoncontent() (pynlpl.formats.folia.SyntaxLayer method), 734

phoncontent() (pynlpl.formats.folia.Table method), 353

phoncontent() (pynlpl.formats.folia.TableHead method), 379

phoncontent() (pynlpl.formats.folia.Term method), 366

phoncontent() (pynlpl.formats.folia.Text method), 392

phoncontent() (pynlpl.formats.folia.TextContent method), 499

phoncontent() (pynlpl.formats.folia.TextMarkupCorrection method), 845

phoncontent() (pynlpl.formats.folia.TextMarkupError method), 855

phoncontent() (pynlpl.formats.folia.TextMarkupGap method), 813

phoncontent() (pynlpl.formats.folia.TextMarkupString method), 823

phoncontent() (pynlpl.formats.folia.TextMarkupStyle method), 834

phoncontent() (pynlpl.formats.folia.TimeSegment method), 639

phoncontent() (pynlpl.formats.folia.TimingLayer method), 757

phoncontent() (pynlpl.formats.folia.Whitespace method), 405

phoncontent() (pynlpl.formats.folia.Word method), 420

phoneme() (pynlpl.formats.folia.Word method), 420

phonemes() (pynlpl.formats.folia.Word method), 420

PhraseTable (class in pynlpl.formats.moses), 1054

PhraseTableClient (class in pynlpl.formats.moses), 1054

poll() (pynlpl.evaluation.ExperimentPool method), 10

pop() (pynlpl.datatypes.FIFOQueue method), 5

pop() (pynlpl.datatypes.PriorityQueue method), 6

pos() (pynlpl.formats.folia.Word method), 420



PosAnnotation (class in `pynlpl.formats.folia`), 437  
 poslog() (`pynlpl.statistics.Distribution` method), 1062  
 postappend() (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 80  
 postappend() (`pynlpl.formats.folia.AbstractElement` method), 31  
 postappend() (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 57  
 postappend() (`pynlpl.formats.folia.AbstractStructureElement` method), 44  
 postappend() (`pynlpl.formats.folia.AbstractTextMarkup` method), 91  
 postappend() (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 68  
 postappend() (`pynlpl.formats.folia.ActorFeature` method), 890  
 postappend() (`pynlpl.formats.folia.Alignment` method), 1014  
 postappend() (`pynlpl.formats.folia.AlignReference` method), 1025  
 postappend() (`pynlpl.formats.folia.Alternative` method), 925  
 postappend() (`pynlpl.formats.folia.AlternativeLayers` method), 936  
 postappend() (`pynlpl.formats.folia.BegindatetimeFeature` method), 901  
 postappend() (`pynlpl.formats.folia.Cell` method), 107  
 postappend() (`pynlpl.formats.folia.Chunk` method), 523  
 postappend() (`pynlpl.formats.folia.ChunkingLayer` method), 652  
 postappend() (`pynlpl.formats.folia.CoreferenceChain` method), 535  
 postappend() (`pynlpl.formats.folia.CoreferenceLayer` method), 664  
 postappend() (`pynlpl.formats.folia.CoreferenceLink` method), 769  
 postappend() (`pynlpl.formats.folia.Correction` method), 948  
 postappend() (`pynlpl.formats.folia.Current` method), 958  
 postappend() (`pynlpl.formats.folia.Definition` method), 120  
 postappend() (`pynlpl.formats.folia.DependenciesLayer` method), 676  
 postappend() (`pynlpl.formats.folia.Dependency` method), 546  
 postappend() (`pynlpl.formats.folia.DependencyDependent` method), 781  
 postappend() (`pynlpl.formats.folia.Description` method), 1036  
 postappend() (`pynlpl.formats.folia.Division` method), 133  
 postappend() (`pynlpl.formats.folia.DomainAnnotation` method), 433  
 postappend() (`pynlpl.formats.folia.EnddatetimeFeature` method), 912  
 postappend() (`pynlpl.formats.folia.EntitiesLayer` method), 687  
 postappend() (`pynlpl.formats.folia.Entity` method), 558  
 postappend() (`pynlpl.formats.folia.Entry` method), 146  
 postappend() (`pynlpl.formats.folia.ErrorDetection` method), 969  
 postappend() (`pynlpl.formats.folia.Event` method), 159  
 postappend() (`pynlpl.formats.folia.Example` method), 172  
 postappend() (`pynlpl.formats.folia.Feature` method), 867  
 postappend() (`pynlpl.formats.folia.Figure` method), 185  
 postappend() (`pynlpl.formats.folia.Gap` method), 196  
 postappend() (`pynlpl.formats.folia.Head` method), 209  
 postappend() (`pynlpl.formats.folia.Headspan` method), 793  
 postappend() (`pynlpl.formats.folia.LangAnnotation` method), 455  
 postappend() (`pynlpl.formats.folia.LemmaAnnotation` method), 466  
 postappend() (`pynlpl.formats.folia.Linebreak` method), 221  
 postappend() (`pynlpl.formats.folia.List` method), 235  
 postappend() (`pynlpl.formats.folia.ListItem` method), 248  
 postappend() (`pynlpl.formats.folia.Metric` method), 1047  
 postappend() (`pynlpl.formats.folia.New` method), 980  
 postappend() (`pynlpl.formats.folia.Note` method), 261  
 postappend() (`pynlpl.formats.folia.Observation` method), 570  
 postappend() (`pynlpl.formats.folia.ObservationLayer` method), 699  
 postappend() (`pynlpl.formats.folia.Original` method), 991  
 postappend() (`pynlpl.formats.folia.Paragraph` method), 274  
 postappend() (`pynlpl.formats.folia.Part` method), 287  
 postappend() (`pynlpl.formats.folia.PhonContent` method), 510  
 postappend() (`pynlpl.formats.folia.PosAnnotation` method), 444  
 postappend() (`pynlpl.formats.folia.Predicate` method), 581  
 postappend() (`pynlpl.formats.folia.Quote` method), 300  
 postappend() (`pynlpl.formats.folia.Reference` method), 313  
 postappend() (`pynlpl.formats.folia.Row` method), 326  
 postappend() (`pynlpl.formats.folia.SemanticRole` method), 628  
 postappend() (`pynlpl.formats.folia.SemanticRolesLayer` method), 746  
 postappend() (`pynlpl.formats.folia.SenseAnnotation` method), 477  
 postappend() (`pynlpl.formats.folia.Sentence` method), 340  
 postappend() (`pynlpl.formats.folia.Sentiment` method), 593  
 postappend() (`pynlpl.formats.folia.SentimentLayer` method), 628

method), 711  
postappend() (pynlpl.formats.folia.Statement method), 605  
postappend() (pynlpl.formats.folia.StatementLayer method), 723  
postappend() (pynlpl.formats.folia.SubjectivityAnnotation method), 488  
postappend() (pynlpl.formats.folia.Suggestion method), 1002  
postappend() (pynlpl.formats.folia.SynsetFeature method), 879  
postappend() (pynlpl.formats.folia.SyntacticUnit method), 616  
postappend() (pynlpl.formats.folia.SyntaxLayer method), 734  
postappend() (pynlpl.formats.folia.Table method), 353  
postappend() (pynlpl.formats.folia.TableHead method), 379  
postappend() (pynlpl.formats.folia.Term method), 366  
postappend() (pynlpl.formats.folia.Text method), 392  
postappend() (pynlpl.formats.folia.TextContent method), 500  
postappend() (pynlpl.formats.folia.TextMarkupCorrection method), 845  
postappend() (pynlpl.formats.folia.TextMarkupError method), 856  
postappend() (pynlpl.formats.folia.TextMarkupGap method), 813  
postappend() (pynlpl.formats.folia.TextMarkupString method), 824  
postappend() (pynlpl.formats.folia.TextMarkupStyle method), 834  
postappend() (pynlpl.formats.folia.TimeSegment method), 640  
postappend() (pynlpl.formats.folia.TimingLayer method), 758  
postappend() (pynlpl.formats.folia.Whitespace method), 405  
postappend() (pynlpl.formats.folia.Word method), 420  
precision() (pynlpl.evaluation.ClassEvaluation method), 10  
Predicate (class in pynlpl.formats.folia), 573  
previous() (pynlpl.formats.folia.AbstractAnnotationLayer method), 80  
previous() (pynlpl.formats.folia.AbstractElement method), 31  
previous() (pynlpl.formats.folia.AbstractSpanAnnotation method), 57  
previous() (pynlpl.formats.folia.AbstractStructureElement method), 44  
previous() (pynlpl.formats.folia.AbstractTextMarkup method), 91  
previous() (pynlpl.formats.folia.AbstractTokenAnnotation method), 68  
previous() (pynlpl.formats.folia.ActorFeature method), 890  
previous() (pynlpl.formats.folia.Alignment method), 1014  
previous() (pynlpl.formats.folia.AlignReference method), 1025  
previous() (pynlpl.formats.folia.Alternative method), 925  
previous() (pynlpl.formats.folia.AlternativeLayers method), 936  
previous() (pynlpl.formats.folia.BegindatetimeFeature method), 901  
previous() (pynlpl.formats.folia.Cell method), 107  
previous() (pynlpl.formats.folia.Chunk method), 523  
previous() (pynlpl.formats.folia.ChunkingLayer method), 652  
previous() (pynlpl.formats.folia.CoreferenceChain method), 535  
previous() (pynlpl.formats.folia.CoreferenceLayer method), 664  
previous() (pynlpl.formats.folia.CoreferenceLink method), 770  
previous() (pynlpl.formats.folia.Correction method), 948  
previous() (pynlpl.formats.folia.Current method), 958  
previous() (pynlpl.formats.folia.Definition method), 120  
previous() (pynlpl.formats.folia.DependenciesLayer method), 676  
previous() (pynlpl.formats.folia.Dependency method), 547  
previous() (pynlpl.formats.folia.DependencyDependent method), 781  
previous() (pynlpl.formats.folia.Description method), 1036  
previous() (pynlpl.formats.folia.Division method), 133  
previous() (pynlpl.formats.folia.DomainAnnotation method), 433  
previous() (pynlpl.formats.folia.EnddatetimeFeature method), 912  
previous() (pynlpl.formats.folia.EntitiesLayer method), 687  
previous() (pynlpl.formats.folia.Entity method), 558  
previous() (pynlpl.formats.folia.Entry method), 146  
previous() (pynlpl.formats.folia.ErrorDetection method), 969  
previous() (pynlpl.formats.folia.Event method), 159  
previous() (pynlpl.formats.folia.Example method), 172  
previous() (pynlpl.formats.folia.Feature method), 867  
previous() (pynlpl.formats.folia.Figure method), 185  
previous() (pynlpl.formats.folia.Gap method), 196  
previous() (pynlpl.formats.folia.Head method), 209  
previous() (pynlpl.formats.folia.Headspan method), 793  
previous() (pynlpl.formats.folia.LangAnnotation method), 455  
previous() (pynlpl.formats.folia.LemmaAnnotation method), 466

- `previous()` (`pynlpl.formats.folia.Linebreak` method), 222
- `previous()` (`pynlpl.formats.folia.List` method), 235
- `previous()` (`pynlpl.formats.folia.ListItem` method), 248
- `previous()` (`pynlpl.formats.folia.Metric` method), 1047
- `previous()` (`pynlpl.formats.folia.New` method), 980
- `previous()` (`pynlpl.formats.folia.Note` method), 261
- `previous()` (`pynlpl.formats.folia.Observation` method), 570
- `previous()` (`pynlpl.formats.folia.ObservationLayer` method), 699
- `previous()` (`pynlpl.formats.folia.Original` method), 991
- `previous()` (`pynlpl.formats.folia.Paragraph` method), 274
- `previous()` (`pynlpl.formats.folia.Part` method), 287
- `previous()` (`pynlpl.formats.folia.PhonContent` method), 510
- `previous()` (`pynlpl.formats.folia.PosAnnotation` method), 444
- `previous()` (`pynlpl.formats.folia.Predicate` method), 581
- `previous()` (`pynlpl.formats.folia.Quote` method), 300
- `previous()` (`pynlpl.formats.folia.Reference` method), 313
- `previous()` (`pynlpl.formats.folia.Row` method), 326
- `previous()` (`pynlpl.formats.folia.SemanticRole` method), 628
- `previous()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 746
- `previous()` (`pynlpl.formats.folia.SenseAnnotation` method), 477
- `previous()` (`pynlpl.formats.folia.Sentence` method), 340
- `previous()` (`pynlpl.formats.folia.Sentiment` method), 593
- `previous()` (`pynlpl.formats.folia.SentimentLayer` method), 711
- `previous()` (`pynlpl.formats.folia.Statement` method), 605
- `previous()` (`pynlpl.formats.folia.StatementLayer` method), 723
- `previous()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 488
- `previous()` (`pynlpl.formats.folia.Suggestion` method), 1002
- `previous()` (`pynlpl.formats.folia.SynsetFeature` method), 879
- `previous()` (`pynlpl.formats.folia.SyntacticUnit` method), 616
- `previous()` (`pynlpl.formats.folia.SyntaxLayer` method), 734
- `previous()` (`pynlpl.formats.folia.Table` method), 353
- `previous()` (`pynlpl.formats.folia.TableHead` method), 379
- `previous()` (`pynlpl.formats.folia.Term` method), 366
- `previous()` (`pynlpl.formats.folia.Text` method), 392
- `previous()` (`pynlpl.formats.folia.TextContent` method), 500
- `previous()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 845
- `previous()` (`pynlpl.formats.folia.TextMarkupError` method), 856
- `previous()` (`pynlpl.formats.folia.TextMarkupGap` method), 813
- `previous()` (`pynlpl.formats.folia.TextMarkupString` method), 824
- `previous()` (`pynlpl.formats.folia.TextMarkupStyle` method), 834
- `previous()` (`pynlpl.formats.folia.TimeSegment` method), 640
- `previous()` (`pynlpl.formats.folia.TimingLayer` method), 758
- `previous()` (`pynlpl.formats.folia.Whitespace` method), 405
- `previous()` (`pynlpl.formats.folia.Word` method), 420
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractAnnotationLayer` attribute), 75
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractElement` attribute), 26
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractSpanAnnotation` attribute), 52
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractStructureElement` attribute), 37
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractTextMarkup` attribute), 86
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AbstractTokenAnnotation` attribute), 63
- `PRIMARYELEMENT` (`pynlpl.formats.folia.ActorFeature` attribute), 885
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Alignment` attribute), 1009
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AlignReference` attribute), 1020
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Alternative` attribute), 919
- `PRIMARYELEMENT` (`pynlpl.formats.folia.AlternativeLayers` attribute), 931
- `PRIMARYELEMENT` (`pynlpl.formats.folia.BegindatetimeFeature` attribute), 896
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Cell` attribute), 100
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Chunk` attribute), 518
- `PRIMARYELEMENT` (`pynlpl.formats.folia.ChunkingLayer` attribute), 646
- `PRIMARYELEMENT` (`pynlpl.formats.folia.CoreferenceChain` attribute), 529
- `PRIMARYELEMENT` (`pynlpl.formats.folia.CoreferenceLayer` attribute), 658
- `PRIMARYELEMENT` (`pynlpl.formats.folia.CoreferenceLink` attribute), 764
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Correction` attribute), 944
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Current` attribute), 953
- `PRIMARYELEMENT` (`pynlpl.formats.folia.Definition` attribute), 113

PRIMARYELEMENT (pynlpl.formats.folia.DependenciesLayer attribute), 670

PRIMARYELEMENT (pynlpl.formats.folia.Dependency attribute), 541

PRIMARYELEMENT (pynlpl.formats.folia.DependencyDependency attribute), 776

PRIMARYELEMENT (pynlpl.formats.folia.Description attribute), 1031

PRIMARYELEMENT (pynlpl.formats.folia.Division attribute), 126

PRIMARYELEMENT (pynlpl.formats.folia.DomainAnnotation attribute), 428

PRIMARYELEMENT (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

PRIMARYELEMENT (pynlpl.formats.folia.EntitiesLayer attribute), 682

PRIMARYELEMENT (pynlpl.formats.folia.Entity attribute), 553

PRIMARYELEMENT (pynlpl.formats.folia.Entry attribute), 139

PRIMARYELEMENT (pynlpl.formats.folia.ErrorDetection attribute), 964

PRIMARYELEMENT (pynlpl.formats.folia.Event attribute), 152

PRIMARYELEMENT (pynlpl.formats.folia.Example attribute), 165

PRIMARYELEMENT (pynlpl.formats.folia.Feature attribute), 862

PRIMARYELEMENT (pynlpl.formats.folia.Figure attribute), 178

PRIMARYELEMENT (pynlpl.formats.folia.Gap attribute), 191

PRIMARYELEMENT (pynlpl.formats.folia.Head attribute), 202

PRIMARYELEMENT (pynlpl.formats.folia.Headspan attribute), 788

PRIMARYELEMENT (pynlpl.formats.folia.LangAnnotation attribute), 450

PRIMARYELEMENT (pynlpl.formats.folia.LemmaAnnotation attribute), 461

PRIMARYELEMENT (pynlpl.formats.folia.Linebreak attribute), 215

PRIMARYELEMENT (pynlpl.formats.folia.List attribute), 228

PRIMARYELEMENT (pynlpl.formats.folia.ListItem attribute), 241

PRIMARYELEMENT (pynlpl.formats.folia.Metric attribute), 1042

PRIMARYELEMENT (pynlpl.formats.folia.New attribute), 975

PRIMARYELEMENT (pynlpl.formats.folia.Note attribute), 254

PRIMARYELEMENT (pynlpl.formats.folia.Observation attribute), 564

PRIMARYELEMENT (pynlpl.formats.folia.ObservationLayer attribute), 694

PRIMARYELEMENT (pynlpl.formats.folia.Original attribute), 986

PRIMARYELEMENT (pynlpl.formats.folia.Paragraph attribute), 267

PRIMARYELEMENT (pynlpl.formats.folia.Part attribute), 280

PRIMARYELEMENT (pynlpl.formats.folia.PhonContent attribute), 506

PRIMARYELEMENT (pynlpl.formats.folia.PosAnnotation attribute), 439

PRIMARYELEMENT (pynlpl.formats.folia.Predicate attribute), 576

PRIMARYELEMENT (pynlpl.formats.folia.Quote attribute), 293

PRIMARYELEMENT (pynlpl.formats.folia.Reference attribute), 306

PRIMARYELEMENT (pynlpl.formats.folia.Row attribute), 319

PRIMARYELEMENT (pynlpl.formats.folia.SemanticRole attribute), 623

PRIMARYELEMENT (pynlpl.formats.folia.SemanticRolesLayer attribute), 741

PRIMARYELEMENT (pynlpl.formats.folia.SenseAnnotation attribute), 472

PRIMARYELEMENT (pynlpl.formats.folia.Sentence attribute), 333

PRIMARYELEMENT (pynlpl.formats.folia.Sentiment attribute), 588

PRIMARYELEMENT (pynlpl.formats.folia.SentimentLayer attribute), 705

PRIMARYELEMENT (pynlpl.formats.folia.Statement attribute), 599

PRIMARYELEMENT (pynlpl.formats.folia.StatementLayer attribute), 717

PRIMARYELEMENT (pynlpl.formats.folia.SubjectivityAnnotation attribute), 483

PRIMARYELEMENT (pynlpl.formats.folia.Suggestion attribute), 997

PRIMARYELEMENT (pynlpl.formats.folia.SynsetFeature attribute), 873

PRIMARYELEMENT (pynlpl.formats.folia.SyntacticUnit attribute), 611

PRIMARYELEMENT (pynlpl.formats.folia.SyntaxLayer attribute), 729

PRIMARYELEMENT (pynlpl.formats.folia.Table attribute), 347

PRIMARYELEMENT (pynlpl.formats.folia.TableHead attribute), 373

PRIMARYELEMENT (pynlpl.formats.folia.Term attribute), 360

PRIMARYELEMENT (pynlpl.formats.folia.Text attribute), 386

PRIMARYELEMENT (pynlpl.formats.folia.TextContent attribute), 495  
 PRIMARYELEMENT (pynlpl.formats.folia.TextMarkupContent attribute), 840  
 PRIMARYELEMENT (pynlpl.formats.folia.TextMarkupError attribute), 851  
 PRIMARYELEMENT (pynlpl.formats.folia.TextMarkupGap attribute), 808  
 PRIMARYELEMENT (pynlpl.formats.folia.TextMarkupString attribute), 819  
 PRIMARYELEMENT (pynlpl.formats.folia.TextMarkupStyle attribute), 830  
 PRIMARYELEMENT (pynlpl.formats.folia.TimeSegment attribute), 634  
 PRIMARYELEMENT (pynlpl.formats.folia.TimingLayer attribute), 752  
 PRIMARYELEMENT (pynlpl.formats.folia.Whitespace attribute), 399  
 PRIMARYELEMENT (pynlpl.formats.folia.Word attribute), 413  
 print\_dptable() (pynlpl.statistics.HiddenMarkovModel method), 1063  
 PRINTABLE (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 75  
 PRINTABLE (pynlpl.formats.folia.AbstractElement attribute), 26  
 PRINTABLE (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52  
 PRINTABLE (pynlpl.formats.folia.AbstractStructureElement attribute), 37  
 PRINTABLE (pynlpl.formats.folia.AbstractTextMarkup attribute), 86  
 PRINTABLE (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63  
 PRINTABLE (pynlpl.formats.folia.ActorFeature attribute), 885  
 PRINTABLE (pynlpl.formats.folia.Alignment attribute), 1009  
 PRINTABLE (pynlpl.formats.folia.AlignReference attribute), 1020  
 PRINTABLE (pynlpl.formats.folia.Alternative attribute), 919  
 PRINTABLE (pynlpl.formats.folia.AlternativeLayers attribute), 931  
 PRINTABLE (pynlpl.formats.folia.BegindatetimeFeature attribute), 896  
 PRINTABLE (pynlpl.formats.folia.Cell attribute), 100  
 PRINTABLE (pynlpl.formats.folia.Chunk attribute), 518  
 PRINTABLE (pynlpl.formats.folia.ChunkingLayer attribute), 646  
 PRINTABLE (pynlpl.formats.folia.CoreferenceChain attribute), 529  
 PRINTABLE (pynlpl.formats.folia.CoreferenceLayer attribute), 658  
 PRINTABLE (pynlpl.formats.folia.CoreferenceLink attribute), 764  
 PRINTABLE (pynlpl.formats.folia.Correction attribute), 944  
 PRINTABLE (pynlpl.formats.folia.Current attribute), 953  
 PRINTABLE (pynlpl.formats.folia.Definition attribute), 113  
 PRINTABLE (pynlpl.formats.folia.DependenciesLayer attribute), 670  
 PRINTABLE (pynlpl.formats.folia.Dependency attribute), 541  
 PRINTABLE (pynlpl.formats.folia.DependencyDependent attribute), 776  
 PRINTABLE (pynlpl.formats.folia.Description attribute), 1031  
 PRINTABLE (pynlpl.formats.folia.Division attribute), 126  
 PRINTABLE (pynlpl.formats.folia.DomainAnnotation attribute), 428  
 PRINTABLE (pynlpl.formats.folia.EnddatetimeFeature attribute), 907  
 PRINTABLE (pynlpl.formats.folia.EntitiesLayer attribute), 682  
 PRINTABLE (pynlpl.formats.folia.Entity attribute), 553  
 PRINTABLE (pynlpl.formats.folia.Entry attribute), 139  
 PRINTABLE (pynlpl.formats.folia.ErrorDetection attribute), 964  
 PRINTABLE (pynlpl.formats.folia.Event attribute), 152  
 PRINTABLE (pynlpl.formats.folia.Example attribute), 165  
 PRINTABLE (pynlpl.formats.folia.Feature attribute), 862  
 PRINTABLE (pynlpl.formats.folia.Figure attribute), 178  
 PRINTABLE (pynlpl.formats.folia.Gap attribute), 191  
 PRINTABLE (pynlpl.formats.folia.Head attribute), 202  
 PRINTABLE (pynlpl.formats.folia.Headspan attribute), 788  
 PRINTABLE (pynlpl.formats.folia.LangAnnotation attribute), 451  
 PRINTABLE (pynlpl.formats.folia.LemmaAnnotation attribute), 462  
 PRINTABLE (pynlpl.formats.folia.Linebreak attribute), 215  
 PRINTABLE (pynlpl.formats.folia.List attribute), 228  
 PRINTABLE (pynlpl.formats.folia.ListItem attribute), 241  
 PRINTABLE (pynlpl.formats.folia.Metric attribute), 1042  
 PRINTABLE (pynlpl.formats.folia.New attribute), 975  
 PRINTABLE (pynlpl.formats.folia.Note attribute), 254  
 PRINTABLE (pynlpl.formats.folia.Observation attribute), 564  
 PRINTABLE (pynlpl.formats.folia.ObservationLayer attribute), 694  
 PRINTABLE (pynlpl.formats.folia.Original attribute),



- 986  
PRINTABLE (pynlpl.formats.folia.Paragraph attribute), 267  
PRINTABLE (pynlpl.formats.folia.Part attribute), 280  
PRINTABLE (pynlpl.formats.folia.PhonContent attribute), 506  
PRINTABLE (pynlpl.formats.folia.PosAnnotation attribute), 440  
PRINTABLE (pynlpl.formats.folia.Predicate attribute), 576  
PRINTABLE (pynlpl.formats.folia.Quote attribute), 293  
PRINTABLE (pynlpl.formats.folia.Reference attribute), 306  
PRINTABLE (pynlpl.formats.folia.Row attribute), 319  
PRINTABLE (pynlpl.formats.folia.SemanticRole attribute), 623  
PRINTABLE (pynlpl.formats.folia.SemanticRolesLayer attribute), 741  
PRINTABLE (pynlpl.formats.folia.SenseAnnotation attribute), 473  
PRINTABLE (pynlpl.formats.folia.Sentence attribute), 333  
PRINTABLE (pynlpl.formats.folia.Sentiment attribute), 588  
PRINTABLE (pynlpl.formats.folia.SentimentLayer attribute), 705  
PRINTABLE (pynlpl.formats.folia.Statement attribute), 599  
PRINTABLE (pynlpl.formats.folia.StatementLayer attribute), 717  
PRINTABLE (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484  
PRINTABLE (pynlpl.formats.folia.Suggestion attribute), 997  
PRINTABLE (pynlpl.formats.folia.SynsetFeature attribute), 874  
PRINTABLE (pynlpl.formats.folia.SyntacticUnit attribute), 611  
PRINTABLE (pynlpl.formats.folia.SyntaxLayer attribute), 729  
PRINTABLE (pynlpl.formats.folia.Table attribute), 347  
PRINTABLE (pynlpl.formats.folia.TableHead attribute), 373  
PRINTABLE (pynlpl.formats.folia.Term attribute), 360  
PRINTABLE (pynlpl.formats.folia.Text attribute), 386  
PRINTABLE (pynlpl.formats.folia.TextContent attribute), 495  
PRINTABLE (pynlpl.formats.folia.TextMarkupCorrection attribute), 840  
PRINTABLE (pynlpl.formats.folia.TextMarkupError attribute), 851  
PRINTABLE (pynlpl.formats.folia.TextMarkupGap attribute), 808  
PRINTABLE (pynlpl.formats.folia.TextMarkupString attribute), 819  
PRINTABLE (pynlpl.formats.folia.TextMarkupStyle attribute), 830  
PRINTABLE (pynlpl.formats.folia.TimeSegment attribute), 634  
PRINTABLE (pynlpl.formats.folia.TimingLayer attribute), 752  
PRINTABLE (pynlpl.formats.folia.Whitespace attribute), 399  
PRINTABLE (pynlpl.formats.folia.Word attribute), 413  
PriorityQueue (class in pynlpl.datatypes), 5  
prob() (pynlpl.lm.lm.ARPALanguageModel.NgramsProbs method), 1057  
ProcessFailed, 10  
product() (in module pynlpl.statistics), 1064  
prune() (pynlpl.datatypes.PriorityQueue method), 6  
prune() (pynlpl.search.AbstractSearch method), 1059  
prune() (pynlpl.search.BeamedBestFirstSearch method), 1060  
prune() (pynlpl.search.EarlyEagerBeamSearch method), 1060  
prune() (pynlpl.search.StochasticBeamSearch method), 1060  
prunebyScore() (pynlpl.datatypes.PriorityQueue method), 6  
publisher() (pynlpl.formats.folia.Document method), 20  
pynlpl.common (module), 3  
pynlpl.datatypes (module), 5  
pynlpl.evaluation (module), 9  
pynlpl.formats.cgn (module), 1053  
pynlpl.formats.folia (module), 11  
pynlpl.formats.giza (module), 1053  
pynlpl.formats.moses (module), 1054  
pynlpl.formats.sonar (module), 1054  
pynlpl.formats.taggedata (module), 1055  
pynlpl.formats.timbl (module), 1055  
pynlpl.lm.client (module), 1058  
pynlpl.lm.lm (module), 1057  
pynlpl.lm.srlm (module), 1057  
pynlpl.search (module), 1059  
pynlpl.statistics (module), 1062  
pynlpl.textprocessors (module), 1066
- ## Q
- Query (class in pynlpl.formats.fql), 802  
Queue (class in pynlpl.datatypes), 6  
Quote (class in pynlpl.formats.folia), 290
- ## R
- randomprune() (pynlpl.datatypes.PriorityQueue method), 6  
Reader (class in pynlpl.formats.folia), 804  
recall() (pynlpl.evaluation.ClassEvaluation method), 10  
reducible() (pynlpl.statistics.MarkovChain method), 1063

- Reference (class in `pynlpl.formats.folia`), 303
- ReflowText (class in `pynlpl.textprocessors`), 1066
- `relaxng()` (`pynlpl.formats.folia.AbstractAnnotationLayer` class method), 80
- `relaxng()` (`pynlpl.formats.folia.AbstractElement` class method), 31
- `relaxng()` (`pynlpl.formats.folia.AbstractSpanAnnotation` class method), 57
- `relaxng()` (`pynlpl.formats.folia.AbstractStructureElement` class method), 44
- `relaxng()` (`pynlpl.formats.folia.AbstractTextMarkup` class method), 91
- `relaxng()` (`pynlpl.formats.folia.AbstractTokenAnnotation` class method), 68
- `relaxng()` (`pynlpl.formats.folia.ActorFeature` class method), 890
- `relaxng()` (`pynlpl.formats.folia.Alignment` class method), 1014
- `relaxng()` (`pynlpl.formats.folia.AlignReference` class method), 1025
- `relaxng()` (`pynlpl.formats.folia.Alternative` class method), 925
- `relaxng()` (`pynlpl.formats.folia.AlternativeLayers` class method), 936
- `relaxng()` (`pynlpl.formats.folia.BegindatetimeFeature` class method), 901
- `relaxng()` (`pynlpl.formats.folia.Cell` class method), 107
- `relaxng()` (`pynlpl.formats.folia.Chunk` class method), 523
- `relaxng()` (`pynlpl.formats.folia.ChunkingLayer` class method), 652
- `relaxng()` (`pynlpl.formats.folia.CoreferenceChain` class method), 535
- `relaxng()` (`pynlpl.formats.folia.CoreferenceLayer` class method), 664
- `relaxng()` (`pynlpl.formats.folia.CoreferenceLink` class method), 770
- `relaxng()` (`pynlpl.formats.folia.Correction` class method), 948
- `relaxng()` (`pynlpl.formats.folia.Current` class method), 959
- `relaxng()` (`pynlpl.formats.folia.Definition` class method), 120
- `relaxng()` (`pynlpl.formats.folia.DependenciesLayer` class method), 676
- `relaxng()` (`pynlpl.formats.folia.Dependency` class method), 547
- `relaxng()` (`pynlpl.formats.folia.DependencyDependent` class method), 782
- `relaxng()` (`pynlpl.formats.folia.Description` class method), 1037
- `relaxng()` (`pynlpl.formats.folia.Division` class method), 133
- `relaxng()` (`pynlpl.formats.folia.DomainAnnotation` class method), 434
- `relaxng()` (`pynlpl.formats.folia.EnddatetimeFeature` class method), 912
- `relaxng()` (`pynlpl.formats.folia.EntitiesLayer` class method), 688
- `relaxng()` (`pynlpl.formats.folia.Entity` class method), 558
- `relaxng()` (`pynlpl.formats.folia.Entry` class method), 146
- `relaxng()` (`pynlpl.formats.folia.ErrorDetection` class method), 970
- `relaxng()` (`pynlpl.formats.folia.Event` class method), 159
- `relaxng()` (`pynlpl.formats.folia.Example` class method), 172
- `relaxng()` (`pynlpl.formats.folia.Feature` class method), 868
- `relaxng()` (`pynlpl.formats.folia.Figure` class method), 185
- `relaxng()` (`pynlpl.formats.folia.Gap` class method), 196
- `relaxng()` (`pynlpl.formats.folia.Head` class method), 209
- `relaxng()` (`pynlpl.formats.folia.Headspan` class method), 793
- `relaxng()` (`pynlpl.formats.folia.LangAnnotation` class method), 456
- `relaxng()` (`pynlpl.formats.folia.LemmaAnnotation` class method), 467
- `relaxng()` (`pynlpl.formats.folia.Linebreak` class method), 222
- `relaxng()` (`pynlpl.formats.folia.List` class method), 235
- `relaxng()` (`pynlpl.formats.folia.ListItem` class method), 248
- `relaxng()` (`pynlpl.formats.folia.Metric` class method), 1048
- `relaxng()` (`pynlpl.formats.folia.New` class method), 981
- `relaxng()` (`pynlpl.formats.folia.Note` class method), 261
- `relaxng()` (`pynlpl.formats.folia.Observation` class method), 570
- `relaxng()` (`pynlpl.formats.folia.ObservationLayer` class method), 699
- `relaxng()` (`pynlpl.formats.folia.Original` class method), 992
- `relaxng()` (`pynlpl.formats.folia.Paragraph` class method), 274
- `relaxng()` (`pynlpl.formats.folia.Part` class method), 287
- `relaxng()` (`pynlpl.formats.folia.PhonContent` class method), 510
- `relaxng()` (`pynlpl.formats.folia.PosAnnotation` class method), 445
- `relaxng()` (`pynlpl.formats.folia.Predicate` class method), 582
- `relaxng()` (`pynlpl.formats.folia.Quote` class method), 300
- `relaxng()` (`pynlpl.formats.folia.Reference` class method), 313
- `relaxng()` (`pynlpl.formats.folia.Row` class method), 326
- `relaxng()` (`pynlpl.formats.folia.SemanticRole` class method), 628
- `relaxng()` (`pynlpl.formats.folia.SemanticRolesLayer` class method), 746
- `relaxng()` (`pynlpl.formats.folia.SenseAnnotation` class

- method), 478
- `relaxng()` (`pynlpl.formats.folia.Sentence` class method), 341
- `relaxng()` (`pynlpl.formats.folia.Sentiment` class method), 593
- `relaxng()` (`pynlpl.formats.folia.SentimentLayer` class method), 711
- `relaxng()` (`pynlpl.formats.folia.Statement` class method), 605
- `relaxng()` (`pynlpl.formats.folia.StatementLayer` class method), 723
- `relaxng()` (`pynlpl.formats.folia.SubjectivityAnnotation` class method), 489
- `relaxng()` (`pynlpl.formats.folia.Suggestion` class method), 1003
- `relaxng()` (`pynlpl.formats.folia.SynsetFeature` class method), 879
- `relaxng()` (`pynlpl.formats.folia.SyntacticUnit` class method), 617
- `relaxng()` (`pynlpl.formats.folia.SyntaxLayer` class method), 735
- `relaxng()` (`pynlpl.formats.folia.Table` class method), 354
- `relaxng()` (`pynlpl.formats.folia.TableHead` class method), 380
- `relaxng()` (`pynlpl.formats.folia.Term` class method), 367
- `relaxng()` (`pynlpl.formats.folia.Text` class method), 393
- `relaxng()` (`pynlpl.formats.folia.TextContent` class method), 500
- `relaxng()` (`pynlpl.formats.folia.TextMarkupCorrection` class method), 845
- `relaxng()` (`pynlpl.formats.folia.TextMarkupError` class method), 856
- `relaxng()` (`pynlpl.formats.folia.TextMarkupGap` class method), 813
- `relaxng()` (`pynlpl.formats.folia.TextMarkupString` class method), 824
- `relaxng()` (`pynlpl.formats.folia.TextMarkupStyle` class method), 835
- `relaxng()` (`pynlpl.formats.folia.TimeSegment` class method), 640
- `relaxng()` (`pynlpl.formats.folia.TimingLayer` class method), 758
- `relaxng()` (`pynlpl.formats.folia.Whitespace` class method), 406
- `relaxng()` (`pynlpl.formats.folia.Word` class method), 421
- `remove()` (`pynlpl.datatypes.PatternSet` method), 5
- `remove()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 80
- `remove()` (`pynlpl.formats.folia.AbstractElement` method), 31
- `remove()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 57
- `remove()` (`pynlpl.formats.folia.AbstractStructureElement` method), 44
- `remove()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 91
- `remove()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 69
- `remove()` (`pynlpl.formats.folia.ActorFeature` method), 890
- `remove()` (`pynlpl.formats.folia.Alignment` method), 1014
- `remove()` (`pynlpl.formats.folia.AlignReference` method), 1025
- `remove()` (`pynlpl.formats.folia.Alternative` method), 925
- `remove()` (`pynlpl.formats.folia.AlternativeLayers` method), 936
- `remove()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 901
- `remove()` (`pynlpl.formats.folia.Cell` method), 107
- `remove()` (`pynlpl.formats.folia.Chunk` method), 523
- `remove()` (`pynlpl.formats.folia.ChunkingLayer` method), 652
- `remove()` (`pynlpl.formats.folia.CoreferenceChain` method), 535
- `remove()` (`pynlpl.formats.folia.CoreferenceLayer` method), 664
- `remove()` (`pynlpl.formats.folia.CoreferenceLink` method), 770
- `remove()` (`pynlpl.formats.folia.Correction` method), 948
- `remove()` (`pynlpl.formats.folia.Current` method), 959
- `remove()` (`pynlpl.formats.folia.Definition` method), 120
- `remove()` (`pynlpl.formats.folia.DependenciesLayer` method), 676
- `remove()` (`pynlpl.formats.folia.Dependency` method), 547
- `remove()` (`pynlpl.formats.folia.DependencyDependent` method), 782
- `remove()` (`pynlpl.formats.folia.Description` method), 1037
- `remove()` (`pynlpl.formats.folia.Division` method), 133
- `remove()` (`pynlpl.formats.folia.DomainAnnotation` method), 434
- `remove()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 912
- `remove()` (`pynlpl.formats.folia.EntitiesLayer` method), 688
- `remove()` (`pynlpl.formats.folia.Entity` method), 558
- `remove()` (`pynlpl.formats.folia.Entry` method), 146
- `remove()` (`pynlpl.formats.folia.ErrorDetection` method), 970
- `remove()` (`pynlpl.formats.folia.Event` method), 159
- `remove()` (`pynlpl.formats.folia.Example` method), 172
- `remove()` (`pynlpl.formats.folia.Feature` method), 868
- `remove()` (`pynlpl.formats.folia.Figure` method), 185
- `remove()` (`pynlpl.formats.folia.Gap` method), 196
- `remove()` (`pynlpl.formats.folia.Head` method), 209
- `remove()` (`pynlpl.formats.folia.Headspan` method), 793
- `remove()` (`pynlpl.formats.folia.LangAnnotation` method), 456
- `remove()` (`pynlpl.formats.folia.LemmaAnnotation`



- method), 467
- remove() (pynlpl.formats.folia.Linebreak method), 222
- remove() (pynlpl.formats.folia.List method), 235
- remove() (pynlpl.formats.folia.ListItem method), 248
- remove() (pynlpl.formats.folia.Metric method), 1048
- remove() (pynlpl.formats.folia.New method), 981
- remove() (pynlpl.formats.folia.Note method), 261
- remove() (pynlpl.formats.folia.Observation method), 570
- remove() (pynlpl.formats.folia.ObservationLayer method), 699
- remove() (pynlpl.formats.folia.Original method), 992
- remove() (pynlpl.formats.folia.Paragraph method), 274
- remove() (pynlpl.formats.folia.Part method), 287
- remove() (pynlpl.formats.folia.PhonContent method), 510
- remove() (pynlpl.formats.folia.PosAnnotation method), 445
- remove() (pynlpl.formats.folia.Predicate method), 582
- remove() (pynlpl.formats.folia.Quote method), 300
- remove() (pynlpl.formats.folia.Reference method), 313
- remove() (pynlpl.formats.folia.Row method), 326
- remove() (pynlpl.formats.folia.SemanticRole method), 628
- remove() (pynlpl.formats.folia.SemanticRolesLayer method), 746
- remove() (pynlpl.formats.folia.SenseAnnotation method), 478
- remove() (pynlpl.formats.folia.Sentence method), 341
- remove() (pynlpl.formats.folia.Sentiment method), 593
- remove() (pynlpl.formats.folia.SentimentLayer method), 711
- remove() (pynlpl.formats.folia.Statement method), 605
- remove() (pynlpl.formats.folia.StatementLayer method), 723
- remove() (pynlpl.formats.folia.SubjectivityAnnotation method), 489
- remove() (pynlpl.formats.folia.Suggestion method), 1003
- remove() (pynlpl.formats.folia.SynsetFeature method), 879
- remove() (pynlpl.formats.folia.SyntacticUnit method), 617
- remove() (pynlpl.formats.folia.SyntaxLayer method), 735
- remove() (pynlpl.formats.folia.Table method), 354
- remove() (pynlpl.formats.folia.TableHead method), 380
- remove() (pynlpl.formats.folia.Term method), 367
- remove() (pynlpl.formats.folia.Text method), 393
- remove() (pynlpl.formats.folia.TextContent method), 500
- remove() (pynlpl.formats.folia.TextMarkupCorrection method), 845
- remove() (pynlpl.formats.folia.TextMarkupError method), 856
- remove() (pynlpl.formats.folia.TextMarkupGap method), 813
- remove() (pynlpl.formats.folia.TextMarkupString method), 824
- remove() (pynlpl.formats.folia.TextMarkupStyle method), 835
- remove() (pynlpl.formats.folia.TimeSegment method), 640
- remove() (pynlpl.formats.folia.TimingLayer method), 758
- remove() (pynlpl.formats.folia.Whitespace method), 406
- remove() (pynlpl.formats.folia.Word method), 421
- replace() (pynlpl.formats.folia.AbstractAnnotationLayer method), 81
- replace() (pynlpl.formats.folia.AbstractElement method), 31
- replace() (pynlpl.formats.folia.AbstractSpanAnnotation method), 58
- replace() (pynlpl.formats.folia.AbstractStructureElement method), 44
- replace() (pynlpl.formats.folia.AbstractTextMarkup method), 91
- replace() (pynlpl.formats.folia.AbstractTokenAnnotation method), 69
- replace() (pynlpl.formats.folia.ActorFeature method), 890
- replace() (pynlpl.formats.folia.Alignment method), 1014
- replace() (pynlpl.formats.folia.AlignReference method), 1025
- replace() (pynlpl.formats.folia.Alternative method), 925
- replace() (pynlpl.formats.folia.AlternativeLayers method), 937
- replace() (pynlpl.formats.folia.BegindatetimeFeature method), 901
- replace() (pynlpl.formats.folia.Cell method), 107
- replace() (pynlpl.formats.folia.Chunk method), 524
- replace() (pynlpl.formats.folia.ChunkingLayer method), 652
- replace() (pynlpl.formats.folia.CoreferenceChain method), 535
- replace() (pynlpl.formats.folia.CoreferenceLayer method), 664
- replace() (pynlpl.formats.folia.CoreferenceLink method), 770
- replace() (pynlpl.formats.folia.Correction method), 948
- replace() (pynlpl.formats.folia.Current method), 959
- replace() (pynlpl.formats.folia.Definition method), 120
- replace() (pynlpl.formats.folia.DependenciesLayer method), 676
- replace() (pynlpl.formats.folia.Dependency method), 547
- replace() (pynlpl.formats.folia.DependencyDependent method), 782
- replace() (pynlpl.formats.folia.Description method), 1037
- replace() (pynlpl.formats.folia.Division method), 133
- replace() (pynlpl.formats.folia.DomainAnnotation method), 434

`replace()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 912

`replace()` (`pynlpl.formats.folia.EntitiesLayer` method), 688

`replace()` (`pynlpl.formats.folia.Entity` method), 558

`replace()` (`pynlpl.formats.folia.Entry` method), 146

`replace()` (`pynlpl.formats.folia.ErrorDetection` method), 970

`replace()` (`pynlpl.formats.folia.Event` method), 159

`replace()` (`pynlpl.formats.folia.Example` method), 172

`replace()` (`pynlpl.formats.folia.Feature` method), 868

`replace()` (`pynlpl.formats.folia.Figure` method), 185

`replace()` (`pynlpl.formats.folia.Gap` method), 196

`replace()` (`pynlpl.formats.folia.Head` method), 209

`replace()` (`pynlpl.formats.folia.Headspan` method), 793

`replace()` (`pynlpl.formats.folia.LangAnnotation` method), 456

`replace()` (`pynlpl.formats.folia.LemmaAnnotation` method), 467

`replace()` (`pynlpl.formats.folia.Linebreak` method), 222

`replace()` (`pynlpl.formats.folia.List` method), 235

`replace()` (`pynlpl.formats.folia.ListItem` method), 248

`replace()` (`pynlpl.formats.folia.Metric` method), 1048

`replace()` (`pynlpl.formats.folia.New` method), 981

`replace()` (`pynlpl.formats.folia.Note` method), 261

`replace()` (`pynlpl.formats.folia.Observation` method), 570

`replace()` (`pynlpl.formats.folia.ObservationLayer` method), 700

`replace()` (`pynlpl.formats.folia.Original` method), 992

`replace()` (`pynlpl.formats.folia.Paragraph` method), 274

`replace()` (`pynlpl.formats.folia.Part` method), 287

`replace()` (`pynlpl.formats.folia.PhonContent` method), 510

`replace()` (`pynlpl.formats.folia.PosAnnotation` method), 445

`replace()` (`pynlpl.formats.folia.Predicate` method), 582

`replace()` (`pynlpl.formats.folia.Quote` method), 300

`replace()` (`pynlpl.formats.folia.Reference` method), 313

`replace()` (`pynlpl.formats.folia.Row` method), 326

`replace()` (`pynlpl.formats.folia.SemanticRole` method), 628

`replace()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 747

`replace()` (`pynlpl.formats.folia.SenseAnnotation` method), 478

`replace()` (`pynlpl.formats.folia.Sentence` method), 341

`replace()` (`pynlpl.formats.folia.Sentiment` method), 593

`replace()` (`pynlpl.formats.folia.SentimentLayer` method), 711

`replace()` (`pynlpl.formats.folia.Statement` method), 605

`replace()` (`pynlpl.formats.folia.StatementLayer` method), 723

`replace()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 489

`replace()` (`pynlpl.formats.folia.Suggestion` method), 1003

`replace()` (`pynlpl.formats.folia.SynsetFeature` method), 879

`replace()` (`pynlpl.formats.folia.SyntacticUnit` method), 617

`replace()` (`pynlpl.formats.folia.SyntaxLayer` method), 735

`replace()` (`pynlpl.formats.folia.Table` method), 354

`replace()` (`pynlpl.formats.folia.TableHead` method), 380

`replace()` (`pynlpl.formats.folia.Term` method), 367

`replace()` (`pynlpl.formats.folia.Text` method), 393

`replace()` (`pynlpl.formats.folia.TextContent` method), 500

`replace()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 845

`replace()` (`pynlpl.formats.folia.TextMarkupError` method), 856

`replace()` (`pynlpl.formats.folia.TextMarkupGap` method), 813

`replace()` (`pynlpl.formats.folia.TextMarkupString` method), 824

`replace()` (`pynlpl.formats.folia.TextMarkupStyle` method), 835

`replace()` (`pynlpl.formats.folia.TimeSegment` method), 640

`replace()` (`pynlpl.formats.folia.TimingLayer` method), 758

`replace()` (`pynlpl.formats.folia.Whitespace` method), 406

`replace()` (`pynlpl.formats.folia.Word` method), 421

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractAnnotationLayer` attribute), 75

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractElement` attribute), 26

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractSpanAnnotation` attribute), 52

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractStructureElement` attribute), 37

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractTextMarkup` attribute), 86

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AbstractTokenAnnotation` attribute), 63

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.ActorFeature` attribute), 885

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.Alignment` attribute), 1009

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AlignReference` attribute), 1020

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.Alternative` attribute), 919

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.AlternativeLayers` attribute), 931

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.BegindatetimeFeature` attribute), 896

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.Cell` attribute), 100

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.Chunk` attribute), 518

`REQUIRED_ATTRIBS` (`pynlpl.formats.folia.ChunkingLayer`

- attribute), 646
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.CoreferenceClass attribute), 529
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.CoreferenceLayer attribute), 658
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.CoreferenceList attribute), 764
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Correction attribute), 944
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Current attribute), 953
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Definition attribute), 113
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Dependencies attribute), 670
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Dependency attribute), 541
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.DependencyDescription attribute), 776
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Description attribute), 1031
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Division attribute), 126
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.DomainAnnotation attribute), 428
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.EnddatetimeFeature attribute), 907
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.EntitiesLayer attribute), 682
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Entity attribute), 553
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Entry attribute), 139
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.ErrorDetection attribute), 964
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Event attribute), 152
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Example attribute), 165
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Feature attribute), 862
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Figure attribute), 178
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Gap attribute), 191
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Head attribute), 202
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Headspan attribute), 788
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.LangAnnotation attribute), 451
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Linebreak attribute), 215
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.List attribute), 228
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.ListItem attribute), 241
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Metric attribute), 1042
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.New attribute), 976
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Note attribute), 254
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Observation attribute), 564
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.ObservationLayer attribute), 694
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Original attribute), 987
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Paragraph attribute), 267
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Part attribute), 280
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.PhonContent attribute), 506
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.PosAnnotation attribute), 440
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Predicate attribute), 576
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Quote attribute), 293
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Reference attribute), 306
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Row attribute), 319
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SemanticRole attribute), 623
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SenseAnnotation attribute), 473
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Sentence attribute), 333
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Sentiment attribute), 588
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SentimentLayer attribute), 705
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Statement attribute), 599
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.StatementLayer attribute), 717
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.Suggestion attribute), 998
- REQUIRED\_ATTRIBS (pynlpl.formats.folia.SynsetFeature attribute), 215

attribute), 874

REQUIRED\_ATTRIBS (pynlpl.formats.folia.SyntacticUnit attribute), 611

REQUIRED\_ATTRIBS (pynlpl.formats.folia.SyntaxLayer attribute), 729

REQUIRED\_ATTRIBS (pynlpl.formats.folia.Table attribute), 347

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TableHead attribute), 373

REQUIRED\_ATTRIBS (pynlpl.formats.folia.Term attribute), 360

REQUIRED\_ATTRIBS (pynlpl.formats.folia.Text attribute), 386

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextContent attribute), 495

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextMarkupContent attribute), 840

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextMarkupError attribute), 851

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextMarkupGap attribute), 808

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextMarkupString attribute), 819

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TextMarkupStyle attribute), 830

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TimeSegment attribute), 634

REQUIRED\_ATTRIBS (pynlpl.formats.folia.TimingLayer attribute), 752

REQUIRED\_ATTRIBS (pynlpl.formats.folia.Whitespace attribute), 399

REQUIRED\_ATTRIBS (pynlpl.formats.folia.Word attribute), 413

REQUIRED\_DATA (pynlpl.formats.folia.AbstractAnnotation attribute), 75

REQUIRED\_DATA (pynlpl.formats.folia.AbstractElement attribute), 26

REQUIRED\_DATA (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52

REQUIRED\_DATA (pynlpl.formats.folia.AbstractStructure attribute), 37

REQUIRED\_DATA (pynlpl.formats.folia.AbstractTextMarkup attribute), 86

REQUIRED\_DATA (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63

REQUIRED\_DATA (pynlpl.formats.folia.ActorFeature attribute), 885

REQUIRED\_DATA (pynlpl.formats.folia.Alignment attribute), 1009

REQUIRED\_DATA (pynlpl.formats.folia.AlignReference attribute), 1020

REQUIRED\_DATA (pynlpl.formats.folia.Alternative attribute), 919

REQUIRED\_DATA (pynlpl.formats.folia.AlternativeLayers attribute), 931

REQUIRED\_DATA (pynlpl.formats.folia.BegindatetimeFeature attribute), 896

REQUIRED\_DATA (pynlpl.formats.folia.Cell attribute), 100

REQUIRED\_DATA (pynlpl.formats.folia.Chunk attribute), 518

REQUIRED\_DATA (pynlpl.formats.folia.ChunkingLayer attribute), 646

REQUIRED\_DATA (pynlpl.formats.folia.CoreferenceChain attribute), 530

REQUIRED\_DATA (pynlpl.formats.folia.CoreferenceLayer attribute), 658

REQUIRED\_DATA (pynlpl.formats.folia.CoreferenceLink attribute), 764

REQUIRED\_DATA (pynlpl.formats.folia.Correction attribute), 944

REQUIRED\_DATA (pynlpl.formats.folia.Current attribute), 953

REQUIRED\_DATA (pynlpl.formats.folia.Definition attribute), 113

REQUIRED\_DATA (pynlpl.formats.folia.DependenciesLayer attribute), 670

REQUIRED\_DATA (pynlpl.formats.folia.Dependency attribute), 541

REQUIRED\_DATA (pynlpl.formats.folia.DependencyDependent attribute), 776

REQUIRED\_DATA (pynlpl.formats.folia.Description attribute), 1031

REQUIRED\_DATA (pynlpl.formats.folia.Division attribute), 126

REQUIRED\_DATA (pynlpl.formats.folia.DomainAnnotation attribute), 429

REQUIRED\_DATA (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

REQUIRED\_DATA (pynlpl.formats.folia.EntitiesLayer attribute), 682

REQUIRED\_DATA (pynlpl.formats.folia.Entity attribute), 553

REQUIRED\_DATA (pynlpl.formats.folia.Entry attribute), 139

REQUIRED\_DATA (pynlpl.formats.folia.ErrorDetection attribute), 965

REQUIRED\_DATA (pynlpl.formats.folia.Event attribute), 152

REQUIRED\_DATA (pynlpl.formats.folia.Example attribute), 165

REQUIRED\_DATA (pynlpl.formats.folia.Feature attribute), 862

REQUIRED\_DATA (pynlpl.formats.folia.Figure attribute), 178

REQUIRED\_DATA (pynlpl.formats.folia.Gap attribute), 191

REQUIRED\_DATA (pynlpl.formats.folia.Head attribute), 931

- 202
- REQUIRED\_DATA (pynlpl.formats.folia.Headspan attribute), 788
- REQUIRED\_DATA (pynlpl.formats.folia.LangAnnotation attribute), 451
- REQUIRED\_DATA (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- REQUIRED\_DATA (pynlpl.formats.folia.Linebreak attribute), 215
- REQUIRED\_DATA (pynlpl.formats.folia.List attribute), 228
- REQUIRED\_DATA (pynlpl.formats.folia.ListItem attribute), 241
- REQUIRED\_DATA (pynlpl.formats.folia.Metric attribute), 1043
- REQUIRED\_DATA (pynlpl.formats.folia.New attribute), 976
- REQUIRED\_DATA (pynlpl.formats.folia.Note attribute), 254
- REQUIRED\_DATA (pynlpl.formats.folia.Observation attribute), 564
- REQUIRED\_DATA (pynlpl.formats.folia.ObservationLayer attribute), 694
- REQUIRED\_DATA (pynlpl.formats.folia.Original attribute), 987
- REQUIRED\_DATA (pynlpl.formats.folia.Paragraph attribute), 267
- REQUIRED\_DATA (pynlpl.formats.folia.Part attribute), 280
- REQUIRED\_DATA (pynlpl.formats.folia.PhonContent attribute), 506
- REQUIRED\_DATA (pynlpl.formats.folia.PosAnnotation attribute), 440
- REQUIRED\_DATA (pynlpl.formats.folia.Predicate attribute), 576
- REQUIRED\_DATA (pynlpl.formats.folia.Quote attribute), 293
- REQUIRED\_DATA (pynlpl.formats.folia.Reference attribute), 306
- REQUIRED\_DATA (pynlpl.formats.folia.Row attribute), 319
- REQUIRED\_DATA (pynlpl.formats.folia.SemanticRole attribute), 623
- REQUIRED\_DATA (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- REQUIRED\_DATA (pynlpl.formats.folia.SenseAnnotation attribute), 473
- REQUIRED\_DATA (pynlpl.formats.folia.Sentence attribute), 333
- REQUIRED\_DATA (pynlpl.formats.folia.Sentiment attribute), 588
- REQUIRED\_DATA (pynlpl.formats.folia.SentimentLayer attribute), 705
- REQUIRED\_DATA (pynlpl.formats.folia.Statement attribute), 599
- REQUIRED\_DATA (pynlpl.formats.folia.StatementLayer attribute), 717
- REQUIRED\_DATA (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- REQUIRED\_DATA (pynlpl.formats.folia.Suggestion attribute), 998
- REQUIRED\_DATA (pynlpl.formats.folia.SynsetFeature attribute), 874
- REQUIRED\_DATA (pynlpl.formats.folia.SyntacticUnit attribute), 611
- REQUIRED\_DATA (pynlpl.formats.folia.SyntaxLayer attribute), 729
- REQUIRED\_DATA (pynlpl.formats.folia.Table attribute), 347
- REQUIRED\_DATA (pynlpl.formats.folia.TableHead attribute), 373
- REQUIRED\_DATA (pynlpl.formats.folia.Term attribute), 360
- REQUIRED\_DATA (pynlpl.formats.folia.Text attribute), 386
- REQUIRED\_DATA (pynlpl.formats.folia.TextContent attribute), 495
- REQUIRED\_DATA (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- REQUIRED\_DATA (pynlpl.formats.folia.TextMarkupError attribute), 851
- REQUIRED\_DATA (pynlpl.formats.folia.TextMarkupGap attribute), 808
- REQUIRED\_DATA (pynlpl.formats.folia.TextMarkupString attribute), 819
- REQUIRED\_DATA (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- REQUIRED\_DATA (pynlpl.formats.folia.TimeSegment attribute), 634
- REQUIRED\_DATA (pynlpl.formats.folia.TimingLayer attribute), 752
- REQUIRED\_DATA (pynlpl.formats.folia.Whitespace attribute), 399
- REQUIRED\_DATA (pynlpl.formats.folia.Word attribute), 413
- reset() (pynlpl.formats.giza.IntersectionAlignment method), 1053
- reset() (pynlpl.formats.giza.MultiWordAlignment method), 1053
- reset() (pynlpl.formats.giza.WordAlignment method), 1054
- reset() (pynlpl.formats.taggedata.Taggerdata method), 1055
- reset() (pynlpl.search.AbstractSearch method), 1059
- resolve() (pynlpl.formats.folia.AbstractTextMarkup method), 92
- resolve() (pynlpl.formats.folia.Alignment method), 1014
- resolve() (pynlpl.formats.folia.AlignReference method),



- 1026
- `resolve()` (`pynlpl.formats.folia.Linebreak` method), 222
- `resolve()` (`pynlpl.formats.folia.Reference` method), 313
- `resolve()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 846
- `resolve()` (`pynlpl.formats.folia.TextMarkupError` method), 856
- `resolve()` (`pynlpl.formats.folia.TextMarkupGap` method), 814
- `resolve()` (`pynlpl.formats.folia.TextMarkupString` method), 824
- `resolve()` (`pynlpl.formats.folia.TextMarkupStyle` method), 835
- `resolveword()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 81
- `resolveword()` (`pynlpl.formats.folia.AbstractElement` method), 31
- `resolveword()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 58
- `resolveword()` (`pynlpl.formats.folia.AbstractStructureElement` method), 44
- `resolveword()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 92
- `resolveword()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 69
- `resolveword()` (`pynlpl.formats.folia.ActorFeature` method), 890
- `resolveword()` (`pynlpl.formats.folia.Alignment` method), 1014
- `resolveword()` (`pynlpl.formats.folia.AlignReference` method), 1026
- `resolveword()` (`pynlpl.formats.folia.Alternative` method), 926
- `resolveword()` (`pynlpl.formats.folia.AlternativeLayers` method), 937
- `resolveword()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 901
- `resolveword()` (`pynlpl.formats.folia.Cell` method), 107
- `resolveword()` (`pynlpl.formats.folia.Chunk` method), 524
- `resolveword()` (`pynlpl.formats.folia.ChunkingLayer` method), 653
- `resolveword()` (`pynlpl.formats.folia.CoreferenceChain` method), 535
- `resolveword()` (`pynlpl.formats.folia.CoreferenceLayer` method), 664
- `resolveword()` (`pynlpl.formats.folia.CoreferenceLink` method), 770
- `resolveword()` (`pynlpl.formats.folia.Correction` method), 949
- `resolveword()` (`pynlpl.formats.folia.Current` method), 959
- `resolveword()` (`pynlpl.formats.folia.Definition` method), 120
- `resolveword()` (`pynlpl.formats.folia.DependenciesLayer` method), 676
- `resolveword()` (`pynlpl.formats.folia.Dependency` method), 547
- `resolveword()` (`pynlpl.formats.folia.DependencyDependent` method), 782
- `resolveword()` (`pynlpl.formats.folia.Description` method), 1037
- `resolveword()` (`pynlpl.formats.folia.Division` method), 133
- `resolveword()` (`pynlpl.formats.folia.DomainAnnotation` method), 434
- `resolveword()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 912
- `resolveword()` (`pynlpl.formats.folia.EntitiesLayer` method), 688
- `resolveword()` (`pynlpl.formats.folia.Entity` method), 559
- `resolveword()` (`pynlpl.formats.folia.Entry` method), 146
- `resolveword()` (`pynlpl.formats.folia.ErrorDetection` method), 970
- `resolveword()` (`pynlpl.formats.folia.Event` method), 159
- `resolveword()` (`pynlpl.formats.folia.Example` method), 172
- `resolveword()` (`pynlpl.formats.folia.Feature` method), 868
- `resolveword()` (`pynlpl.formats.folia.Figure` method), 185
- `resolveword()` (`pynlpl.formats.folia.Gap` method), 196
- `resolveword()` (`pynlpl.formats.folia.Head` method), 209
- `resolveword()` (`pynlpl.formats.folia.Headspan` method), 794
- `resolveword()` (`pynlpl.formats.folia.LangAnnotation` method), 456
- `resolveword()` (`pynlpl.formats.folia.LemmaAnnotation` method), 467
- `resolveword()` (`pynlpl.formats.folia.Linebreak` method), 222
- `resolveword()` (`pynlpl.formats.folia.List` method), 235
- `resolveword()` (`pynlpl.formats.folia.ListItem` method), 248
- `resolveword()` (`pynlpl.formats.folia.Metric` method), 1048
- `resolveword()` (`pynlpl.formats.folia.New` method), 981
- `resolveword()` (`pynlpl.formats.folia.Note` method), 261
- `resolveword()` (`pynlpl.formats.folia.Observation` method), 570
- `resolveword()` (`pynlpl.formats.folia.ObservationLayer` method), 700
- `resolveword()` (`pynlpl.formats.folia.Original` method), 992
- `resolveword()` (`pynlpl.formats.folia.Paragraph` method), 274
- `resolveword()` (`pynlpl.formats.folia.Part` method), 287
- `resolveword()` (`pynlpl.formats.folia.PhonContent` method), 510
- `resolveword()` (`pynlpl.formats.folia.PosAnnotation` method), 445
- `resolveword()` (`pynlpl.formats.folia.Predicate` method), 582

- [resolveword\(\) \(pynlpl.formats.folia.Quote method\), 300](#)  
[resolveword\(\) \(pynlpl.formats.folia.Reference method\), 313](#)  
[resolveword\(\) \(pynlpl.formats.folia.Row method\), 326](#)  
[resolveword\(\) \(pynlpl.formats.folia.SemanticRole method\), 629](#)  
[resolveword\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 747](#)  
[resolveword\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 478](#)  
[resolveword\(\) \(pynlpl.formats.folia.Sentence method\), 341](#)  
[resolveword\(\) \(pynlpl.formats.folia.Sentiment method\), 594](#)  
[resolveword\(\) \(pynlpl.formats.folia.SentimentLayer method\), 711](#)  
[resolveword\(\) \(pynlpl.formats.folia.Statement method\), 605](#)  
[resolveword\(\) \(pynlpl.formats.folia.StatementLayer method\), 723](#)  
[resolveword\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 489](#)  
[resolveword\(\) \(pynlpl.formats.folia.Suggestion method\), 1003](#)  
[resolveword\(\) \(pynlpl.formats.folia.SynsetFeature method\), 879](#)  
[resolveword\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 617](#)  
[resolveword\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 735](#)  
[resolveword\(\) \(pynlpl.formats.folia.Table method\), 354](#)  
[resolveword\(\) \(pynlpl.formats.folia.TableHead method\), 380](#)  
[resolveword\(\) \(pynlpl.formats.folia.Term method\), 367](#)  
[resolveword\(\) \(pynlpl.formats.folia.Text method\), 393](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextContent method\), 500](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextMarkupCorrection method\), 846](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextMarkupError method\), 856](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextMarkupGap method\), 814](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextMarkupString method\), 824](#)  
[resolveword\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 835](#)  
[resolveword\(\) \(pynlpl.formats.folia.TimeSegment method\), 640](#)  
[resolveword\(\) \(pynlpl.formats.folia.TimingLayer method\), 758](#)  
[resolveword\(\) \(pynlpl.formats.folia.Whitespace method\), 406](#)  
[resolveword\(\) \(pynlpl.formats.folia.Word method\), 421](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 81](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractElement method\), 31](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 58](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 44](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractTextMarkup method\), 92](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AbstractTokenAnnotation method\), 69](#)  
[rightcontext\(\) \(pynlpl.formats.folia.ActorFeature method\), 890](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Alignment method\), 1014](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AlignReference method\), 1026](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Alternative method\), 926](#)  
[rightcontext\(\) \(pynlpl.formats.folia.AlternativeLayers method\), 937](#)  
[rightcontext\(\) \(pynlpl.formats.folia.BeginatetimeFeature method\), 901](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Cell method\), 107](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Chunk method\), 524](#)  
[rightcontext\(\) \(pynlpl.formats.folia.ChunkingLayer method\), 653](#)  
[rightcontext\(\) \(pynlpl.formats.folia.CoreferenceChain method\), 535](#)  
[rightcontext\(\) \(pynlpl.formats.folia.CoreferenceLayer method\), 664](#)  
[rightcontext\(\) \(pynlpl.formats.folia.CoreferenceLink method\), 770](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Correction method\), 949](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Current method\), 959](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Definition method\), 120](#)  
[rightcontext\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 676](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Dependency method\), 547](#)  
[rightcontext\(\) \(pynlpl.formats.folia.DependencyDependent method\), 782](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Description method\), 1037](#)  
[rightcontext\(\) \(pynlpl.formats.folia.Division method\), 133](#)  
[rightcontext\(\) \(pynlpl.formats.folia.DomainAnnotation method\), 434](#)  
[rightcontext\(\) \(pynlpl.formats.folia.EnddatetimeFeature method\), 912](#)  
[rightcontext\(\) \(pynlpl.formats.folia.EntitiesLayer](#)

method), 688  
rightcontext() (pynlpl.formats.folia.Entity method), 559  
rightcontext() (pynlpl.formats.folia.Entry method), 146  
rightcontext() (pynlpl.formats.folia.ErrorDetection method), 970  
rightcontext() (pynlpl.formats.folia.Event method), 159  
rightcontext() (pynlpl.formats.folia.Example method), 172  
rightcontext() (pynlpl.formats.folia.Feature method), 868  
rightcontext() (pynlpl.formats.folia.Figure method), 185  
rightcontext() (pynlpl.formats.folia.Gap method), 196  
rightcontext() (pynlpl.formats.folia.Head method), 209  
rightcontext() (pynlpl.formats.folia.Headspar method), 794  
rightcontext() (pynlpl.formats.folia.LangAnnotation method), 456  
rightcontext() (pynlpl.formats.folia.LemmaAnnotation method), 467  
rightcontext() (pynlpl.formats.folia.Linebreak method), 222  
rightcontext() (pynlpl.formats.folia.List method), 235  
rightcontext() (pynlpl.formats.folia.ListItem method), 248  
rightcontext() (pynlpl.formats.folia.Metric method), 1048  
rightcontext() (pynlpl.formats.folia.New method), 981  
rightcontext() (pynlpl.formats.folia.Note method), 261  
rightcontext() (pynlpl.formats.folia.Observation method), 570  
rightcontext() (pynlpl.formats.folia.ObservationLayer method), 700  
rightcontext() (pynlpl.formats.folia.Original method), 992  
rightcontext() (pynlpl.formats.folia.Paragraph method), 274  
rightcontext() (pynlpl.formats.folia.Part method), 287  
rightcontext() (pynlpl.formats.folia.PhonContent method), 511  
rightcontext() (pynlpl.formats.folia.PosAnnotation method), 445  
rightcontext() (pynlpl.formats.folia.Predicate method), 582  
rightcontext() (pynlpl.formats.folia.Quote method), 300  
rightcontext() (pynlpl.formats.folia.Reference method), 313  
rightcontext() (pynlpl.formats.folia.Row method), 326  
rightcontext() (pynlpl.formats.folia.SemanticRole method), 629  
rightcontext() (pynlpl.formats.folia.SemanticRolesLayer method), 747  
rightcontext() (pynlpl.formats.folia.SenseAnnotation method), 478  
rightcontext() (pynlpl.formats.folia.Sentence method), 341  
rightcontext() (pynlpl.formats.folia.Sentiment method), 594  
rightcontext() (pynlpl.formats.folia.SentimentLayer method), 711  
rightcontext() (pynlpl.formats.folia.Statement method), 605  
rightcontext() (pynlpl.formats.folia.StatementLayer method), 723  
rightcontext() (pynlpl.formats.folia.SubjectivityAnnotation method), 489  
rightcontext() (pynlpl.formats.folia.Suggestion method), 1003  
rightcontext() (pynlpl.formats.folia.SynsetFeature method), 879  
rightcontext() (pynlpl.formats.folia.SyntacticUnit method), 617  
rightcontext() (pynlpl.formats.folia.SyntaxLayer method), 735  
rightcontext() (pynlpl.formats.folia.Table method), 354  
rightcontext() (pynlpl.formats.folia.TableHead method), 380  
rightcontext() (pynlpl.formats.folia.Term method), 367  
rightcontext() (pynlpl.formats.folia.Text method), 393  
rightcontext() (pynlpl.formats.folia.TextContent method), 501  
rightcontext() (pynlpl.formats.folia.TextMarkupCorrection method), 846  
rightcontext() (pynlpl.formats.folia.TextMarkupError method), 856  
rightcontext() (pynlpl.formats.folia.TextMarkupGap method), 814  
rightcontext() (pynlpl.formats.folia.TextMarkupString method), 824  
rightcontext() (pynlpl.formats.folia.TextMarkupStyle method), 835  
rightcontext() (pynlpl.formats.folia.TimeSegment method), 640  
rightcontext() (pynlpl.formats.folia.TimingLayer method), 758  
rightcontext() (pynlpl.formats.folia.Whitespace method), 406  
rightcontext() (pynlpl.formats.folia.Word method), 421  
rmse() (in module pynlpl.evaluation), 11  
rmse() (pynlpl.evaluation.OrdinalEvaluation method), 10  
root() (pynlpl.datatypes.Trie method), 7  
Row (class in pynlpl.formats.folia), 316  
run() (pynlpl.evaluation.AbstractExperiment method), 9  
run() (pynlpl.evaluation.ExperimentPool method), 10

## S

sample() (pynlpl.evaluation.AbstractExperiment method), 9  
save() (pynlpl.formats.folia.Document method), 20  
save() (pynlpl.formats.sonar.CorporaDocumentX method), 1054



- save() (pynlpl.lm.lm.SimpleLanguageModel method), 1057
- save() (pynlpl.statistics.FrequencyList method), 1063
- score() (pynlpl.datatypes.PriorityQueue method), 6
- score() (pynlpl.evaluation.AbstractExperiment method), 9
- score() (pynlpl.lm.lm.ARPALanguageModel method), 1057
- score() (pynlpl.search.AbstractSearchState method), 1060
- scoresentence() (pynlpl.lm.client.LMClient method), 1058
- scoresentence() (pynlpl.lm.lm.SimpleLanguageModel method), 1057
- scoresentence() (pynlpl.lm.srlm.SRILM method), 1057
- scoreword() (pynlpl.lm.lm.ARPALanguageModel method), 1057
- searchall() (pynlpl.search.AbstractSearch method), 1059
- searchbest() (pynlpl.evaluation.WSPParamSearch method), 10
- searchbest() (pynlpl.search.AbstractSearch method), 1059
- searchfirst() (pynlpl.search.AbstractSearch method), 1059
- searchlast() (pynlpl.search.AbstractSearch method), 1059
- searchtop() (pynlpl.search.AbstractSearch method), 1059
- select() (pynlpl.formats.folia.AbstractAnnotationLayer method), 81
- select() (pynlpl.formats.folia.AbstractElement method), 31
- select() (pynlpl.formats.folia.AbstractSpanAnnotation method), 58
- select() (pynlpl.formats.folia.AbstractStructureElement method), 44
- select() (pynlpl.formats.folia.AbstractTextMarkup method), 92
- select() (pynlpl.formats.folia.AbstractTokenAnnotation method), 69
- select() (pynlpl.formats.folia.ActorFeature method), 890
- select() (pynlpl.formats.folia.Alignment method), 1014
- select() (pynlpl.formats.folia.AlignReference method), 1026
- select() (pynlpl.formats.folia.Alternative method), 926
- select() (pynlpl.formats.folia.AlternativeLayers method), 937
- select() (pynlpl.formats.folia.BegindatetimeFeature method), 901
- select() (pynlpl.formats.folia.Cell method), 107
- select() (pynlpl.formats.folia.Chunk method), 524
- select() (pynlpl.formats.folia.ChunkingLayer method), 653
- select() (pynlpl.formats.folia.CoreferenceChain method), 535
- select() (pynlpl.formats.folia.CoreferenceLayer method), 664
- select() (pynlpl.formats.folia.CoreferenceLink method), 770
- select() (pynlpl.formats.folia.Correction method), 949
- select() (pynlpl.formats.folia.Current method), 959
- select() (pynlpl.formats.folia.Definition method), 120
- select() (pynlpl.formats.folia.DependenciesLayer method), 676
- select() (pynlpl.formats.folia.Dependency method), 547
- select() (pynlpl.formats.folia.DependencyDependent method), 782
- select() (pynlpl.formats.folia.Description method), 1037
- select() (pynlpl.formats.folia.Division method), 133
- select() (pynlpl.formats.folia.Document method), 20
- select() (pynlpl.formats.folia.DomainAnnotation method), 434
- select() (pynlpl.formats.folia.EnddatetimeFeature method), 912
- select() (pynlpl.formats.folia.EntitiesLayer method), 688
- select() (pynlpl.formats.folia.Entity method), 559
- select() (pynlpl.formats.folia.Entry method), 146
- select() (pynlpl.formats.folia.ErrorDetection method), 970
- select() (pynlpl.formats.folia.Event method), 159
- select() (pynlpl.formats.folia.Example method), 172
- select() (pynlpl.formats.folia.Feature method), 868
- select() (pynlpl.formats.folia.Figure method), 185
- select() (pynlpl.formats.folia.Gap method), 196
- select() (pynlpl.formats.folia.Head method), 209
- select() (pynlpl.formats.folia.Headsplan method), 794
- select() (pynlpl.formats.folia.LangAnnotation method), 456
- select() (pynlpl.formats.folia.LemmaAnnotation method), 467
- select() (pynlpl.formats.folia.Linebreak method), 222
- select() (pynlpl.formats.folia.List method), 235
- select() (pynlpl.formats.folia.ListItem method), 248
- select() (pynlpl.formats.folia.Metric method), 1048
- select() (pynlpl.formats.folia.New method), 981
- select() (pynlpl.formats.folia.Note method), 261
- select() (pynlpl.formats.folia.Observation method), 570
- select() (pynlpl.formats.folia.ObservationLayer method), 700
- select() (pynlpl.formats.folia.Original method), 992
- select() (pynlpl.formats.folia.Paragraph method), 274
- select() (pynlpl.formats.folia.Part method), 287
- select() (pynlpl.formats.folia.PhonContent method), 511
- select() (pynlpl.formats.folia.PosAnnotation method), 445
- select() (pynlpl.formats.folia.Predicate method), 582
- select() (pynlpl.formats.folia.Quote method), 300
- select() (pynlpl.formats.folia.Reference method), 313
- select() (pynlpl.formats.folia.Row method), 326
- select() (pynlpl.formats.folia.SemanticRole method), 629
- select() (pynlpl.formats.folia.SemanticRolesLayer method), 747

`select()` (`pynlpl.formats.folia.SenseAnnotation` method), 478

`select()` (`pynlpl.formats.folia.Sentence` method), 341

`select()` (`pynlpl.formats.folia.Sentiment` method), 594

`select()` (`pynlpl.formats.folia.SentimentLayer` method), 712

`select()` (`pynlpl.formats.folia.Statement` method), 605

`select()` (`pynlpl.formats.folia.StatementLayer` method), 723

`select()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 489

`select()` (`pynlpl.formats.folia.Suggestion` method), 1003

`select()` (`pynlpl.formats.folia.SynsetFeature` method), 879

`select()` (`pynlpl.formats.folia.SyntacticUnit` method), 617

`select()` (`pynlpl.formats.folia.SyntaxLayer` method), 735

`select()` (`pynlpl.formats.folia.Table` method), 354

`select()` (`pynlpl.formats.folia.TableHead` method), 380

`select()` (`pynlpl.formats.folia.Term` method), 367

`select()` (`pynlpl.formats.folia.Text` method), 393

`select()` (`pynlpl.formats.folia.TextContent` method), 501

`select()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 846

`select()` (`pynlpl.formats.folia.TextMarkupError` method), 856

`select()` (`pynlpl.formats.folia.TextMarkupGap` method), 814

`select()` (`pynlpl.formats.folia.TextMarkupString` method), 824

`select()` (`pynlpl.formats.folia.TextMarkupStyle` method), 835

`select()` (`pynlpl.formats.folia.TimeSegment` method), 640

`select()` (`pynlpl.formats.folia.TimingLayer` method), 759

`select()` (`pynlpl.formats.folia.Whitespace` method), 406

`select()` (`pynlpl.formats.folia.Word` method), 421

`SemanticRole` (class in `pynlpl.formats.folia`), 620

`SemanticRolesLayer` (class in `pynlpl.formats.folia`), 738

`sense()` (`pynlpl.formats.folia.Word` method), 422

`SenseAnnotation` (class in `pynlpl.formats.folia`), 470

`Sentence` (class in `pynlpl.formats.folia`), 329

`sentence()` (`pynlpl.formats.folia.Word` method), 422

`sentences()` (`pynlpl.formats.folia.AbstractStructureElement` method), 45

`sentences()` (`pynlpl.formats.folia.Cell` method), 108

`sentences()` (`pynlpl.formats.folia.Definition` method), 121

`sentences()` (`pynlpl.formats.folia.Division` method), 134

`sentences()` (`pynlpl.formats.folia.Document` method), 20

`sentences()` (`pynlpl.formats.folia.Entry` method), 147

`sentences()` (`pynlpl.formats.folia.Event` method), 160

`sentences()` (`pynlpl.formats.folia.Example` method), 173

`sentences()` (`pynlpl.formats.folia.Figure` method), 186

`sentences()` (`pynlpl.formats.folia.Head` method), 210

`sentences()` (`pynlpl.formats.folia.Linebreak` method), 223

`sentences()` (`pynlpl.formats.folia.List` method), 236

`sentences()` (`pynlpl.formats.folia.ListItem` method), 249

`sentences()` (`pynlpl.formats.folia.Note` method), 262

`sentences()` (`pynlpl.formats.folia.Paragraph` method), 275

`sentences()` (`pynlpl.formats.folia.Part` method), 288

`sentences()` (`pynlpl.formats.folia.Quote` method), 301

`sentences()` (`pynlpl.formats.folia.Reference` method), 314

`sentences()` (`pynlpl.formats.folia.Row` method), 327

`sentences()` (`pynlpl.formats.folia.Sentence` method), 341

`sentences()` (`pynlpl.formats.folia.Table` method), 355

`sentences()` (`pynlpl.formats.folia.TableHead` method), 381

`sentences()` (`pynlpl.formats.folia.Term` method), 368

`sentences()` (`pynlpl.formats.folia.Text` method), 394

`sentences()` (`pynlpl.formats.folia.Whitespace` method), 407

`sentences()` (`pynlpl.formats.folia.Word` method), 422

`sentences()` (`pynlpl.formats.sonar.CorpusDocument` method), 1054

`sentences()` (`pynlpl.formats.sonar.CorpusDocumentX` method), 1054

`Sentiment` (class in `pynlpl.formats.folia`), 585

`SentimentLayer` (class in `pynlpl.formats.folia`), 703

`sequence()` (`pynlpl.datatypes.Trie` method), 7

`setdoc()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 81

`setdoc()` (`pynlpl.formats.folia.AbstractElement` method), 32

`setdoc()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 58

`setdoc()` (`pynlpl.formats.folia.AbstractStructureElement` method), 45

`setdoc()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 92

`setdoc()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 69

`setdoc()` (`pynlpl.formats.folia.ActorFeature` method), 891

`setdoc()` (`pynlpl.formats.folia.Alignment` method), 1015

`setdoc()` (`pynlpl.formats.folia.AlignReference` method), 1026

`setdoc()` (`pynlpl.formats.folia.Alternative` method), 926

`setdoc()` (`pynlpl.formats.folia.AlternativeLayers` method), 937

`setdoc()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 902

`setdoc()` (`pynlpl.formats.folia.Cell` method), 108

`setdoc()` (`pynlpl.formats.folia.Chunk` method), 524

`setdoc()` (`pynlpl.formats.folia.ChunkingLayer` method), 653

`setdoc()` (`pynlpl.formats.folia.CoreferenceChain` method), 536

`setdoc()` (`pynlpl.formats.folia.CoreferenceLayer` method), 665

`setdoc()` (`pynlpl.formats.folia.CoreferenceLink` method), 771

`setdoc()` (`pynlpl.formats.folia.Correction` method), 949

`setdoc()` (`pynlpl.formats.folia.Current` method), 960

- [setdoc\(\) \(pynlpl.formats.folia.Definition method\), 121](#)  
[setdoc\(\) \(pynlpl.formats.folia.DependenciesLayer method\), 677](#)  
[setdoc\(\) \(pynlpl.formats.folia.Dependency method\), 548](#)  
[setdoc\(\) \(pynlpl.formats.folia.DependencyDependent method\), 783](#)  
[setdoc\(\) \(pynlpl.formats.folia.Description method\), 1038](#)  
[setdoc\(\) \(pynlpl.formats.folia.Division method\), 134](#)  
[setdoc\(\) \(pynlpl.formats.folia.DomainAnnotation method\), 435](#)  
[setdoc\(\) \(pynlpl.formats.folia.EnddatetimeFeature method\), 913](#)  
[setdoc\(\) \(pynlpl.formats.folia.EntitiesLayer method\), 689](#)  
[setdoc\(\) \(pynlpl.formats.folia.Entity method\), 559](#)  
[setdoc\(\) \(pynlpl.formats.folia.Entry method\), 147](#)  
[setdoc\(\) \(pynlpl.formats.folia.ErrorDetection method\), 971](#)  
[setdoc\(\) \(pynlpl.formats.folia.Event method\), 160](#)  
[setdoc\(\) \(pynlpl.formats.folia.Example method\), 173](#)  
[setdoc\(\) \(pynlpl.formats.folia.Feature method\), 869](#)  
[setdoc\(\) \(pynlpl.formats.folia.Figure method\), 186](#)  
[setdoc\(\) \(pynlpl.formats.folia.Gap method\), 197](#)  
[setdoc\(\) \(pynlpl.formats.folia.Head method\), 210](#)  
[setdoc\(\) \(pynlpl.formats.folia.Headspan method\), 794](#)  
[setdoc\(\) \(pynlpl.formats.folia.LangAnnotation method\), 457](#)  
[setdoc\(\) \(pynlpl.formats.folia.LemmaAnnotation method\), 468](#)  
[setdoc\(\) \(pynlpl.formats.folia.Linebreak method\), 223](#)  
[setdoc\(\) \(pynlpl.formats.folia.List method\), 236](#)  
[setdoc\(\) \(pynlpl.formats.folia.ListItem method\), 249](#)  
[setdoc\(\) \(pynlpl.formats.folia.Metric method\), 1049](#)  
[setdoc\(\) \(pynlpl.formats.folia.New method\), 982](#)  
[setdoc\(\) \(pynlpl.formats.folia.Note method\), 262](#)  
[setdoc\(\) \(pynlpl.formats.folia.Observation method\), 571](#)  
[setdoc\(\) \(pynlpl.formats.folia.ObservationLayer method\), 700](#)  
[setdoc\(\) \(pynlpl.formats.folia.Original method\), 993](#)  
[setdoc\(\) \(pynlpl.formats.folia.Paragraph method\), 275](#)  
[setdoc\(\) \(pynlpl.formats.folia.Part method\), 288](#)  
[setdoc\(\) \(pynlpl.formats.folia.PhonContent method\), 511](#)  
[setdoc\(\) \(pynlpl.formats.folia.PosAnnotation method\), 446](#)  
[setdoc\(\) \(pynlpl.formats.folia.Predicate method\), 583](#)  
[setdoc\(\) \(pynlpl.formats.folia.Quote method\), 301](#)  
[setdoc\(\) \(pynlpl.formats.folia.Reference method\), 314](#)  
[setdoc\(\) \(pynlpl.formats.folia.Row method\), 327](#)  
[setdoc\(\) \(pynlpl.formats.folia.SemanticRole method\), 629](#)  
[setdoc\(\) \(pynlpl.formats.folia.SemanticRolesLayer method\), 747](#)  
[setdoc\(\) \(pynlpl.formats.folia.SenseAnnotation method\), 479](#)  
[setdoc\(\) \(pynlpl.formats.folia.Sentence method\), 342](#)  
[setdoc\(\) \(pynlpl.formats.folia.Sentiment method\), 594](#)  
[setdoc\(\) \(pynlpl.formats.folia.SentimentLayer method\), 712](#)  
[setdoc\(\) \(pynlpl.formats.folia.Statement method\), 606](#)  
[setdoc\(\) \(pynlpl.formats.folia.StatementLayer method\), 724](#)  
[setdoc\(\) \(pynlpl.formats.folia.SubjectivityAnnotation method\), 490](#)  
[setdoc\(\) \(pynlpl.formats.folia.Suggestion method\), 1004](#)  
[setdoc\(\) \(pynlpl.formats.folia.SynsetFeature method\), 880](#)  
[setdoc\(\) \(pynlpl.formats.folia.SyntacticUnit method\), 618](#)  
[setdoc\(\) \(pynlpl.formats.folia.SyntaxLayer method\), 736](#)  
[setdoc\(\) \(pynlpl.formats.folia.Table method\), 355](#)  
[setdoc\(\) \(pynlpl.formats.folia.TableHead method\), 381](#)  
[setdoc\(\) \(pynlpl.formats.folia.Term method\), 368](#)  
[setdoc\(\) \(pynlpl.formats.folia.Text method\), 394](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextContent method\), 501](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextMarkupCorrection method\), 846](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextMarkupError method\), 857](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextMarkupGap method\), 814](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextMarkupString method\), 825](#)  
[setdoc\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 836](#)  
[setdoc\(\) \(pynlpl.formats.folia.TimeSegment method\), 641](#)  
[setdoc\(\) \(pynlpl.formats.folia.TimingLayer method\), 759](#)  
[setdoc\(\) \(pynlpl.formats.folia.Whitespace method\), 407](#)  
[setdoc\(\) \(pynlpl.formats.folia.Word method\), 422](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractAnnotationLayer method\), 81](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractElement method\), 32](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractSpanAnnotation method\), 58](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractStructureElement method\), 45](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractTextMarkup method\), 92](#)  
[setdocument\(\) \(pynlpl.formats.folia.AbstractTokenAnnotation method\), 69](#)  
[setdocument\(\) \(pynlpl.formats.folia.ActorFeature method\), 891](#)  
[setdocument\(\) \(pynlpl.formats.folia.Alignment method\), 1015](#)  
[setdocument\(\) \(pynlpl.formats.folia.AlignReference method\), 1026](#)  
[setdocument\(\) \(pynlpl.formats.folia.Alternative method\), 926](#)  
[setdocument\(\) \(pynlpl.formats.folia.AlternativeLayers method\), 937](#)

`setdocument()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 902

`setdocument()` (`pynlpl.formats.folia.Cell` method), 108

`setdocument()` (`pynlpl.formats.folia.Chunk` method), 524

`setdocument()` (`pynlpl.formats.folia.ChunkingLayer` method), 653

`setdocument()` (`pynlpl.formats.folia.CoreferenceChain` method), 536

`setdocument()` (`pynlpl.formats.folia.CoreferenceLayer` method), 665

`setdocument()` (`pynlpl.formats.folia.CoreferenceLink` method), 771

`setdocument()` (`pynlpl.formats.folia.Correction` method), 949

`setdocument()` (`pynlpl.formats.folia.Current` method), 960

`setdocument()` (`pynlpl.formats.folia.Definition` method), 121

`setdocument()` (`pynlpl.formats.folia.DependenciesLayer` method), 677

`setdocument()` (`pynlpl.formats.folia.Dependency` method), 548

`setdocument()` (`pynlpl.formats.folia.DependencyDependent` method), 783

`setdocument()` (`pynlpl.formats.folia.Description` method), 1038

`setdocument()` (`pynlpl.formats.folia.Division` method), 134

`setdocument()` (`pynlpl.formats.folia.DomainAnnotation` method), 435

`setdocument()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 913

`setdocument()` (`pynlpl.formats.folia.EntitiesLayer` method), 689

`setdocument()` (`pynlpl.formats.folia.Entity` method), 559

`setdocument()` (`pynlpl.formats.folia.Entry` method), 147

`setdocument()` (`pynlpl.formats.folia.ErrorDetection` method), 971

`setdocument()` (`pynlpl.formats.folia.Event` method), 160

`setdocument()` (`pynlpl.formats.folia.Example` method), 173

`setdocument()` (`pynlpl.formats.folia.Feature` method), 869

`setdocument()` (`pynlpl.formats.folia.Figure` method), 186

`setdocument()` (`pynlpl.formats.folia.Gap` method), 197

`setdocument()` (`pynlpl.formats.folia.Head` method), 210

`setdocument()` (`pynlpl.formats.folia.Headspan` method), 794

`setdocument()` (`pynlpl.formats.folia.LangAnnotation` method), 457

`setdocument()` (`pynlpl.formats.folia.LemmaAnnotation` method), 468

`setdocument()` (`pynlpl.formats.folia.Linebreak` method), 223

`setdocument()` (`pynlpl.formats.folia.List` method), 236

`setdocument()` (`pynlpl.formats.folia.ListItem` method), 249

`setdocument()` (`pynlpl.formats.folia.Metric` method), 1049

`setdocument()` (`pynlpl.formats.folia.New` method), 982

`setdocument()` (`pynlpl.formats.folia.Note` method), 262

`setdocument()` (`pynlpl.formats.folia.Observation` method), 571

`setdocument()` (`pynlpl.formats.folia.ObservationLayer` method), 700

`setdocument()` (`pynlpl.formats.folia.Original` method), 993

`setdocument()` (`pynlpl.formats.folia.Paragraph` method), 275

`setdocument()` (`pynlpl.formats.folia.Part` method), 288

`setdocument()` (`pynlpl.formats.folia.PhonContent` method), 511

`setdocument()` (`pynlpl.formats.folia.PosAnnotation` method), 446

`setdocument()` (`pynlpl.formats.folia.Predicate` method), 583

`setdocument()` (`pynlpl.formats.folia.Quote` method), 301

`setdocument()` (`pynlpl.formats.folia.Reference` method), 314

`setdocument()` (`pynlpl.formats.folia.Row` method), 327

`setdocument()` (`pynlpl.formats.folia.SemanticRole` method), 629

`setdocument()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 747

`setdocument()` (`pynlpl.formats.folia.SenseAnnotation` method), 479

`setdocument()` (`pynlpl.formats.folia.Sentence` method), 342

`setdocument()` (`pynlpl.formats.folia.Sentiment` method), 594

`setdocument()` (`pynlpl.formats.folia.SentimentLayer` method), 712

`setdocument()` (`pynlpl.formats.folia.Statement` method), 606

`setdocument()` (`pynlpl.formats.folia.StatementLayer` method), 724

`setdocument()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 490

`setdocument()` (`pynlpl.formats.folia.Suggestion` method), 1004

`setdocument()` (`pynlpl.formats.folia.SynsetFeature` method), 880

`setdocument()` (`pynlpl.formats.folia.SyntacticUnit` method), 618

`setdocument()` (`pynlpl.formats.folia.SyntaxLayer` method), 736

`setdocument()` (`pynlpl.formats.folia.Table` method), 355

`setdocument()` (`pynlpl.formats.folia.TableHead` method), 381

`setdocument()` (`pynlpl.formats.folia.Term` method), 368

- setdocument() (pynlpl.formats.folia.Text method), 394  
 setdocument() (pynlpl.formats.folia.TextContent method), 501  
 setdocument() (pynlpl.formats.folia.TextMarkupCorrection method), 846  
 setdocument() (pynlpl.formats.folia.TextMarkupError method), 857  
 setdocument() (pynlpl.formats.folia.TextMarkupGap method), 814  
 setdocument() (pynlpl.formats.folia.TextMarkupString method), 825  
 setdocument() (pynlpl.formats.folia.TextMarkupStyle method), 836  
 setdocument() (pynlpl.formats.folia.TimeSegment method), 641  
 setdocument() (pynlpl.formats.folia.TimingLayer method), 759  
 setdocument() (pynlpl.formats.folia.Whitespace method), 407  
 setdocument() (pynlpl.formats.folia.Word method), 422  
 setemission() (pynlpl.statistics.HiddenMarkovModel method), 1063  
 setimdi() (pynlpl.formats.folia.Document method), 20  
 SETONLY (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 75  
 SETONLY (pynlpl.formats.folia.AbstractElement attribute), 26  
 SETONLY (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52  
 SETONLY (pynlpl.formats.folia.AbstractStructureElement attribute), 37  
 SETONLY (pynlpl.formats.folia.AbstractTextMarkup attribute), 86  
 SETONLY (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63  
 SETONLY (pynlpl.formats.folia.ActorFeature attribute), 885  
 SETONLY (pynlpl.formats.folia.Alignment attribute), 1009  
 SETONLY (pynlpl.formats.folia.AlignReference attribute), 1020  
 SETONLY (pynlpl.formats.folia.Alternative attribute), 919  
 SETONLY (pynlpl.formats.folia.AlternativeLayers attribute), 931  
 SETONLY (pynlpl.formats.folia.BegindatetimeFeature attribute), 896  
 SETONLY (pynlpl.formats.folia.Cell attribute), 100  
 SETONLY (pynlpl.formats.folia.Chunk attribute), 518  
 SETONLY (pynlpl.formats.folia.ChunkingLayer attribute), 647  
 SETONLY (pynlpl.formats.folia.CoreferenceChain attribute), 530  
 SETONLY (pynlpl.formats.folia.CoreferenceLayer attribute), 658  
 SETONLY (pynlpl.formats.folia.CoreferenceLink attribute), 764  
 SETONLY (pynlpl.formats.folia.Correction attribute), 944  
 SETONLY (pynlpl.formats.folia.Current attribute), 953  
 SETONLY (pynlpl.formats.folia.Definition attribute), 113  
 SETONLY (pynlpl.formats.folia.DependenciesLayer attribute), 670  
 SETONLY (pynlpl.formats.folia.Dependency attribute), 541  
 SETONLY (pynlpl.formats.folia.DependencyDependent attribute), 776  
 SETONLY (pynlpl.formats.folia.Description attribute), 1031  
 SETONLY (pynlpl.formats.folia.Division attribute), 126  
 SETONLY (pynlpl.formats.folia.DomainAnnotation attribute), 429  
 SETONLY (pynlpl.formats.folia.EnddatetimeFeature attribute), 907  
 SETONLY (pynlpl.formats.folia.EntitiesLayer attribute), 682  
 SETONLY (pynlpl.formats.folia.Entity attribute), 553  
 SETONLY (pynlpl.formats.folia.Entry attribute), 139  
 SETONLY (pynlpl.formats.folia.ErrorDetection attribute), 965  
 SETONLY (pynlpl.formats.folia.Event attribute), 152  
 SETONLY (pynlpl.formats.folia.Example attribute), 165  
 SETONLY (pynlpl.formats.folia.Feature attribute), 862  
 SETONLY (pynlpl.formats.folia.Figure attribute), 178  
 SETONLY (pynlpl.formats.folia.Gap attribute), 191  
 SETONLY (pynlpl.formats.folia.Head attribute), 202  
 SETONLY (pynlpl.formats.folia.Headspan attribute), 788  
 SETONLY (pynlpl.formats.folia.LangAnnotation attribute), 451  
 SETONLY (pynlpl.formats.folia.LemmaAnnotation attribute), 462  
 SETONLY (pynlpl.formats.folia.Linebreak attribute), 215  
 SETONLY (pynlpl.formats.folia.List attribute), 228  
 SETONLY (pynlpl.formats.folia.ListItem attribute), 241  
 SETONLY (pynlpl.formats.folia.Metric attribute), 1043  
 SETONLY (pynlpl.formats.folia.New attribute), 976  
 SETONLY (pynlpl.formats.folia.Note attribute), 254  
 SETONLY (pynlpl.formats.folia.Observation attribute), 564  
 SETONLY (pynlpl.formats.folia.ObservationLayer attribute), 694  
 SETONLY (pynlpl.formats.folia.Original attribute), 987  
 SETONLY (pynlpl.formats.folia.Paragraph attribute), 267  
 SETONLY (pynlpl.formats.folia.Part attribute), 280  
 SETONLY (pynlpl.formats.folia.PhonContent attribute), 506  
 SETONLY (pynlpl.formats.folia.PosAnnotation attribute), 440



SETONLY (pynlpl.formats.folia.Predicate attribute), 576  
SETONLY (pynlpl.formats.folia.Quote attribute), 293  
SETONLY (pynlpl.formats.folia.Reference attribute), 306  
SETONLY (pynlpl.formats.folia.Row attribute), 319  
SETONLY (pynlpl.formats.folia.SemanticRole attribute), 623  
SETONLY (pynlpl.formats.folia.SemanticRolesLayer attribute), 741  
SETONLY (pynlpl.formats.folia.SenseAnnotation attribute), 473  
SETONLY (pynlpl.formats.folia.Sentence attribute), 333  
SETONLY (pynlpl.formats.folia.Sentiment attribute), 588  
SETONLY (pynlpl.formats.folia.SentimentLayer attribute), 705  
SETONLY (pynlpl.formats.folia.Statement attribute), 599  
SETONLY (pynlpl.formats.folia.StatementLayer attribute), 717  
SETONLY (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484  
SETONLY (pynlpl.formats.folia.Suggestion attribute), 998  
SETONLY (pynlpl.formats.folia.SynsetFeature attribute), 874  
SETONLY (pynlpl.formats.folia.SyntacticUnit attribute), 611  
SETONLY (pynlpl.formats.folia.SyntaxLayer attribute), 729  
SETONLY (pynlpl.formats.folia.Table attribute), 347  
SETONLY (pynlpl.formats.folia.TableHead attribute), 373  
SETONLY (pynlpl.formats.folia.Term attribute), 360  
SETONLY (pynlpl.formats.folia.Text attribute), 386  
SETONLY (pynlpl.formats.folia.TextContent attribute), 495  
SETONLY (pynlpl.formats.folia.TextMarkupCorrection attribute), 840  
SETONLY (pynlpl.formats.folia.TextMarkupError attribute), 851  
SETONLY (pynlpl.formats.folia.TextMarkupGap attribute), 808  
SETONLY (pynlpl.formats.folia.TextMarkupString attribute), 819  
SETONLY (pynlpl.formats.folia.TextMarkupStyle attribute), 830  
SETONLY (pynlpl.formats.folia.TimeSegment attribute), 634  
SETONLY (pynlpl.formats.folia.TimingLayer attribute), 752  
SETONLY (pynlpl.formats.folia.Whitespace attribute), 399  
SETONLY (pynlpl.formats.folia.Word attribute), 413  
setparents() (pynlpl.formats.folia.AbstractAnnotationLayer method), 81  
setparents() (pynlpl.formats.folia.AbstractElement method), 32  
setparents() (pynlpl.formats.folia.AbstractSpanAnnotation method), 58  
setparents() (pynlpl.formats.folia.AbstractStructureElement method), 45  
setparents() (pynlpl.formats.folia.AbstractTextMarkup method), 92  
setparents() (pynlpl.formats.folia.AbstractTokenAnnotation method), 70  
setparents() (pynlpl.formats.folia.ActorFeature method), 891  
setparents() (pynlpl.formats.folia.Alignment method), 1015  
setparents() (pynlpl.formats.folia.AlignReference method), 1026  
setparents() (pynlpl.formats.folia.Alternative method), 926  
setparents() (pynlpl.formats.folia.AlternativeLayers method), 937  
setparents() (pynlpl.formats.folia.BegindatetimeFeature method), 902  
setparents() (pynlpl.formats.folia.Cell method), 108  
setparents() (pynlpl.formats.folia.Chunk method), 524  
setparents() (pynlpl.formats.folia.ChunkingLayer method), 653  
setparents() (pynlpl.formats.folia.CoreferenceChain method), 536  
setparents() (pynlpl.formats.folia.CoreferenceLayer method), 665  
setparents() (pynlpl.formats.folia.CoreferenceLink method), 771  
setparents() (pynlpl.formats.folia.Correction method), 949  
setparents() (pynlpl.formats.folia.Current method), 960  
setparents() (pynlpl.formats.folia.Definition method), 121  
setparents() (pynlpl.formats.folia.DependenciesLayer method), 677  
setparents() (pynlpl.formats.folia.Dependency method), 548  
setparents() (pynlpl.formats.folia.DependencyDependent method), 783  
setparents() (pynlpl.formats.folia.Description method), 1038  
setparents() (pynlpl.formats.folia.Division method), 134  
setparents() (pynlpl.formats.folia.DomainAnnotation method), 435  
setparents() (pynlpl.formats.folia.EnddatetimeFeature method), 913  
setparents() (pynlpl.formats.folia.EntitiesLayer method), 689  
setparents() (pynlpl.formats.folia.Entity method), 559  
setparents() (pynlpl.formats.folia.Entry method), 147  
setparents() (pynlpl.formats.folia.ErrorDetection method), 971

- setparents() (pynlpl.formats.folia.Event method), 160
- setparents() (pynlpl.formats.folia.Example method), 173
- setparents() (pynlpl.formats.folia.Feature method), 869
- setparents() (pynlpl.formats.folia.Figure method), 186
- setparents() (pynlpl.formats.folia.Gap method), 197
- setparents() (pynlpl.formats.folia.Head method), 210
- setparents() (pynlpl.formats.folia.Headspan method), 794
- setparents() (pynlpl.formats.folia.LangAnnotation method), 457
- setparents() (pynlpl.formats.folia.LemmaAnnotation method), 468
- setparents() (pynlpl.formats.folia.Linebreak method), 223
- setparents() (pynlpl.formats.folia.List method), 236
- setparents() (pynlpl.formats.folia.ListItem method), 249
- setparents() (pynlpl.formats.folia.Metric method), 1049
- setparents() (pynlpl.formats.folia.New method), 982
- setparents() (pynlpl.formats.folia.Note method), 262
- setparents() (pynlpl.formats.folia.Observation method), 571
- setparents() (pynlpl.formats.folia.ObservationLayer method), 700
- setparents() (pynlpl.formats.folia.Original method), 993
- setparents() (pynlpl.formats.folia.Paragraph method), 275
- setparents() (pynlpl.formats.folia.Part method), 288
- setparents() (pynlpl.formats.folia.PhonContent method), 511
- setparents() (pynlpl.formats.folia.PosAnnotation method), 446
- setparents() (pynlpl.formats.folia.Predicate method), 583
- setparents() (pynlpl.formats.folia.Quote method), 301
- setparents() (pynlpl.formats.folia.Reference method), 314
- setparents() (pynlpl.formats.folia.Row method), 327
- setparents() (pynlpl.formats.folia.SemanticRole method), 629
- setparents() (pynlpl.formats.folia.SemanticRolesLayer method), 747
- setparents() (pynlpl.formats.folia.SenseAnnotation method), 479
- setparents() (pynlpl.formats.folia.Sentence method), 342
- setparents() (pynlpl.formats.folia.Sentiment method), 594
- setparents() (pynlpl.formats.folia.SentimentLayer method), 712
- setparents() (pynlpl.formats.folia.Statement method), 606
- setparents() (pynlpl.formats.folia.StatementLayer method), 724
- setparents() (pynlpl.formats.folia.SubjectivityAnnotation method), 490
- setparents() (pynlpl.formats.folia.Suggestion method), 1004
- setparents() (pynlpl.formats.folia.SynsetFeature method), 880
- setparents() (pynlpl.formats.folia.SyntacticUnit method), 618
- setparents() (pynlpl.formats.folia.SyntaxLayer method), 736
- setparents() (pynlpl.formats.folia.Table method), 355
- setparents() (pynlpl.formats.folia.TableHead method), 381
- setparents() (pynlpl.formats.folia.Term method), 368
- setparents() (pynlpl.formats.folia.Text method), 394
- setparents() (pynlpl.formats.folia.TextContent method), 501
- setparents() (pynlpl.formats.folia.TextMarkupCorrection method), 846
- setparents() (pynlpl.formats.folia.TextMarkupError method), 857
- setparents() (pynlpl.formats.folia.TextMarkupGap method), 814
- setparents() (pynlpl.formats.folia.TextMarkupString method), 825
- setparents() (pynlpl.formats.folia.TextMarkupStyle method), 836
- setparents() (pynlpl.formats.folia.TimeSegment method), 641
- setparents() (pynlpl.formats.folia.TimingLayer method), 759
- setparents() (pynlpl.formats.folia.Whitespace method), 407
- setparents() (pynlpl.formats.folia.Word method), 422
- setphon() (pynlpl.formats.folia.PhonContent method), 511
- setspan() (pynlpl.formats.folia.AbstractSpanAnnotation method), 59
- setspan() (pynlpl.formats.folia.Chunk method), 525
- setspan() (pynlpl.formats.folia.CoreferenceChain method), 536
- setspan() (pynlpl.formats.folia.CoreferenceLink method), 771
- setspan() (pynlpl.formats.folia.Dependency method), 548
- setspan() (pynlpl.formats.folia.DependencyDependent method), 783
- setspan() (pynlpl.formats.folia.Entity method), 559
- setspan() (pynlpl.formats.folia.Headspan method), 794
- setspan() (pynlpl.formats.folia.Observation method), 571
- setspan() (pynlpl.formats.folia.Predicate method), 583
- setspan() (pynlpl.formats.folia.SemanticRole method), 629
- setspan() (pynlpl.formats.folia.Sentiment method), 594
- setspan() (pynlpl.formats.folia.Statement method), 606
- setspan() (pynlpl.formats.folia.SyntacticUnit method), 618
- setspan() (pynlpl.formats.folia.TimeSegment method), 641
- settext() (pynlpl.formats.folia.AbstractAnnotationLayer method), 82
- settext() (pynlpl.formats.folia.AbstractElement method), 32
- settext() (pynlpl.formats.folia.AbstractSpanAnnotation

method), 59  
settext() (pynlpl.formats.folia.AbstractStructureElement method), 45  
settext() (pynlpl.formats.folia.AbstractTextMarkup method), 92  
settext() (pynlpl.formats.folia.AbstractTokenAnnotation method), 70  
settext() (pynlpl.formats.folia.ActorFeature method), 891  
settext() (pynlpl.formats.folia.Alignment method), 1015  
settext() (pynlpl.formats.folia.AlignReference method), 1026  
settext() (pynlpl.formats.folia.Alternative method), 926  
settext() (pynlpl.formats.folia.AlternativeLayers method), 938  
settext() (pynlpl.formats.folia.BegindatetimeFeature method), 902  
settext() (pynlpl.formats.folia.Cell method), 108  
settext() (pynlpl.formats.folia.Chunk method), 525  
settext() (pynlpl.formats.folia.ChunkingLayer method), 653  
settext() (pynlpl.formats.folia.CoreferenceChain method), 536  
settext() (pynlpl.formats.folia.CoreferenceLayer method), 665  
settext() (pynlpl.formats.folia.CoreferenceLink method), 771  
settext() (pynlpl.formats.folia.Correction method), 949  
settext() (pynlpl.formats.folia.Current method), 960  
settext() (pynlpl.formats.folia.Definition method), 121  
settext() (pynlpl.formats.folia.DependenciesLayer method), 677  
settext() (pynlpl.formats.folia.Dependency method), 548  
settext() (pynlpl.formats.folia.DependencyDependent method), 783  
settext() (pynlpl.formats.folia.Description method), 1038  
settext() (pynlpl.formats.folia.Division method), 134  
settext() (pynlpl.formats.folia.DomainAnnotation method), 435  
settext() (pynlpl.formats.folia.EnddatetimeFeature method), 913  
settext() (pynlpl.formats.folia.EntitiesLayer method), 689  
settext() (pynlpl.formats.folia.Entity method), 559  
settext() (pynlpl.formats.folia.Entry method), 147  
settext() (pynlpl.formats.folia.ErrorDetection method), 971  
settext() (pynlpl.formats.folia.Event method), 160  
settext() (pynlpl.formats.folia.Example method), 173  
settext() (pynlpl.formats.folia.Feature method), 869  
settext() (pynlpl.formats.folia.Figure method), 186  
settext() (pynlpl.formats.folia.Gap method), 197  
settext() (pynlpl.formats.folia.Head method), 210  
settext() (pynlpl.formats.folia.Headspan method), 794  
settext() (pynlpl.formats.folia.LangAnnotation method), 457  
settext() (pynlpl.formats.folia.LemmaAnnotation method), 468  
settext() (pynlpl.formats.folia.Linebreak method), 223  
settext() (pynlpl.formats.folia.List method), 236  
settext() (pynlpl.formats.folia.ListItem method), 249  
settext() (pynlpl.formats.folia.Metric method), 1049  
settext() (pynlpl.formats.folia.New method), 982  
settext() (pynlpl.formats.folia.Note method), 262  
settext() (pynlpl.formats.folia.Observation method), 571  
settext() (pynlpl.formats.folia.ObservationLayer method), 701  
settext() (pynlpl.formats.folia.Original method), 993  
settext() (pynlpl.formats.folia.Paragraph method), 275  
settext() (pynlpl.formats.folia.Part method), 288  
settext() (pynlpl.formats.folia.PhonContent method), 511  
settext() (pynlpl.formats.folia.PosAnnotation method), 446  
settext() (pynlpl.formats.folia.Predicate method), 583  
settext() (pynlpl.formats.folia.Quote method), 301  
settext() (pynlpl.formats.folia.Reference method), 314  
settext() (pynlpl.formats.folia.Row method), 327  
settext() (pynlpl.formats.folia.SemanticRole method), 629  
settext() (pynlpl.formats.folia.SemanticRolesLayer method), 748  
settext() (pynlpl.formats.folia.SenseAnnotation method), 479  
settext() (pynlpl.formats.folia.Sentence method), 342  
settext() (pynlpl.formats.folia.Sentiment method), 594  
settext() (pynlpl.formats.folia.SentimentLayer method), 712  
settext() (pynlpl.formats.folia.Statement method), 606  
settext() (pynlpl.formats.folia.StatementLayer method), 724  
settext() (pynlpl.formats.folia.SubjectivityAnnotation method), 490  
settext() (pynlpl.formats.folia.Suggestion method), 1004  
settext() (pynlpl.formats.folia.SynsetFeature method), 880  
settext() (pynlpl.formats.folia.SyntacticUnit method), 618  
settext() (pynlpl.formats.folia.SyntaxLayer method), 736  
settext() (pynlpl.formats.folia.Table method), 355  
settext() (pynlpl.formats.folia.TableHead method), 381  
settext() (pynlpl.formats.folia.Term method), 368  
settext() (pynlpl.formats.folia.Text method), 394  
settext() (pynlpl.formats.folia.TextContent method), 501  
settext() (pynlpl.formats.folia.TextMarkupCorrection method), 847  
settext() (pynlpl.formats.folia.TextMarkupError method), 857  
settext() (pynlpl.formats.folia.TextMarkupGap method), 815  
settext() (pynlpl.formats.folia.TextMarkupString method), 825



- [settext\(\) \(pynlpl.formats.folia.TextMarkupStyle method\), 836](#)  
[settext\(\) \(pynlpl.formats.folia.TimeSegment method\), 641](#)  
[settext\(\) \(pynlpl.formats.folia.TimingLayer method\), 759](#)  
[settext\(\) \(pynlpl.formats.folia.Whitespace method\), 407](#)  
[settext\(\) \(pynlpl.formats.folia.Word method\), 422](#)  
[settransitions\(\) \(pynlpl.statistics.MarkovChain method\), 1063](#)  
[SimpleLanguageModel \(class in pynlpl.lm.lm\), 1057](#)  
[size\(\) \(pynlpl.datatypes.Trie method\), 7](#)  
[size\(\) \(pynlpl.statistics.MarkovChain method\), 1064](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractAnnotationLayer attribute\), 75](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractElement attribute\), 26](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractSpanAnnotation attribute\), 52](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractStructureElement attribute\), 37](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractTextMarkup attribute\), 86](#)  
[SPEAKABLE \(pynlpl.formats.folia.AbstractTokenAnnotation attribute\), 63](#)  
[SPEAKABLE \(pynlpl.formats.folia.ActorFeature attribute\), 885](#)  
[SPEAKABLE \(pynlpl.formats.folia.Alignment attribute\), 1009](#)  
[SPEAKABLE \(pynlpl.formats.folia.AlignReference attribute\), 1020](#)  
[SPEAKABLE \(pynlpl.formats.folia.Alternative attribute\), 919](#)  
[SPEAKABLE \(pynlpl.formats.folia.AlternativeLayers attribute\), 931](#)  
[SPEAKABLE \(pynlpl.formats.folia.BegindatetimeFeature attribute\), 896](#)  
[SPEAKABLE \(pynlpl.formats.folia.Cell attribute\), 100](#)  
[SPEAKABLE \(pynlpl.formats.folia.Chunk attribute\), 518](#)  
[SPEAKABLE \(pynlpl.formats.folia.ChunkingLayer attribute\), 647](#)  
[SPEAKABLE \(pynlpl.formats.folia.CoreferenceChain attribute\), 530](#)  
[SPEAKABLE \(pynlpl.formats.folia.CoreferenceLayer attribute\), 658](#)  
[SPEAKABLE \(pynlpl.formats.folia.CoreferenceLink attribute\), 764](#)  
[SPEAKABLE \(pynlpl.formats.folia.Correction attribute\), 944](#)  
[SPEAKABLE \(pynlpl.formats.folia.Current attribute\), 953](#)  
[SPEAKABLE \(pynlpl.formats.folia.Definition attribute\), 113](#)  
[SPEAKABLE \(pynlpl.formats.folia.DependenciesLayer attribute\), 670](#)  
[SPEAKABLE \(pynlpl.formats.folia.Dependency attribute\), 541](#)  
[SPEAKABLE \(pynlpl.formats.folia.DependencyDependent attribute\), 776](#)  
[SPEAKABLE \(pynlpl.formats.folia.Description attribute\), 1032](#)  
[SPEAKABLE \(pynlpl.formats.folia.Division attribute\), 126](#)  
[SPEAKABLE \(pynlpl.formats.folia.DomainAnnotation attribute\), 429](#)  
[SPEAKABLE \(pynlpl.formats.folia.EnddatetimeFeature attribute\), 907](#)  
[SPEAKABLE \(pynlpl.formats.folia.EntitiesLayer attribute\), 682](#)  
[SPEAKABLE \(pynlpl.formats.folia.Entity attribute\), 553](#)  
[SPEAKABLE \(pynlpl.formats.folia.Entry attribute\), 139](#)  
[SPEAKABLE \(pynlpl.formats.folia.ErrorDetection attribute\), 965](#)  
[SPEAKABLE \(pynlpl.formats.folia.Event attribute\), 152](#)  
[SPEAKABLE \(pynlpl.formats.folia.Example attribute\), 165](#)  
[SPEAKABLE \(pynlpl.formats.folia.Feature attribute\), 862](#)  
[SPEAKABLE \(pynlpl.formats.folia.Figure attribute\), 178](#)  
[SPEAKABLE \(pynlpl.formats.folia.Gap attribute\), 191](#)  
[SPEAKABLE \(pynlpl.formats.folia.Head attribute\), 202](#)  
[SPEAKABLE \(pynlpl.formats.folia.Headspan attribute\), 788](#)  
[SPEAKABLE \(pynlpl.formats.folia.LangAnnotation attribute\), 451](#)  
[SPEAKABLE \(pynlpl.formats.folia.LemmaAnnotation attribute\), 462](#)  
[SPEAKABLE \(pynlpl.formats.folia.Linebreak attribute\), 215](#)  
[SPEAKABLE \(pynlpl.formats.folia.List attribute\), 228](#)  
[SPEAKABLE \(pynlpl.formats.folia.ListItem attribute\), 241](#)  
[SPEAKABLE \(pynlpl.formats.folia.Metric attribute\), 1043](#)  
[SPEAKABLE \(pynlpl.formats.folia.New attribute\), 976](#)  
[SPEAKABLE \(pynlpl.formats.folia.Note attribute\), 254](#)  
[SPEAKABLE \(pynlpl.formats.folia.Observation attribute\), 564](#)  
[SPEAKABLE \(pynlpl.formats.folia.ObservationLayer attribute\), 694](#)  
[SPEAKABLE \(pynlpl.formats.folia.Original attribute\), 987](#)  
[SPEAKABLE \(pynlpl.formats.folia.Paragraph attribute\), 267](#)  
[SPEAKABLE \(pynlpl.formats.folia.Part attribute\), 280](#)  
[SPEAKABLE \(pynlpl.formats.folia.PhonContent attribute\), 506](#)  
[SPEAKABLE \(pynlpl.formats.folia.PosAnnotation attribute\), 440](#)

SPEAKABLE (pynlpl.formats.folia.Predicate attribute), 576

SPEAKABLE (pynlpl.formats.folia.Quote attribute), 293

SPEAKABLE (pynlpl.formats.folia.Reference attribute), 306

SPEAKABLE (pynlpl.formats.folia.Row attribute), 319

SPEAKABLE (pynlpl.formats.folia.SemanticRole attribute), 623

SPEAKABLE (pynlpl.formats.folia.SemanticRolesLayer attribute), 741

SPEAKABLE (pynlpl.formats.folia.SenseAnnotation attribute), 473

SPEAKABLE (pynlpl.formats.folia.Sentence attribute), 333

SPEAKABLE (pynlpl.formats.folia.Sentiment attribute), 588

SPEAKABLE (pynlpl.formats.folia.SentimentLayer attribute), 705

SPEAKABLE (pynlpl.formats.folia.Statement attribute), 599

SPEAKABLE (pynlpl.formats.folia.StatementLayer attribute), 717

SPEAKABLE (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484

SPEAKABLE (pynlpl.formats.folia.Suggestion attribute), 998

SPEAKABLE (pynlpl.formats.folia.SynsetFeature attribute), 874

SPEAKABLE (pynlpl.formats.folia.SyntacticUnit attribute), 611

SPEAKABLE (pynlpl.formats.folia.SyntaxLayer attribute), 729

SPEAKABLE (pynlpl.formats.folia.Table attribute), 347

SPEAKABLE (pynlpl.formats.folia.TableHead attribute), 373

SPEAKABLE (pynlpl.formats.folia.Term attribute), 360

SPEAKABLE (pynlpl.formats.folia.Text attribute), 386

SPEAKABLE (pynlpl.formats.folia.TextContent attribute), 495

SPEAKABLE (pynlpl.formats.folia.TextMarkupCorrection attribute), 840

SPEAKABLE (pynlpl.formats.folia.TextMarkupError attribute), 851

SPEAKABLE (pynlpl.formats.folia.TextMarkupGap attribute), 808

SPEAKABLE (pynlpl.formats.folia.TextMarkupString attribute), 819

SPEAKABLE (pynlpl.formats.folia.TextMarkupStyle attribute), 830

SPEAKABLE (pynlpl.formats.folia.TimeSegment attribute), 634

SPEAKABLE (pynlpl.formats.folia.TimingLayer attribute), 752

SPEAKABLE (pynlpl.formats.folia.Whitespace attribute), 399

SPEAKABLE (pynlpl.formats.folia.Word attribute), 413

specificity() (pynlpl.evaluation.ClassEvaluation method), 10

speech\_speaker() (pynlpl.formats.folia.AbstractAnnotationLayer method), 82

speech\_speaker() (pynlpl.formats.folia.AbstractElement method), 32

speech\_speaker() (pynlpl.formats.folia.AbstractSpanAnnotation method), 59

speech\_speaker() (pynlpl.formats.folia.AbstractStructureElement method), 45

speech\_speaker() (pynlpl.formats.folia.AbstractTextMarkup method), 92

speech\_speaker() (pynlpl.formats.folia.AbstractTokenAnnotation method), 70

speech\_speaker() (pynlpl.formats.folia.ActorFeature method), 891

speech\_speaker() (pynlpl.formats.folia.Alignment method), 1015

speech\_speaker() (pynlpl.formats.folia.AlignReference method), 1027

speech\_speaker() (pynlpl.formats.folia.Alternative method), 927

speech\_speaker() (pynlpl.formats.folia.AlternativeLayers method), 938

speech\_speaker() (pynlpl.formats.folia.BegindatetimeFeature method), 902

speech\_speaker() (pynlpl.formats.folia.Cell method), 108

speech\_speaker() (pynlpl.formats.folia.Chunk method), 525

speech\_speaker() (pynlpl.formats.folia.ChunkingLayer method), 654

speech\_speaker() (pynlpl.formats.folia.CoreferenceChain method), 536

speech\_speaker() (pynlpl.formats.folia.CoreferenceLayer method), 665

speech\_speaker() (pynlpl.formats.folia.CoreferenceLink method), 771

speech\_speaker() (pynlpl.formats.folia.Correction method), 950

speech\_speaker() (pynlpl.formats.folia.Current method), 960

speech\_speaker() (pynlpl.formats.folia.Definition method), 121

speech\_speaker() (pynlpl.formats.folia.DependenciesLayer method), 677

speech\_speaker() (pynlpl.formats.folia.Dependency method), 548

speech\_speaker() (pynlpl.formats.folia.DependencyDependent method), 783

speech\_speaker() (pynlpl.formats.folia.Description method), 1038

speech\_speaker() (pynlpl.formats.folia.Division method),

- 134
- `speech_speaker()` (`pynlpl.formats.folia.DomainAnnotation` method), 435
- `speech_speaker()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 913
- `speech_speaker()` (`pynlpl.formats.folia.EntitiesLayer` method), 689
- `speech_speaker()` (`pynlpl.formats.folia.Entity` method), 560
- `speech_speaker()` (`pynlpl.formats.folia.Entry` method), 147
- `speech_speaker()` (`pynlpl.formats.folia.ErrorDetection` method), 971
- `speech_speaker()` (`pynlpl.formats.folia.Event` method), 160
- `speech_speaker()` (`pynlpl.formats.folia.Example` method), 173
- `speech_speaker()` (`pynlpl.formats.folia.Feature` method), 869
- `speech_speaker()` (`pynlpl.formats.folia.Figure` method), 186
- `speech_speaker()` (`pynlpl.formats.folia.Gap` method), 197
- `speech_speaker()` (`pynlpl.formats.folia.Head` method), 210
- `speech_speaker()` (`pynlpl.formats.folia.Headspan` method), 795
- `speech_speaker()` (`pynlpl.formats.folia.LangAnnotation` method), 457
- `speech_speaker()` (`pynlpl.formats.folia.LemmaAnnotation` method), 468
- `speech_speaker()` (`pynlpl.formats.folia.Linebreak` method), 223
- `speech_speaker()` (`pynlpl.formats.folia.List` method), 236
- `speech_speaker()` (`pynlpl.formats.folia.ListItem` method), 249
- `speech_speaker()` (`pynlpl.formats.folia.Metric` method), 1049
- `speech_speaker()` (`pynlpl.formats.folia.New` method), 982
- `speech_speaker()` (`pynlpl.formats.folia.Note` method), 262
- `speech_speaker()` (`pynlpl.formats.folia.Observation` method), 571
- `speech_speaker()` (`pynlpl.formats.folia.ObservationLayer` method), 701
- `speech_speaker()` (`pynlpl.formats.folia.Original` method), 993
- `speech_speaker()` (`pynlpl.formats.folia.Paragraph` method), 275
- `speech_speaker()` (`pynlpl.formats.folia.Part` method), 288
- `speech_speaker()` (`pynlpl.formats.folia.PhonContent` method), 512
- `speech_speaker()` (`pynlpl.formats.folia.PosAnnotation` method), 446
- `speech_speaker()` (`pynlpl.formats.folia.Predicate` method), 583
- `speech_speaker()` (`pynlpl.formats.folia.Quote` method), 301
- `speech_speaker()` (`pynlpl.formats.folia.Reference` method), 314
- `speech_speaker()` (`pynlpl.formats.folia.Row` method), 327
- `speech_speaker()` (`pynlpl.formats.folia.SemanticRole` method), 630
- `speech_speaker()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 748
- `speech_speaker()` (`pynlpl.formats.folia.SenseAnnotation` method), 479
- `speech_speaker()` (`pynlpl.formats.folia.Sentence` method), 342
- `speech_speaker()` (`pynlpl.formats.folia.Sentiment` method), 595
- `speech_speaker()` (`pynlpl.formats.folia.SentimentLayer` method), 712
- `speech_speaker()` (`pynlpl.formats.folia.Statement` method), 606
- `speech_speaker()` (`pynlpl.formats.folia.StatementLayer` method), 724
- `speech_speaker()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 490
- `speech_speaker()` (`pynlpl.formats.folia.Suggestion` method), 1004
- `speech_speaker()` (`pynlpl.formats.folia.SynsetFeature` method), 880
- `speech_speaker()` (`pynlpl.formats.folia.SyntacticUnit` method), 618
- `speech_speaker()` (`pynlpl.formats.folia.SyntaxLayer` method), 736
- `speech_speaker()` (`pynlpl.formats.folia.Table` method), 355
- `speech_speaker()` (`pynlpl.formats.folia.TableHead` method), 381
- `speech_speaker()` (`pynlpl.formats.folia.Term` method), 368
- `speech_speaker()` (`pynlpl.formats.folia.Text` method), 394
- `speech_speaker()` (`pynlpl.formats.folia.TextContent` method), 501
- `speech_speaker()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 847
- `speech_speaker()` (`pynlpl.formats.folia.TextMarkupError` method), 857
- `speech_speaker()` (`pynlpl.formats.folia.TextMarkupGap` method), 815
- `speech_speaker()` (`pynlpl.formats.folia.TextMarkupString` method), 825
- `speech_speaker()` (`pynlpl.formats.folia.TextMarkupStyle` method), 836
- `speech_speaker()` (`pynlpl.formats.folia.TimeSegment` method), 641
- `speech_speaker()` (`pynlpl.formats.folia.TimingLayer` method), 583

method), 759  
speech\_speaker() (pynlpl.formats.folia.Whitespace method), 407  
speech\_speaker() (pynlpl.formats.folia.Word method), 422  
speech\_src() (pynlpl.formats.folia.AbstractAnnotationLayers method), 82  
speech\_src() (pynlpl.formats.folia.AbstractElement method), 32  
speech\_src() (pynlpl.formats.folia.AbstractSpanAnnotation method), 59  
speech\_src() (pynlpl.formats.folia.AbstractStructureElements method), 45  
speech\_src() (pynlpl.formats.folia.AbstractTextMarkup method), 93  
speech\_src() (pynlpl.formats.folia.AbstractTokenAnnotations method), 70  
speech\_src() (pynlpl.formats.folia.ActorFeature method), 891  
speech\_src() (pynlpl.formats.folia.Alignment method), 1016  
speech\_src() (pynlpl.formats.folia.AlignReference method), 1027  
speech\_src() (pynlpl.formats.folia.Alternative method), 927  
speech\_src() (pynlpl.formats.folia.AlternativeLayers method), 938  
speech\_src() (pynlpl.formats.folia.BegindatetimeFeature method), 902  
speech\_src() (pynlpl.formats.folia.Cell method), 108  
speech\_src() (pynlpl.formats.folia.Chunk method), 525  
speech\_src() (pynlpl.formats.folia.ChunkingLayer method), 654  
speech\_src() (pynlpl.formats.folia.CoreferenceChain method), 536  
speech\_src() (pynlpl.formats.folia.CoreferenceLayer method), 666  
speech\_src() (pynlpl.formats.folia.CoreferenceLink method), 771  
speech\_src() (pynlpl.formats.folia.Correction method), 950  
speech\_src() (pynlpl.formats.folia.Current method), 960  
speech\_src() (pynlpl.formats.folia.Definition method), 121  
speech\_src() (pynlpl.formats.folia.DependenciesLayer method), 677  
speech\_src() (pynlpl.formats.folia.Dependency method), 548  
speech\_src() (pynlpl.formats.folia.DependencyDependent method), 783  
speech\_src() (pynlpl.formats.folia.Description method), 1038  
speech\_src() (pynlpl.formats.folia.Division method), 134  
speech\_src() (pynlpl.formats.folia.DomainAnnotation method), 435  
speech\_src() (pynlpl.formats.folia.EnddatetimeFeature method), 913  
speech\_src() (pynlpl.formats.folia.EntitiesLayer method), 689  
speech\_src() (pynlpl.formats.folia.Entity method), 560  
speech\_src() (pynlpl.formats.folia.Entry method), 147  
speech\_src() (pynlpl.formats.folia.ErrorDetection method), 971  
speech\_src() (pynlpl.formats.folia.Event method), 160  
speech\_src() (pynlpl.formats.folia.Example method), 173  
speech\_src() (pynlpl.formats.folia.Feature method), 869  
speech\_src() (pynlpl.formats.folia.Figure method), 186  
speech\_src() (pynlpl.formats.folia.Gap method), 198  
speech\_src() (pynlpl.formats.folia.Head method), 210  
speech\_src() (pynlpl.formats.folia.Headspan method), 795  
speech\_src() (pynlpl.formats.folia.LangAnnotation method), 457  
speech\_src() (pynlpl.formats.folia.LemmaAnnotation method), 468  
speech\_src() (pynlpl.formats.folia.Linebreak method), 223  
speech\_src() (pynlpl.formats.folia.List method), 236  
speech\_src() (pynlpl.formats.folia.ListItem method), 249  
speech\_src() (pynlpl.formats.folia.Metric method), 1049  
speech\_src() (pynlpl.formats.folia.New method), 982  
speech\_src() (pynlpl.formats.folia.Note method), 262  
speech\_src() (pynlpl.formats.folia.Observation method), 571  
speech\_src() (pynlpl.formats.folia.ObservationLayer method), 701  
speech\_src() (pynlpl.formats.folia.Original method), 993  
speech\_src() (pynlpl.formats.folia.Paragraph method), 275  
speech\_src() (pynlpl.formats.folia.Part method), 288  
speech\_src() (pynlpl.formats.folia.PhonContent method), 512  
speech\_src() (pynlpl.formats.folia.PosAnnotation method), 446  
speech\_src() (pynlpl.formats.folia.Predicate method), 583  
speech\_src() (pynlpl.formats.folia.Quote method), 301  
speech\_src() (pynlpl.formats.folia.Reference method), 314  
speech\_src() (pynlpl.formats.folia.Row method), 327  
speech\_src() (pynlpl.formats.folia.SemanticRole method), 630  
speech\_src() (pynlpl.formats.folia.SemanticRolesLayer method), 748  
speech\_src() (pynlpl.formats.folia.SenseAnnotation method), 479  
speech\_src() (pynlpl.formats.folia.Sentence method), 342  
speech\_src() (pynlpl.formats.folia.Sentiment method), 595

[speech\\_src\(\)](#) (pynlpl.formats.folia.SentimentLayer method), [713](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Statement method), [606](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.StatementLayer method), [724](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.SubjectivityAnnotation method), [490](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Suggestion method), [1004](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.SynsetFeature method), [880](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.SyntacticUnit method), [618](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.SyntaxLayer method), [736](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Table method), [355](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TableHead method), [381](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Term method), [368](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Text method), [394](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextContent method), [502](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextMarkupCorrection method), [847](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextMarkupError method), [857](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextMarkupGap method), [815](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextMarkupString method), [825](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TextMarkupStyle method), [836](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TimeSegment method), [641](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.TimingLayer method), [760](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Whitespace method), [407](#)  
[speech\\_src\(\)](#) (pynlpl.formats.folia.Word method), [422](#)  
[split\(\)](#) (pynlpl.formats.folia.Word method), [422](#)  
[split\\_sentences\(\)](#) (in module pynlpl.textprocessors), [1066](#)  
[splitword\(\)](#) (pynlpl.formats.folia.Sentence method), [342](#)  
[SRILM](#) (class in pynlpl.lm.srilm), [1057](#)  
[SRILMException](#), [1058](#)  
[start\(\)](#) (pynlpl.evaluation.AbstractExperiment method), [9](#)  
[start\(\)](#) (pynlpl.evaluation.ExperimentPool method), [10](#)  
[startcommand\(\)](#) (pynlpl.evaluation.AbstractExperiment method), [9](#)  
[Statement](#) (class in pynlpl.formats.folia), [597](#)  
[StatementLayer](#) (class in pynlpl.formats.folia), [714](#)  
[stddev\(\)](#) (in module pynlpl.statistics), [1064](#)  
[StochasticBeamSearch](#) (class in pynlpl.search), [1060](#)  
[stochasticprune\(\)](#) (pynlpl.datatypes.PriorityQueue method), [6](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractAnnotationLayer method), [82](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractElement method), [32](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractSpanAnnotation method), [59](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractStructureElement method), [46](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractTextMarkup method), [93](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AbstractTokenAnnotation method), [70](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.ActorFeature method), [892](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Alignment method), [1016](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AlignReference method), [1027](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Alternative method), [927](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.AlternativeLayers method), [938](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.BegindatetimeFeature method), [903](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Cell method), [109](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Chunk method), [525](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.ChunkingLayer method), [654](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.CoreferenceChain method), [537](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.CoreferenceLayer method), [666](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.CoreferenceLink method), [771](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Correction method), [950](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Current method), [960](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Definition method), [122](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.DependenciesLayer method), [677](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Dependency method), [548](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.DependencyDependent method), [783](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Description method), [1038](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Division method), [135](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.DomainAnnotation method), [435](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.EnddatetimeFeature method), [914](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.EntitiesLayer method), [689](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Entity method), [560](#)  
[stricttext\(\)](#) (pynlpl.formats.folia.Entry method), [148](#)



`stricttext()` (`pynlpl.formats.folia.ErrorDetection` method), 971

`stricttext()` (`pynlpl.formats.folia.Event` method), 161

`stricttext()` (`pynlpl.formats.folia.Example` method), 174

`stricttext()` (`pynlpl.formats.folia.Feature` method), 869

`stricttext()` (`pynlpl.formats.folia.Figure` method), 187

`stricttext()` (`pynlpl.formats.folia.Gap` method), 198

`stricttext()` (`pynlpl.formats.folia.Head` method), 211

`stricttext()` (`pynlpl.formats.folia.Headspan` method), 795

`stricttext()` (`pynlpl.formats.folia.LangAnnotation` method), 457

`stricttext()` (`pynlpl.formats.folia.LemmaAnnotation` method), 468

`stricttext()` (`pynlpl.formats.folia.Linebreak` method), 223

`stricttext()` (`pynlpl.formats.folia.List` method), 237

`stricttext()` (`pynlpl.formats.folia.ListItem` method), 250

`stricttext()` (`pynlpl.formats.folia.Metric` method), 1049

`stricttext()` (`pynlpl.formats.folia.New` method), 982

`stricttext()` (`pynlpl.formats.folia.Note` method), 263

`stricttext()` (`pynlpl.formats.folia.Observation` method), 572

`stricttext()` (`pynlpl.formats.folia.ObservationLayer` method), 701

`stricttext()` (`pynlpl.formats.folia.Original` method), 993

`stricttext()` (`pynlpl.formats.folia.Paragraph` method), 276

`stricttext()` (`pynlpl.formats.folia.Part` method), 289

`stricttext()` (`pynlpl.formats.folia.PhonContent` method), 512

`stricttext()` (`pynlpl.formats.folia.PosAnnotation` method), 446

`stricttext()` (`pynlpl.formats.folia.Predicate` method), 583

`stricttext()` (`pynlpl.formats.folia.Quote` method), 302

`stricttext()` (`pynlpl.formats.folia.Reference` method), 315

`stricttext()` (`pynlpl.formats.folia.Row` method), 328

`stricttext()` (`pynlpl.formats.folia.SemanticRole` method), 630

`stricttext()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 748

`stricttext()` (`pynlpl.formats.folia.SenseAnnotation` method), 479

`stricttext()` (`pynlpl.formats.folia.Sentence` method), 342

`stricttext()` (`pynlpl.formats.folia.Sentiment` method), 595

`stricttext()` (`pynlpl.formats.folia.SentimentLayer` method), 713

`stricttext()` (`pynlpl.formats.folia.Statement` method), 607

`stricttext()` (`pynlpl.formats.folia.StatementLayer` method), 724

`stricttext()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 490

`stricttext()` (`pynlpl.formats.folia.Suggestion` method), 1004

`stricttext()` (`pynlpl.formats.folia.SynsetFeature` method), 880

`stricttext()` (`pynlpl.formats.folia.SyntacticUnit` method), 618

`stricttext()` (`pynlpl.formats.folia.SyntaxLayer` method), 736

`stricttext()` (`pynlpl.formats.folia.Table` method), 355

`stricttext()` (`pynlpl.formats.folia.TableHead` method), 381

`stricttext()` (`pynlpl.formats.folia.Term` method), 368

`stricttext()` (`pynlpl.formats.folia.Text` method), 394

`stricttext()` (`pynlpl.formats.folia.TextContent` method), 502

`stricttext()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 847

`stricttext()` (`pynlpl.formats.folia.TextMarkupError` method), 858

`stricttext()` (`pynlpl.formats.folia.TextMarkupGap` method), 815

`stricttext()` (`pynlpl.formats.folia.TextMarkupString` method), 826

`stricttext()` (`pynlpl.formats.folia.TextMarkupStyle` method), 836

`stricttext()` (`pynlpl.formats.folia.TimeSegment` method), 642

`stricttext()` (`pynlpl.formats.folia.TimingLayer` method), 760

`stricttext()` (`pynlpl.formats.folia.Whitespace` method), 407

`stricttext()` (`pynlpl.formats.folia.Word` method), 422

`strip_accents()` (in module `pynlpl.textprocessors`), 1067

`SubjectivityAnnotation` (class in `pynlpl.formats.folia`), 481

`SUBSET` (`pynlpl.formats.folia.AbstractAnnotationLayer` attribute), 75

`SUBSET` (`pynlpl.formats.folia.AbstractElement` attribute), 26

`SUBSET` (`pynlpl.formats.folia.AbstractSpanAnnotation` attribute), 52

`SUBSET` (`pynlpl.formats.folia.AbstractStructureElement` attribute), 37

`SUBSET` (`pynlpl.formats.folia.AbstractTextMarkup` attribute), 86

`SUBSET` (`pynlpl.formats.folia.AbstractTokenAnnotation` attribute), 63

`SUBSET` (`pynlpl.formats.folia.ActorFeature` attribute), 885

`SUBSET` (`pynlpl.formats.folia.Alignment` attribute), 1009

`SUBSET` (`pynlpl.formats.folia.AlignReference` attribute), 1020

`SUBSET` (`pynlpl.formats.folia.Alternative` attribute), 919

`SUBSET` (`pynlpl.formats.folia.AlternativeLayers` attribute), 931

`SUBSET` (`pynlpl.formats.folia.BegindatetimeFeature` attribute), 896

`SUBSET` (`pynlpl.formats.folia.Cell` attribute), 101

`SUBSET` (`pynlpl.formats.folia.Chunk` attribute), 518

- SUBSET (pynlpl.formats.folia.ChunkingLayer attribute), 647
- SUBSET (pynlpl.formats.folia.CoreferenceChain attribute), 530
- SUBSET (pynlpl.formats.folia.CoreferenceLayer attribute), 658
- SUBSET (pynlpl.formats.folia.CoreferenceLink attribute), 764
- SUBSET (pynlpl.formats.folia.Correction attribute), 944
- SUBSET (pynlpl.formats.folia.Current attribute), 954
- SUBSET (pynlpl.formats.folia.Definition attribute), 113
- SUBSET (pynlpl.formats.folia.DependenciesLayer attribute), 670
- SUBSET (pynlpl.formats.folia.Dependency attribute), 541
- SUBSET (pynlpl.formats.folia.DependencyDependent attribute), 776
- SUBSET (pynlpl.formats.folia.Description attribute), 1032
- SUBSET (pynlpl.formats.folia.Division attribute), 126
- SUBSET (pynlpl.formats.folia.DomainAnnotation attribute), 429
- SUBSET (pynlpl.formats.folia.EnddatetimeFeature attribute), 907
- SUBSET (pynlpl.formats.folia.EntitiesLayer attribute), 682
- SUBSET (pynlpl.formats.folia.Entity attribute), 553
- SUBSET (pynlpl.formats.folia.Entry attribute), 139
- SUBSET (pynlpl.formats.folia.ErrorDetection attribute), 965
- SUBSET (pynlpl.formats.folia.Event attribute), 152
- SUBSET (pynlpl.formats.folia.Example attribute), 165
- SUBSET (pynlpl.formats.folia.Feature attribute), 862
- SUBSET (pynlpl.formats.folia.Figure attribute), 178
- SUBSET (pynlpl.formats.folia.Gap attribute), 191
- SUBSET (pynlpl.formats.folia.Head attribute), 202
- SUBSET (pynlpl.formats.folia.Headspan attribute), 788
- SUBSET (pynlpl.formats.folia.LangAnnotation attribute), 451
- SUBSET (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- SUBSET (pynlpl.formats.folia.Linebreak attribute), 215
- SUBSET (pynlpl.formats.folia.List attribute), 228
- SUBSET (pynlpl.formats.folia.ListItem attribute), 241
- SUBSET (pynlpl.formats.folia.Metric attribute), 1043
- SUBSET (pynlpl.formats.folia.New attribute), 976
- SUBSET (pynlpl.formats.folia.Note attribute), 254
- SUBSET (pynlpl.formats.folia.Observation attribute), 565
- SUBSET (pynlpl.formats.folia.ObservationLayer attribute), 694
- SUBSET (pynlpl.formats.folia.Original attribute), 987
- SUBSET (pynlpl.formats.folia.Paragraph attribute), 267
- SUBSET (pynlpl.formats.folia.Part attribute), 280
- SUBSET (pynlpl.formats.folia.PhonContent attribute), 506
- SUBSET (pynlpl.formats.folia.PosAnnotation attribute), 440
- SUBSET (pynlpl.formats.folia.Predicate attribute), 576
- SUBSET (pynlpl.formats.folia.Quote attribute), 293
- SUBSET (pynlpl.formats.folia.Reference attribute), 306
- SUBSET (pynlpl.formats.folia.Row attribute), 319
- SUBSET (pynlpl.formats.folia.SemanticRole attribute), 623
- SUBSET (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- SUBSET (pynlpl.formats.folia.SenseAnnotation attribute), 473
- SUBSET (pynlpl.formats.folia.Sentence attribute), 333
- SUBSET (pynlpl.formats.folia.Sentiment attribute), 588
- SUBSET (pynlpl.formats.folia.SentimentLayer attribute), 705
- SUBSET (pynlpl.formats.folia.Statement attribute), 599
- SUBSET (pynlpl.formats.folia.StatementLayer attribute), 717
- SUBSET (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- SUBSET (pynlpl.formats.folia.Suggestion attribute), 998
- SUBSET (pynlpl.formats.folia.SynsetFeature attribute), 874
- SUBSET (pynlpl.formats.folia.SyntacticUnit attribute), 611
- SUBSET (pynlpl.formats.folia.SyntaxLayer attribute), 729
- SUBSET (pynlpl.formats.folia.Table attribute), 347
- SUBSET (pynlpl.formats.folia.TableHead attribute), 373
- SUBSET (pynlpl.formats.folia.Term attribute), 360
- SUBSET (pynlpl.formats.folia.Text attribute), 386
- SUBSET (pynlpl.formats.folia.TextContent attribute), 495
- SUBSET (pynlpl.formats.folia.TextMarkupCorrection attribute), 840
- SUBSET (pynlpl.formats.folia.TextMarkupError attribute), 851
- SUBSET (pynlpl.formats.folia.TextMarkupGap attribute), 808
- SUBSET (pynlpl.formats.folia.TextMarkupString attribute), 819
- SUBSET (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- SUBSET (pynlpl.formats.folia.TimeSegment attribute), 634
- SUBSET (pynlpl.formats.folia.TimingLayer attribute), 752
- SUBSET (pynlpl.formats.folia.Whitespace attribute), 399
- SUBSET (pynlpl.formats.folia.Word attribute), 413
- Suggestion (class in pynlpl.formats.folia), 995
- suggestions() (pynlpl.formats.folia.Correction method),

950

sum() (pynlpl.statistics.FrequencyList method), 1063

swap() (in module pynlpl.textprocessors), 1067

SynsetFeature (class in pynlpl.formats.folia), 871

SyntacticUnit (class in pynlpl.formats.folia), 608

SyntaxLayer (class in pynlpl.formats.folia), 726

## T

Table (class in pynlpl.formats.folia), 344

TableHead (class in pynlpl.formats.folia), 370

Taggerdata (class in pynlpl.formats.taggerdata), 1055

targetword() (pynlpl.formats.giza.MultiWordAlignment method), 1054

targetword() (pynlpl.formats.giza.WordAlignment method), 1054

targetwords() (pynlpl.formats.giza.MultiWordAlignment method), 1054

Term (class in pynlpl.formats.folia), 357

test() (pynlpl.evaluation.WSPParamSearch method), 10

test() (pynlpl.search.AbstractSearchState method), 1060

Text (class in pynlpl.formats.folia), 383

text() (pynlpl.formats.folia.AbstractAnnotationLayer method), 82

text() (pynlpl.formats.folia.AbstractElement method), 32

text() (pynlpl.formats.folia.AbstractSpanAnnotation method), 59

text() (pynlpl.formats.folia.AbstractStructureElement method), 46

text() (pynlpl.formats.folia.AbstractTextMarkup method), 93

text() (pynlpl.formats.folia.AbstractTokenAnnotation method), 70

text() (pynlpl.formats.folia.ActorFeature method), 892

text() (pynlpl.formats.folia.Alignment method), 1016

text() (pynlpl.formats.folia.AlignReference method), 1027

text() (pynlpl.formats.folia.Alternative method), 927

text() (pynlpl.formats.folia.AlternativeLayers method), 938

text() (pynlpl.formats.folia.BegindatetimeFeature method), 903

text() (pynlpl.formats.folia.Cell method), 109

text() (pynlpl.formats.folia.Chunk method), 525

text() (pynlpl.formats.folia.ChunkingLayer method), 654

text() (pynlpl.formats.folia.CoreferenceChain method), 537

text() (pynlpl.formats.folia.CoreferenceLayer method), 666

text() (pynlpl.formats.folia.CoreferenceLink method), 772

text() (pynlpl.formats.folia.Correction method), 950

text() (pynlpl.formats.folia.Current method), 960

text() (pynlpl.formats.folia.Definition method), 122

text() (pynlpl.formats.folia.DependenciesLayer method), 677

text() (pynlpl.formats.folia.Dependency method), 548

text() (pynlpl.formats.folia.DependencyDependent method), 783

text() (pynlpl.formats.folia.Description method), 1038

text() (pynlpl.formats.folia.Division method), 135

text() (pynlpl.formats.folia.Document method), 20

text() (pynlpl.formats.folia.DomainAnnotation method), 435

text() (pynlpl.formats.folia.EnddatetimeFeature method), 914

text() (pynlpl.formats.folia.EntitiesLayer method), 689

text() (pynlpl.formats.folia.Entity method), 560

text() (pynlpl.formats.folia.Entry method), 148

text() (pynlpl.formats.folia.ErrorDetection method), 971

text() (pynlpl.formats.folia.Event method), 161

text() (pynlpl.formats.folia.Example method), 174

text() (pynlpl.formats.folia.Feature method), 869

text() (pynlpl.formats.folia.Figure method), 187

text() (pynlpl.formats.folia.Gap method), 198

text() (pynlpl.formats.folia.Head method), 211

text() (pynlpl.formats.folia.Headspan method), 795

text() (pynlpl.formats.folia.LangAnnotation method), 457

text() (pynlpl.formats.folia.LemmaAnnotation method), 468

text() (pynlpl.formats.folia.Linebreak method), 223

text() (pynlpl.formats.folia.List method), 237

text() (pynlpl.formats.folia.ListItem method), 250

text() (pynlpl.formats.folia.Metric method), 1049

text() (pynlpl.formats.folia.New method), 982

text() (pynlpl.formats.folia.Note method), 263

text() (pynlpl.formats.folia.Observation method), 572

text() (pynlpl.formats.folia.ObservationLayer method), 701

text() (pynlpl.formats.folia.Original method), 993

text() (pynlpl.formats.folia.Paragraph method), 276

text() (pynlpl.formats.folia.Part method), 289

text() (pynlpl.formats.folia.PhonContent method), 512

text() (pynlpl.formats.folia.PosAnnotation method), 446

text() (pynlpl.formats.folia.Predicate method), 583

text() (pynlpl.formats.folia.Quote method), 302

text() (pynlpl.formats.folia.Reference method), 315

text() (pynlpl.formats.folia.Row method), 328

text() (pynlpl.formats.folia.SemanticRole method), 630

text() (pynlpl.formats.folia.SemanticRolesLayer method), 748

text() (pynlpl.formats.folia.SenseAnnotation method), 479

text() (pynlpl.formats.folia.Sentence method), 342

text() (pynlpl.formats.folia.Sentiment method), 595

text() (pynlpl.formats.folia.SentimentLayer method), 713

text() (pynlpl.formats.folia.Statement method), 607

text() (pynlpl.formats.folia.StatementLayer method), 724



- text() (pynlpl.formats.folia.SubjectivityAnnotation method), 490
- text() (pynlpl.formats.folia.Suggestion method), 1004
- text() (pynlpl.formats.folia.SynsetFeature method), 880
- text() (pynlpl.formats.folia.SyntacticUnit method), 618
- text() (pynlpl.formats.folia.SyntaxLayer method), 736
- text() (pynlpl.formats.folia.Table method), 355
- text() (pynlpl.formats.folia.TableHead method), 381
- text() (pynlpl.formats.folia.Term method), 368
- text() (pynlpl.formats.folia.Text method), 394
- text() (pynlpl.formats.folia.TextContent method), 502
- text() (pynlpl.formats.folia.TextMarkupCorrection method), 847
- text() (pynlpl.formats.folia.TextMarkupError method), 858
- text() (pynlpl.formats.folia.TextMarkupGap method), 815
- text() (pynlpl.formats.folia.TextMarkupString method), 826
- text() (pynlpl.formats.folia.TextMarkupStyle method), 836
- text() (pynlpl.formats.folia.TimeSegment method), 642
- text() (pynlpl.formats.folia.TimingLayer method), 760
- text() (pynlpl.formats.folia.Whitespace method), 407
- text() (pynlpl.formats.folia.Word method), 422
- TEXTCONTAINER (pynlpl.formats.folia.AbstractAnnotation attribute), 75
- TEXTCONTAINER (pynlpl.formats.folia.AbstractElement attribute), 26
- TEXTCONTAINER (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52
- TEXTCONTAINER (pynlpl.formats.folia.AbstractStructure attribute), 37
- TEXTCONTAINER (pynlpl.formats.folia.AbstractTextMarkup attribute), 86
- TEXTCONTAINER (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63
- TEXTCONTAINER (pynlpl.formats.folia.ActorFeature attribute), 885
- TEXTCONTAINER (pynlpl.formats.folia.Alignment attribute), 1009
- TEXTCONTAINER (pynlpl.formats.folia.AlignReference attribute), 1020
- TEXTCONTAINER (pynlpl.formats.folia.Alternative attribute), 919
- TEXTCONTAINER (pynlpl.formats.folia.AlternativeLayers attribute), 931
- TEXTCONTAINER (pynlpl.formats.folia.BegindatetimeFeature attribute), 896
- TEXTCONTAINER (pynlpl.formats.folia.Cell attribute), 101
- TEXTCONTAINER (pynlpl.formats.folia.Chunk attribute), 518
- TEXTCONTAINER (pynlpl.formats.folia.ChunkingLayer attribute), 647
- TEXTCONTAINER (pynlpl.formats.folia.CoreferenceChain attribute), 530
- TEXTCONTAINER (pynlpl.formats.folia.CoreferenceLayer attribute), 658
- TEXTCONTAINER (pynlpl.formats.folia.CoreferenceLink attribute), 764
- TEXTCONTAINER (pynlpl.formats.folia.Correction attribute), 944
- TEXTCONTAINER (pynlpl.formats.folia.Current attribute), 954
- TEXTCONTAINER (pynlpl.formats.folia.Definition attribute), 113
- TEXTCONTAINER (pynlpl.formats.folia.DependenciesLayer attribute), 670
- TEXTCONTAINER (pynlpl.formats.folia.Dependency attribute), 541
- TEXTCONTAINER (pynlpl.formats.folia.DependencyDependent attribute), 776
- TEXTCONTAINER (pynlpl.formats.folia.Description attribute), 1032
- TEXTCONTAINER (pynlpl.formats.folia.Division attribute), 126
- TEXTCONTAINER (pynlpl.formats.folia.DomainAnnotation attribute), 429
- TEXTCONTAINER (pynlpl.formats.folia.EnddatetimeFeature attribute), 907
- TEXTCONTAINER (pynlpl.formats.folia.EntitiesLayer attribute), 682
- TEXTCONTAINER (pynlpl.formats.folia.Entity attribute), 553
- TEXTCONTAINER (pynlpl.formats.folia.Entry attribute), 139
- TEXTCONTAINER (pynlpl.formats.folia.ErrorDetection attribute), 965
- TEXTCONTAINER (pynlpl.formats.folia.Event attribute), 152
- TEXTCONTAINER (pynlpl.formats.folia.Example attribute), 165
- TEXTCONTAINER (pynlpl.formats.folia.Feature attribute), 862
- TEXTCONTAINER (pynlpl.formats.folia.Figure attribute), 178
- TEXTCONTAINER (pynlpl.formats.folia.Gap attribute), 191
- TEXTCONTAINER (pynlpl.formats.folia.Head attribute), 203
- TEXTCONTAINER (pynlpl.formats.folia.Headspan attribute), 788
- TEXTCONTAINER (pynlpl.formats.folia.LangAnnotation attribute), 451
- TEXTCONTAINER (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- TEXTCONTAINER (pynlpl.formats.folia.Linebreak attribute), 215

TEXTCONTAINER (pynlpl.formats.folia.List attribute), 228

TEXTCONTAINER (pynlpl.formats.folia.ListItem attribute), 241

TEXTCONTAINER (pynlpl.formats.folia.Metric attribute), 1043

TEXTCONTAINER (pynlpl.formats.folia.New attribute), 976

TEXTCONTAINER (pynlpl.formats.folia.Note attribute), 254

TEXTCONTAINER (pynlpl.formats.folia.Observation attribute), 565

TEXTCONTAINER (pynlpl.formats.folia.ObservationLayer attribute), 694

TEXTCONTAINER (pynlpl.formats.folia.Original attribute), 987

TEXTCONTAINER (pynlpl.formats.folia.Paragraph attribute), 267

TEXTCONTAINER (pynlpl.formats.folia.Part attribute), 280

TEXTCONTAINER (pynlpl.formats.folia.PhonContent attribute), 506

TEXTCONTAINER (pynlpl.formats.folia.PosAnnotation attribute), 440

TEXTCONTAINER (pynlpl.formats.folia.Predicate attribute), 576

TEXTCONTAINER (pynlpl.formats.folia.Quote attribute), 293

TEXTCONTAINER (pynlpl.formats.folia.Reference attribute), 306

TEXTCONTAINER (pynlpl.formats.folia.Row attribute), 319

TEXTCONTAINER (pynlpl.formats.folia.SemanticRole attribute), 623

TEXTCONTAINER (pynlpl.formats.folia.SemanticRolesLayer attribute), 741

TEXTCONTAINER (pynlpl.formats.folia.SenseAnnotation attribute), 473

TEXTCONTAINER (pynlpl.formats.folia.Sentence attribute), 333

TEXTCONTAINER (pynlpl.formats.folia.Sentiment attribute), 588

TEXTCONTAINER (pynlpl.formats.folia.SentimentLayer attribute), 705

TEXTCONTAINER (pynlpl.formats.folia.Statement attribute), 599

TEXTCONTAINER (pynlpl.formats.folia.StatementLayer attribute), 717

TEXTCONTAINER (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484

TEXTCONTAINER (pynlpl.formats.folia.Suggestion attribute), 998

TEXTCONTAINER (pynlpl.formats.folia.SynsetFeature attribute), 874

TEXTCONTAINER (pynlpl.formats.folia.SyntacticUnit attribute), 611

TEXTCONTAINER (pynlpl.formats.folia.SyntaxLayer attribute), 729

TEXTCONTAINER (pynlpl.formats.folia.Table attribute), 347

TEXTCONTAINER (pynlpl.formats.folia.TableHead attribute), 373

TEXTCONTAINER (pynlpl.formats.folia.Term attribute), 360

TEXTCONTAINER (pynlpl.formats.folia.Text attribute), 386

TEXTCONTAINER (pynlpl.formats.folia.TextContent attribute), 495

TEXTCONTAINER (pynlpl.formats.folia.TextMarkupCorrection attribute), 840

TEXTCONTAINER (pynlpl.formats.folia.TextMarkupError attribute), 851

TEXTCONTAINER (pynlpl.formats.folia.TextMarkupGap attribute), 808

TEXTCONTAINER (pynlpl.formats.folia.TextMarkupString attribute), 819

TEXTCONTAINER (pynlpl.formats.folia.TextMarkupStyle attribute), 830

TEXTCONTAINER (pynlpl.formats.folia.TimeSegment attribute), 634

TEXTCONTAINER (pynlpl.formats.folia.TimingLayer attribute), 752

TEXTCONTAINER (pynlpl.formats.folia.Whitespace attribute), 399

TEXTCONTAINER (pynlpl.formats.folia.Word attribute), 413

TextContent (class in pynlpl.formats.folia), 492

textcontent() (pynlpl.formats.folia.AbstractAnnotationLayer method), 83

textcontent() (pynlpl.formats.folia.AbstractElement method), 33

textcontent() (pynlpl.formats.folia.AbstractSpanAnnotation method), 60

textcontent() (pynlpl.formats.folia.AbstractStructureElement method), 46

textcontent() (pynlpl.formats.folia.AbstractTextMarkup method), 93

textcontent() (pynlpl.formats.folia.AbstractTokenAnnotation method), 71

textcontent() (pynlpl.formats.folia.ActorFeature method), 892

textcontent() (pynlpl.formats.folia.Alignment method), 1016

textcontent() (pynlpl.formats.folia.AlignReference method), 1027

textcontent() (pynlpl.formats.folia.Alternative method), 927

textcontent() (pynlpl.formats.folia.AlternativeLayers

- method), 939
- textcontent() (pynlpl.formats.folia.BegindatetimeFeature method), 903
- textcontent() (pynlpl.formats.folia.Cell method), 109
- textcontent() (pynlpl.formats.folia.Chunk method), 526
- textcontent() (pynlpl.formats.folia.ChunkingLayer method), 654
- textcontent() (pynlpl.formats.folia.CoreferenceChain method), 537
- textcontent() (pynlpl.formats.folia.CoreferenceLayer method), 666
- textcontent() (pynlpl.formats.folia.CoreferenceLink method), 772
- textcontent() (pynlpl.formats.folia.Correction method), 950
- textcontent() (pynlpl.formats.folia.Current method), 961
- textcontent() (pynlpl.formats.folia.Definition method), 122
- textcontent() (pynlpl.formats.folia.DependenciesLayer method), 678
- textcontent() (pynlpl.formats.folia.Dependency method), 549
- textcontent() (pynlpl.formats.folia.DependencyDependent method), 784
- textcontent() (pynlpl.formats.folia.Description method), 1039
- textcontent() (pynlpl.formats.folia.Division method), 135
- textcontent() (pynlpl.formats.folia.DomainAnnotation method), 436
- textcontent() (pynlpl.formats.folia.EnddatetimeFeature method), 914
- textcontent() (pynlpl.formats.folia.EntitiesLayer method), 690
- textcontent() (pynlpl.formats.folia.Entity method), 561
- textcontent() (pynlpl.formats.folia.Entry method), 148
- textcontent() (pynlpl.formats.folia.ErrorDetection method), 972
- textcontent() (pynlpl.formats.folia.Event method), 161
- textcontent() (pynlpl.formats.folia.Example method), 174
- textcontent() (pynlpl.formats.folia.Feature method), 870
- textcontent() (pynlpl.formats.folia.Figure method), 187
- textcontent() (pynlpl.formats.folia.Gap method), 198
- textcontent() (pynlpl.formats.folia.Head method), 211
- textcontent() (pynlpl.formats.folia.Headspan method), 796
- textcontent() (pynlpl.formats.folia.LangAnnotation method), 458
- textcontent() (pynlpl.formats.folia.LemmaAnnotation method), 469
- textcontent() (pynlpl.formats.folia.Linebreak method), 224
- textcontent() (pynlpl.formats.folia.List method), 237
- textcontent() (pynlpl.formats.folia.ListItem method), 250
- textcontent() (pynlpl.formats.folia.Metric method), 1050
- textcontent() (pynlpl.formats.folia.New method), 983
- textcontent() (pynlpl.formats.folia.Note method), 263
- textcontent() (pynlpl.formats.folia.Observation method), 572
- textcontent() (pynlpl.formats.folia.ObservationLayer method), 702
- textcontent() (pynlpl.formats.folia.Original method), 994
- textcontent() (pynlpl.formats.folia.Paragraph method), 276
- textcontent() (pynlpl.formats.folia.Part method), 289
- textcontent() (pynlpl.formats.folia.PhonContent method), 512
- textcontent() (pynlpl.formats.folia.PosAnnotation method), 447
- textcontent() (pynlpl.formats.folia.Predicate method), 584
- textcontent() (pynlpl.formats.folia.Quote method), 302
- textcontent() (pynlpl.formats.folia.Reference method), 315
- textcontent() (pynlpl.formats.folia.Row method), 328
- textcontent() (pynlpl.formats.folia.SemanticRole method), 631
- textcontent() (pynlpl.formats.folia.SemanticRolesLayer method), 749
- textcontent() (pynlpl.formats.folia.SenseAnnotation method), 480
- textcontent() (pynlpl.formats.folia.Sentence method), 343
- textcontent() (pynlpl.formats.folia.Sentiment method), 596
- textcontent() (pynlpl.formats.folia.SentimentLayer method), 713
- textcontent() (pynlpl.formats.folia.Statement method), 607
- textcontent() (pynlpl.formats.folia.StatementLayer method), 725
- textcontent() (pynlpl.formats.folia.SubjectivityAnnotation method), 491
- textcontent() (pynlpl.formats.folia.Suggestion method), 1005
- textcontent() (pynlpl.formats.folia.SynsetFeature method), 881
- textcontent() (pynlpl.formats.folia.SyntacticUnit method), 619
- textcontent() (pynlpl.formats.folia.SyntaxLayer method), 737
- textcontent() (pynlpl.formats.folia.Table method), 356
- textcontent() (pynlpl.formats.folia.TableHead method), 382
- textcontent() (pynlpl.formats.folia.Term method), 369
- textcontent() (pynlpl.formats.folia.Text method), 395
- textcontent() (pynlpl.formats.folia.TextContent method), 502
- textcontent() (pynlpl.formats.folia.TextMarkupCorrection method), 848

textcontent() (pynlpl.formats.folia.TextMarkupError method), 858

textcontent() (pynlpl.formats.folia.TextMarkupGap method), 816

textcontent() (pynlpl.formats.folia.TextMarkupString method), 826

textcontent() (pynlpl.formats.folia.TextMarkupStyle method), 837

textcontent() (pynlpl.formats.folia.TimeSegment method), 642

textcontent() (pynlpl.formats.folia.TimingLayer method), 760

textcontent() (pynlpl.formats.folia.Whitespace method), 408

textcontent() (pynlpl.formats.folia.Word method), 423

TEXTDELIMITER (pynlpl.formats.folia.AbstractAnnotation attribute), 75

TEXTDELIMITER (pynlpl.formats.folia.AbstractElement attribute), 26

TEXTDELIMITER (pynlpl.formats.folia.AbstractSpanAnnotation attribute), 52

TEXTDELIMITER (pynlpl.formats.folia.AbstractStructureElement attribute), 37

TEXTDELIMITER (pynlpl.formats.folia.AbstractTextMarkup attribute), 86

TEXTDELIMITER (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63

TEXTDELIMITER (pynlpl.formats.folia.ActorFeature attribute), 885

TEXTDELIMITER (pynlpl.formats.folia.Alignment attribute), 1009

TEXTDELIMITER (pynlpl.formats.folia.AlignReference attribute), 1020

TEXTDELIMITER (pynlpl.formats.folia.Alternative attribute), 919

TEXTDELIMITER (pynlpl.formats.folia.AlternativeLayers attribute), 931

TEXTDELIMITER (pynlpl.formats.folia.BegindatetimeFeature attribute), 896

TEXTDELIMITER (pynlpl.formats.folia.Cell attribute), 101

TEXTDELIMITER (pynlpl.formats.folia.Chunk attribute), 518

TEXTDELIMITER (pynlpl.formats.folia.ChunkingLayer attribute), 647

TEXTDELIMITER (pynlpl.formats.folia.CoreferenceChain attribute), 530

TEXTDELIMITER (pynlpl.formats.folia.CoreferenceLayer attribute), 658

TEXTDELIMITER (pynlpl.formats.folia.CoreferenceLink attribute), 764

TEXTDELIMITER (pynlpl.formats.folia.Correction attribute), 944

TEXTDELIMITER (pynlpl.formats.folia.Current attribute), 954

TEXTDELIMITER (pynlpl.formats.folia.Definition attribute), 113

TEXTDELIMITER (pynlpl.formats.folia.DependenciesLayer attribute), 670

TEXTDELIMITER (pynlpl.formats.folia.Dependency attribute), 541

TEXTDELIMITER (pynlpl.formats.folia.DependencyDependent attribute), 776

TEXTDELIMITER (pynlpl.formats.folia.Description attribute), 1032

TEXTDELIMITER (pynlpl.formats.folia.Division attribute), 126

TEXTDELIMITER (pynlpl.formats.folia.DomainAnnotation attribute), 429

TEXTDELIMITER (pynlpl.formats.folia.EnddatetimeFeature attribute), 907

TEXTDELIMITER (pynlpl.formats.folia.EntitiesLayer attribute), 682

TEXTDELIMITER (pynlpl.formats.folia.Entity attribute), 553

TEXTDELIMITER (pynlpl.formats.folia.Entry attribute), 139

TEXTDELIMITER (pynlpl.formats.folia.ErrorDetection attribute), 965

TEXTDELIMITER (pynlpl.formats.folia.Event attribute), 152

TEXTDELIMITER (pynlpl.formats.folia.Example attribute), 165

TEXTDELIMITER (pynlpl.formats.folia.Feature attribute), 862

TEXTDELIMITER (pynlpl.formats.folia.Figure attribute), 178

TEXTDELIMITER (pynlpl.formats.folia.Gap attribute), 191

TEXTDELIMITER (pynlpl.formats.folia.Head attribute), 203

TEXTDELIMITER (pynlpl.formats.folia.Headspan attribute), 788

TEXTDELIMITER (pynlpl.formats.folia.LangAnnotation attribute), 451

TEXTDELIMITER (pynlpl.formats.folia.LemmaAnnotation attribute), 462

TEXTDELIMITER (pynlpl.formats.folia.Linebreak attribute), 215

TEXTDELIMITER (pynlpl.formats.folia.List attribute), 228

TEXTDELIMITER (pynlpl.formats.folia.ListItem attribute), 241

TEXTDELIMITER (pynlpl.formats.folia.Metric attribute), 1043

TEXTDELIMITER (pynlpl.formats.folia.New attribute), 976

TEXTDELIMITER (pynlpl.formats.folia.Note attribute),

- 254
- TEXTDELIMITER (pynlpl.formats.folia.Observation attribute), 565
- TEXTDELIMITER (pynlpl.formats.folia.ObservationLayer attribute), 694
- TEXTDELIMITER (pynlpl.formats.folia.Original attribute), 987
- TEXTDELIMITER (pynlpl.formats.folia.Paragraph attribute), 267
- TEXTDELIMITER (pynlpl.formats.folia.Part attribute), 280
- TEXTDELIMITER (pynlpl.formats.folia.PhonContent attribute), 506
- TEXTDELIMITER (pynlpl.formats.folia.PosAnnotation attribute), 440
- TEXTDELIMITER (pynlpl.formats.folia.Predicate attribute), 576
- TEXTDELIMITER (pynlpl.formats.folia.Quote attribute), 293
- TEXTDELIMITER (pynlpl.formats.folia.Reference attribute), 306
- TEXTDELIMITER (pynlpl.formats.folia.Row attribute), 319
- TEXTDELIMITER (pynlpl.formats.folia.SemanticRole attribute), 623
- TEXTDELIMITER (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- TEXTDELIMITER (pynlpl.formats.folia.SenseAnnotation attribute), 473
- TEXTDELIMITER (pynlpl.formats.folia.Sentence attribute), 333
- TEXTDELIMITER (pynlpl.formats.folia.Sentiment attribute), 588
- TEXTDELIMITER (pynlpl.formats.folia.SentimentLayer attribute), 706
- TEXTDELIMITER (pynlpl.formats.folia.Statement attribute), 599
- TEXTDELIMITER (pynlpl.formats.folia.StatementLayer attribute), 717
- TEXTDELIMITER (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- TEXTDELIMITER (pynlpl.formats.folia.Suggestion attribute), 998
- TEXTDELIMITER (pynlpl.formats.folia.SynsetFeature attribute), 874
- TEXTDELIMITER (pynlpl.formats.folia.SyntacticUnit attribute), 611
- TEXTDELIMITER (pynlpl.formats.folia.SyntaxLayer attribute), 729
- TEXTDELIMITER (pynlpl.formats.folia.Table attribute), 347
- TEXTDELIMITER (pynlpl.formats.folia.TableHead attribute), 373
- TEXTDELIMITER (pynlpl.formats.folia.Term attribute), 360
- TEXTDELIMITER (pynlpl.formats.folia.Text attribute), 386
- TEXTDELIMITER (pynlpl.formats.folia.TextContent attribute), 495
- TEXTDELIMITER (pynlpl.formats.folia.TextMarkupCorrection attribute), 841
- TEXTDELIMITER (pynlpl.formats.folia.TextMarkupError attribute), 851
- TEXTDELIMITER (pynlpl.formats.folia.TextMarkupGap attribute), 809
- TEXTDELIMITER (pynlpl.formats.folia.TextMarkupString attribute), 819
- TEXTDELIMITER (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- TEXTDELIMITER (pynlpl.formats.folia.TimeSegment attribute), 634
- TEXTDELIMITER (pynlpl.formats.folia.TimingLayer attribute), 753
- TEXTDELIMITER (pynlpl.formats.folia.Whitespace attribute), 399
- TEXTDELIMITER (pynlpl.formats.folia.Word attribute), 413
- TextMarkupCorrection (class in pynlpl.formats.folia), 838
- TextMarkupError (class in pynlpl.formats.folia), 849
- TextMarkupGap (class in pynlpl.formats.folia), 806
- TextMarkupString (class in pynlpl.formats.folia), 817
- TextMarkupStyle (class in pynlpl.formats.folia), 827
- textvalidation() (pynlpl.formats.folia.AbstractAnnotationLayer method), 83
- textvalidation() (pynlpl.formats.folia.AbstractElement method), 33
- textvalidation() (pynlpl.formats.folia.AbstractSpanAnnotation method), 60
- textvalidation() (pynlpl.formats.folia.AbstractStructureElement method), 47
- textvalidation() (pynlpl.formats.folia.AbstractTextMarkup method), 94
- textvalidation() (pynlpl.formats.folia.AbstractTokenAnnotation method), 71
- textvalidation() (pynlpl.formats.folia.ActorFeature method), 893
- textvalidation() (pynlpl.formats.folia.Alignment method), 1017
- textvalidation() (pynlpl.formats.folia.AlignReference method), 1028
- textvalidation() (pynlpl.formats.folia.Alternative method), 928
- textvalidation() (pynlpl.formats.folia.AlternativeLayers method), 939
- textvalidation() (pynlpl.formats.folia.BegindatetimeFeature method), 904
- textvalidation() (pynlpl.formats.folia.Cell method), 110



`textvalidation()` (`pynlpl.formats.folia.Chunk` method), 526  
`textvalidation()` (`pynlpl.formats.folia.ChunkingLayer` method), 655  
`textvalidation()` (`pynlpl.formats.folia.CoreferenceChain` method), 538  
`textvalidation()` (`pynlpl.formats.folia.CoreferenceLayer` method), 667  
`textvalidation()` (`pynlpl.formats.folia.CoreferenceLink` method), 773  
`textvalidation()` (`pynlpl.formats.folia.Correction` method), 950  
`textvalidation()` (`pynlpl.formats.folia.Current` method), 961  
`textvalidation()` (`pynlpl.formats.folia.Definition` method), 123  
`textvalidation()` (`pynlpl.formats.folia.DependenciesLayer` method), 678  
`textvalidation()` (`pynlpl.formats.folia.Dependency` method), 549  
`textvalidation()` (`pynlpl.formats.folia.DependencyDependent` method), 784  
`textvalidation()` (`pynlpl.formats.folia.Description` method), 1039  
`textvalidation()` (`pynlpl.formats.folia.Division` method), 136  
`textvalidation()` (`pynlpl.formats.folia.DomainAnnotation` method), 436  
`textvalidation()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 915  
`textvalidation()` (`pynlpl.formats.folia.EntitiesLayer` method), 690  
`textvalidation()` (`pynlpl.formats.folia.Entity` method), 561  
`textvalidation()` (`pynlpl.formats.folia.Entry` method), 149  
`textvalidation()` (`pynlpl.formats.folia.ErrorDetection` method), 972  
`textvalidation()` (`pynlpl.formats.folia.Event` method), 162  
`textvalidation()` (`pynlpl.formats.folia.Example` method), 175  
`textvalidation()` (`pynlpl.formats.folia.Feature` method), 870  
`textvalidation()` (`pynlpl.formats.folia.Figure` method), 188  
`textvalidation()` (`pynlpl.formats.folia.Gap` method), 199  
`textvalidation()` (`pynlpl.formats.folia.Head` method), 212  
`textvalidation()` (`pynlpl.formats.folia.Headspan` method), 796  
`textvalidation()` (`pynlpl.formats.folia.LangAnnotation` method), 458  
`textvalidation()` (`pynlpl.formats.folia.LemmaAnnotation` method), 469  
`textvalidation()` (`pynlpl.formats.folia.Linebreak` method), 224  
`textvalidation()` (`pynlpl.formats.folia.List` method), 238  
`textvalidation()` (`pynlpl.formats.folia.ListItem` method), 251  
`textvalidation()` (`pynlpl.formats.folia.Metric` method), 1050  
`textvalidation()` (`pynlpl.formats.folia.New` method), 983  
`textvalidation()` (`pynlpl.formats.folia.Note` method), 264  
`textvalidation()` (`pynlpl.formats.folia.Observation` method), 573  
`textvalidation()` (`pynlpl.formats.folia.ObservationLayer` method), 702  
`textvalidation()` (`pynlpl.formats.folia.Original` method), 994  
`textvalidation()` (`pynlpl.formats.folia.Paragraph` method), 277  
`textvalidation()` (`pynlpl.formats.folia.Part` method), 290  
`textvalidation()` (`pynlpl.formats.folia.PhonContent` method), 513  
`textvalidation()` (`pynlpl.formats.folia.PosAnnotation` method), 447  
`textvalidation()` (`pynlpl.formats.folia.Predicate` method), 584  
`textvalidation()` (`pynlpl.formats.folia.Quote` method), 303  
`textvalidation()` (`pynlpl.formats.folia.Reference` method), 316  
`textvalidation()` (`pynlpl.formats.folia.Row` method), 329  
`textvalidation()` (`pynlpl.formats.folia.SemanticRole` method), 631  
`textvalidation()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 749  
`textvalidation()` (`pynlpl.formats.folia.SenseAnnotation` method), 480  
`textvalidation()` (`pynlpl.formats.folia.Sentence` method), 343  
`textvalidation()` (`pynlpl.formats.folia.Sentiment` method), 596  
`textvalidation()` (`pynlpl.formats.folia.SentimentLayer` method), 714  
`textvalidation()` (`pynlpl.formats.folia.Statement` method), 608  
`textvalidation()` (`pynlpl.formats.folia.StatementLayer` method), 725  
`textvalidation()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 491  
`textvalidation()` (`pynlpl.formats.folia.Suggestion` method), 1005  
`textvalidation()` (`pynlpl.formats.folia.SynsetFeature` method), 881  
`textvalidation()` (`pynlpl.formats.folia.SyntacticUnit` method), 619  
`textvalidation()` (`pynlpl.formats.folia.SyntaxLayer` method), 737  
`textvalidation()` (`pynlpl.formats.folia.Table` method), 356  
`textvalidation()` (`pynlpl.formats.folia.TableHead` method), 382  
`textvalidation()` (`pynlpl.formats.folia.Term` method), 369  
`textvalidation()` (`pynlpl.formats.folia.Text` method), 395

textvalidation() (pynlpl.formats.folia.TextContent method), 502  
 textvalidation() (pynlpl.formats.folia.TextMarkupCorrection method), 848  
 textvalidation() (pynlpl.formats.folia.TextMarkupError method), 859  
 textvalidation() (pynlpl.formats.folia.TextMarkupGap method), 816  
 textvalidation() (pynlpl.formats.folia.TextMarkupString method), 827  
 textvalidation() (pynlpl.formats.folia.TextMarkupStyle method), 837  
 textvalidation() (pynlpl.formats.folia.TimeSegment method), 643  
 textvalidation() (pynlpl.formats.folia.TimingLayer method), 761  
 textvalidation() (pynlpl.formats.folia.Whitespace method), 408  
 textvalidation() (pynlpl.formats.folia.Word method), 423  
 TimblOutput (class in pynlpl.formats.timbl), 1055  
 TimeSegment (class in pynlpl.formats.folia), 632  
 TimingLayer (class in pynlpl.formats.folia), 750  
 title() (pynlpl.formats.folia.Document method), 20  
 tokenise() (in module pynlpl.textprocessors), 1067  
 tokenize() (in module pynlpl.textprocessors), 1067  
 Tokenizer (class in pynlpl.textprocessors), 1066  
 tokens() (pynlpl.statistics.FrequencyList method), 1063  
 toktext() (pynlpl.formats.folia.AbstractAnnotationLayer method), 83  
 toktext() (pynlpl.formats.folia.AbstractElement method), 34  
 toktext() (pynlpl.formats.folia.AbstractSpanAnnotation method), 60  
 toktext() (pynlpl.formats.folia.AbstractStructureElement method), 47  
 toktext() (pynlpl.formats.folia.AbstractTextMarkup method), 94  
 toktext() (pynlpl.formats.folia.AbstractTokenAnnotation method), 71  
 toktext() (pynlpl.formats.folia.ActorFeature method), 893  
 toktext() (pynlpl.formats.folia.Alignment method), 1017  
 toktext() (pynlpl.formats.folia.AlignReference method), 1028  
 toktext() (pynlpl.formats.folia.Alternative method), 928  
 toktext() (pynlpl.formats.folia.AlternativeLayers method), 939  
 toktext() (pynlpl.formats.folia.BegindatetimeFeature method), 904  
 toktext() (pynlpl.formats.folia.Cell method), 110  
 toktext() (pynlpl.formats.folia.Chunk method), 526  
 toktext() (pynlpl.formats.folia.ChunkingLayer method), 655  
 toktext() (pynlpl.formats.folia.CoreferenceChain method), 538  
 toktext() (pynlpl.formats.folia.CoreferenceLayer method), 667  
 toktext() (pynlpl.formats.folia.CoreferenceLink method), 773  
 toktext() (pynlpl.formats.folia.Correction method), 950  
 toktext() (pynlpl.formats.folia.Current method), 961  
 toktext() (pynlpl.formats.folia.Definition method), 123  
 toktext() (pynlpl.formats.folia.DependenciesLayer method), 679  
 toktext() (pynlpl.formats.folia.Dependency method), 550  
 toktext() (pynlpl.formats.folia.DependencyDependent method), 784  
 toktext() (pynlpl.formats.folia.Description method), 1039  
 toktext() (pynlpl.formats.folia.Division method), 136  
 toktext() (pynlpl.formats.folia.DomainAnnotation method), 436  
 toktext() (pynlpl.formats.folia.EnddatetimeFeature method), 915  
 toktext() (pynlpl.formats.folia.EntitiesLayer method), 690  
 toktext() (pynlpl.formats.folia.Entity method), 561  
 toktext() (pynlpl.formats.folia.Entry method), 149  
 toktext() (pynlpl.formats.folia.ErrorDetection method), 972  
 toktext() (pynlpl.formats.folia.Event method), 162  
 toktext() (pynlpl.formats.folia.Example method), 175  
 toktext() (pynlpl.formats.folia.Feature method), 870  
 toktext() (pynlpl.formats.folia.Figure method), 188  
 toktext() (pynlpl.formats.folia.Gap method), 199  
 toktext() (pynlpl.formats.folia.Head method), 212  
 toktext() (pynlpl.formats.folia.Headspan method), 796  
 toktext() (pynlpl.formats.folia.LangAnnotation method), 458  
 toktext() (pynlpl.formats.folia.LemmaAnnotation method), 469  
 toktext() (pynlpl.formats.folia.Linebreak method), 225  
 toktext() (pynlpl.formats.folia.List method), 238  
 toktext() (pynlpl.formats.folia.ListItem method), 251  
 toktext() (pynlpl.formats.folia.Metric method), 1050  
 toktext() (pynlpl.formats.folia.New method), 983  
 toktext() (pynlpl.formats.folia.Note method), 264  
 toktext() (pynlpl.formats.folia.Observation method), 573  
 toktext() (pynlpl.formats.folia.ObservationLayer method), 702  
 toktext() (pynlpl.formats.folia.Original method), 994  
 toktext() (pynlpl.formats.folia.Paragraph method), 277  
 toktext() (pynlpl.formats.folia.Part method), 290  
 toktext() (pynlpl.formats.folia.PhonContent method), 513  
 toktext() (pynlpl.formats.folia.PosAnnotation method), 447  
 toktext() (pynlpl.formats.folia.Predicate method), 584  
 toktext() (pynlpl.formats.folia.Quote method), 303  
 toktext() (pynlpl.formats.folia.Reference method), 316  
 toktext() (pynlpl.formats.folia.Row method), 329

`toktext()` (`pynlpl.formats.folia.SemanticRole` method), 631

`toktext()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 749

`toktext()` (`pynlpl.formats.folia.SenseAnnotation` method), 480

`toktext()` (`pynlpl.formats.folia.Sentence` method), 343

`toktext()` (`pynlpl.formats.folia.Sentiment` method), 596

`toktext()` (`pynlpl.formats.folia.SentimentLayer` method), 714

`toktext()` (`pynlpl.formats.folia.Statement` method), 608

`toktext()` (`pynlpl.formats.folia.StatementLayer` method), 726

`toktext()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 491

`toktext()` (`pynlpl.formats.folia.Suggestion` method), 1005

`toktext()` (`pynlpl.formats.folia.SynsetFeature` method), 882

`toktext()` (`pynlpl.formats.folia.SyntacticUnit` method), 619

`toktext()` (`pynlpl.formats.folia.SyntaxLayer` method), 737

`toktext()` (`pynlpl.formats.folia.Table` method), 356

`toktext()` (`pynlpl.formats.folia.TableHead` method), 382

`toktext()` (`pynlpl.formats.folia.Term` method), 369

`toktext()` (`pynlpl.formats.folia.Text` method), 396

`toktext()` (`pynlpl.formats.folia.TextContent` method), 502

`toktext()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 848

`toktext()` (`pynlpl.formats.folia.TextMarkupError` method), 859

`toktext()` (`pynlpl.formats.folia.TextMarkupGap` method), 816

`toktext()` (`pynlpl.formats.folia.TextMarkupString` method), 827

`toktext()` (`pynlpl.formats.folia.TextMarkupStyle` method), 837

`toktext()` (`pynlpl.formats.folia.TimeSegment` method), 643

`toktext()` (`pynlpl.formats.folia.TimingLayer` method), 761

`toktext()` (`pynlpl.formats.folia.Whitespace` method), 409

`toktext()` (`pynlpl.formats.folia.Word` method), 424

`tp_rate()` (`pynlpl.evaluation.ClassEvaluation` method), 10

`traversal()` (`pynlpl.search.AbstractSearch` method), 1059

`traversal()` (`pynlpl.search.IterativeDeepening` method), 1060

`traversalsize()` (`pynlpl.search.AbstractSearch` method), 1059

`traversalsize()` (`pynlpl.search.IterativeDeepening` method), 1060

`Tree` (class in `pynlpl.datatypes`), 6

`Trie` (class in `pynlpl.datatypes`), 6

`typetokenratio()` (`pynlpl.statistics.FrequencyList` method), 1063

## U

`u()` (in module `pynlpl.common`), 3

`unalias()` (`pynlpl.formats.folia.Document` method), 20

`updatetext()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 83

`updatetext()` (`pynlpl.formats.folia.AbstractElement` method), 34

`updatetext()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 60

`updatetext()` (`pynlpl.formats.folia.AbstractStructureElement` method), 47

`updatetext()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 94

`updatetext()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 71

`updatetext()` (`pynlpl.formats.folia.ActorFeature` method), 893

`updatetext()` (`pynlpl.formats.folia.Alignment` method), 1017

`updatetext()` (`pynlpl.formats.folia.AlignReference` method), 1028

`updatetext()` (`pynlpl.formats.folia.Alternative` method), 928

`updatetext()` (`pynlpl.formats.folia.AlternativeLayers` method), 939

`updatetext()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 904

`updatetext()` (`pynlpl.formats.folia.Cell` method), 110

`updatetext()` (`pynlpl.formats.folia.Chunk` method), 526

`updatetext()` (`pynlpl.formats.folia.ChunkingLayer` method), 655

`updatetext()` (`pynlpl.formats.folia.CoreferenceChain` method), 538

`updatetext()` (`pynlpl.formats.folia.CoreferenceLayer` method), 667

`updatetext()` (`pynlpl.formats.folia.CoreferenceLink` method), 773

`updatetext()` (`pynlpl.formats.folia.Correction` method), 950

`updatetext()` (`pynlpl.formats.folia.Current` method), 961

`updatetext()` (`pynlpl.formats.folia.Definition` method), 123

`updatetext()` (`pynlpl.formats.folia.DependenciesLayer` method), 679

`updatetext()` (`pynlpl.formats.folia.Dependency` method), 550

`updatetext()` (`pynlpl.formats.folia.DependencyDependent` method), 784

`updatetext()` (`pynlpl.formats.folia.Description` method), 1039

`updatetext()` (`pynlpl.formats.folia.Division` method), 136

`updatetext()` (`pynlpl.formats.folia.DomainAnnotation` method), 436

`updatetext()` (`pynlpl.formats.folia.EnddatetimeFeature`



- method), 915
  - updatetext() (pynlpl.formats.folia.EntitiesLayer method), 690
  - updatetext() (pynlpl.formats.folia.Entity method), 561
  - updatetext() (pynlpl.formats.folia.Entry method), 149
  - updatetext() (pynlpl.formats.folia.ErrorDetection method), 972
  - updatetext() (pynlpl.formats.folia.Event method), 162
  - updatetext() (pynlpl.formats.folia.Example method), 175
  - updatetext() (pynlpl.formats.folia.Feature method), 870
  - updatetext() (pynlpl.formats.folia.Figure method), 188
  - updatetext() (pynlpl.formats.folia.Gap method), 199
  - updatetext() (pynlpl.formats.folia.Head method), 212
  - updatetext() (pynlpl.formats.folia.Headspan method), 796
  - updatetext() (pynlpl.formats.folia.LangAnnotation method), 458
  - updatetext() (pynlpl.formats.folia.LemmaAnnotation method), 469
  - updatetext() (pynlpl.formats.folia.Linebreak method), 225
  - updatetext() (pynlpl.formats.folia.List method), 238
  - updatetext() (pynlpl.formats.folia.ListItem method), 251
  - updatetext() (pynlpl.formats.folia.Metric method), 1050
  - updatetext() (pynlpl.formats.folia.New method), 983
  - updatetext() (pynlpl.formats.folia.Note method), 264
  - updatetext() (pynlpl.formats.folia.Observation method), 573
  - updatetext() (pynlpl.formats.folia.ObservationLayer method), 702
  - updatetext() (pynlpl.formats.folia.Original method), 994
  - updatetext() (pynlpl.formats.folia.Paragraph method), 277
  - updatetext() (pynlpl.formats.folia.Part method), 290
  - updatetext() (pynlpl.formats.folia.PhonContent method), 513
  - updatetext() (pynlpl.formats.folia.PosAnnotation method), 447
  - updatetext() (pynlpl.formats.folia.Predicate method), 584
  - updatetext() (pynlpl.formats.folia.Quote method), 303
  - updatetext() (pynlpl.formats.folia.Reference method), 316
  - updatetext() (pynlpl.formats.folia.Row method), 329
  - updatetext() (pynlpl.formats.folia.SemanticRole method), 631
  - updatetext() (pynlpl.formats.folia.SemanticRolesLayer method), 749
  - updatetext() (pynlpl.formats.folia.SenseAnnotation method), 480
  - updatetext() (pynlpl.formats.folia.Sentence method), 343
  - updatetext() (pynlpl.formats.folia.Sentiment method), 596
  - updatetext() (pynlpl.formats.folia.SentimentLayer method), 714
  - updatetext() (pynlpl.formats.folia.Statement method), 608
  - updatetext() (pynlpl.formats.folia.StatementLayer method), 726
  - updatetext() (pynlpl.formats.folia.SubjectivityAnnotation method), 491
  - updatetext() (pynlpl.formats.folia.Suggestion method), 1005
  - updatetext() (pynlpl.formats.folia.SynsetFeature method), 882
  - updatetext() (pynlpl.formats.folia.SyntacticUnit method), 619
  - updatetext() (pynlpl.formats.folia.SyntaxLayer method), 737
  - updatetext() (pynlpl.formats.folia.Table method), 356
  - updatetext() (pynlpl.formats.folia.TableHead method), 382
  - updatetext() (pynlpl.formats.folia.Term method), 369
  - updatetext() (pynlpl.formats.folia.Text method), 396
  - updatetext() (pynlpl.formats.folia.TextContent method), 502
  - updatetext() (pynlpl.formats.folia.TextMarkupCorrection method), 848
  - updatetext() (pynlpl.formats.folia.TextMarkupError method), 859
  - updatetext() (pynlpl.formats.folia.TextMarkupGap method), 816
  - updatetext() (pynlpl.formats.folia.TextMarkupString method), 827
  - updatetext() (pynlpl.formats.folia.TextMarkupStyle method), 837
  - updatetext() (pynlpl.formats.folia.TimeSegment method), 643
  - updatetext() (pynlpl.formats.folia.TimingLayer method), 761
  - updatetext() (pynlpl.formats.folia.Whitespace method), 409
  - updatetext() (pynlpl.formats.folia.Word method), 424
- ## V
- validate() (pynlpl.formats.sonar.CorporusDocumentX method), 1054
  - values() (pynlpl.statistics.Distribution method), 1062
  - values() (pynlpl.statistics.FrequencyList method), 1063
  - vector\_add() (in module pynlpl.statistics), 1064
  - visited() (pynlpl.search.AbstractSearch method), 1059
  - viterbi() (pynlpl.statistics.HiddenMarkovModel method), 1063
- ## W
- wait() (pynlpl.evaluation.AbstractExperiment method), 9
  - walk() (pynlpl.datatypes.Trie method), 7
  - Whitespace (class in pynlpl.formats.folia), 396
  - Windower (class in pynlpl.textprocessors), 1066
  - Word (class in pynlpl.formats.folia), 409

WordAlignment (class in `pynlpl.formats.giza`), 1054  
`words()` (`pynlpl.formats.folia.AbstractStructureElement` method), 47  
`words()` (`pynlpl.formats.folia.Cell` method), 110  
`words()` (`pynlpl.formats.folia.Definition` method), 123  
`words()` (`pynlpl.formats.folia.Division` method), 136  
`words()` (`pynlpl.formats.folia.Document` method), 20  
`words()` (`pynlpl.formats.folia.Entry` method), 149  
`words()` (`pynlpl.formats.folia.Event` method), 162  
`words()` (`pynlpl.formats.folia.Example` method), 175  
`words()` (`pynlpl.formats.folia.Figure` method), 188  
`words()` (`pynlpl.formats.folia.Head` method), 212  
`words()` (`pynlpl.formats.folia.Linebreak` method), 225  
`words()` (`pynlpl.formats.folia.List` method), 238  
`words()` (`pynlpl.formats.folia.ListItem` method), 251  
`words()` (`pynlpl.formats.folia.Note` method), 264  
`words()` (`pynlpl.formats.folia.Paragraph` method), 277  
`words()` (`pynlpl.formats.folia.Part` method), 290  
`words()` (`pynlpl.formats.folia.Quote` method), 303  
`words()` (`pynlpl.formats.folia.Reference` method), 316  
`words()` (`pynlpl.formats.folia.Row` method), 329  
`words()` (`pynlpl.formats.folia.Sentence` method), 344  
`words()` (`pynlpl.formats.folia.Table` method), 357  
`words()` (`pynlpl.formats.folia.TableHead` method), 383  
`words()` (`pynlpl.formats.folia.Term` method), 370  
`words()` (`pynlpl.formats.folia.Text` method), 396  
`words()` (`pynlpl.formats.folia.Whitespace` method), 409  
`words()` (`pynlpl.formats.folia.Word` method), 424  
`words()` (`pynlpl.formats.sonar.CorpusDocument` method), 1054  
`words()` (`pynlpl.formats.sonar.CorpusDocumentX` method), 1055  
WPSParmSearch (class in `pynlpl.evaluation`), 10  
`wrefs()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 60  
`wrefs()` (`pynlpl.formats.folia.Chunk` method), 526  
`wrefs()` (`pynlpl.formats.folia.CoreferenceChain` method), 538  
`wrefs()` (`pynlpl.formats.folia.CoreferenceLink` method), 773  
`wrefs()` (`pynlpl.formats.folia.Dependency` method), 550  
`wrefs()` (`pynlpl.formats.folia.DependencyDependent` method), 785  
`wrefs()` (`pynlpl.formats.folia.Entity` method), 561  
`wrefs()` (`pynlpl.formats.folia.Headspan` method), 796  
`wrefs()` (`pynlpl.formats.folia.Observation` method), 573  
`wrefs()` (`pynlpl.formats.folia.Predicate` method), 585  
`wrefs()` (`pynlpl.formats.folia.SemanticRole` method), 631  
`wrefs()` (`pynlpl.formats.folia.Sentiment` method), 596  
`wrefs()` (`pynlpl.formats.folia.Statement` method), 608  
`wrefs()` (`pynlpl.formats.folia.SyntacticUnit` method), 620  
`wrefs()` (`pynlpl.formats.folia.TimeSegment` method), 643  
`write()` (`pynlpl.formats.taggedata.Taggedata` method), 1055

## X

`XLINK` (`pynlpl.formats.folia.AbstractAnnotationLayer` attribute), 75  
`XLINK` (`pynlpl.formats.folia.AbstractElement` attribute), 26  
`XLINK` (`pynlpl.formats.folia.AbstractSpanAnnotation` attribute), 52  
`XLINK` (`pynlpl.formats.folia.AbstractStructureElement` attribute), 37  
`XLINK` (`pynlpl.formats.folia.AbstractTextMarkup` attribute), 86  
`XLINK` (`pynlpl.formats.folia.AbstractTokenAnnotation` attribute), 63  
`XLINK` (`pynlpl.formats.folia.ActorFeature` attribute), 885  
`XLINK` (`pynlpl.formats.folia.Alignment` attribute), 1009  
`XLINK` (`pynlpl.formats.folia.AlignReference` attribute), 1020  
`XLINK` (`pynlpl.formats.folia.Alternative` attribute), 919  
`XLINK` (`pynlpl.formats.folia.AlternativeLayers` attribute), 931  
`XLINK` (`pynlpl.formats.folia.BegindatetimeFeature` attribute), 896  
`XLINK` (`pynlpl.formats.folia.Cell` attribute), 101  
`XLINK` (`pynlpl.formats.folia.Chunk` attribute), 518  
`XLINK` (`pynlpl.formats.folia.ChunkingLayer` attribute), 647  
`XLINK` (`pynlpl.formats.folia.CoreferenceChain` attribute), 530  
`XLINK` (`pynlpl.formats.folia.CoreferenceLayer` attribute), 658  
`XLINK` (`pynlpl.formats.folia.CoreferenceLink` attribute), 764  
`XLINK` (`pynlpl.formats.folia.Correction` attribute), 944  
`XLINK` (`pynlpl.formats.folia.Current` attribute), 954  
`XLINK` (`pynlpl.formats.folia.Definition` attribute), 114  
`XLINK` (`pynlpl.formats.folia.DependenciesLayer` attribute), 670  
`XLINK` (`pynlpl.formats.folia.Dependency` attribute), 541  
`XLINK` (`pynlpl.formats.folia.DependencyDependent` attribute), 776  
`XLINK` (`pynlpl.formats.folia.Description` attribute), 1032  
`XLINK` (`pynlpl.formats.folia.Division` attribute), 126  
`XLINK` (`pynlpl.formats.folia.DomainAnnotation` attribute), 429  
`XLINK` (`pynlpl.formats.folia.EnddatetimeFeature` attribute), 907  
`XLINK` (`pynlpl.formats.folia.EntitiesLayer` attribute), 682  
`XLINK` (`pynlpl.formats.folia.Entity` attribute), 553  
`XLINK` (`pynlpl.formats.folia.Entry` attribute), 139  
`XLINK` (`pynlpl.formats.folia.ErrorDetection` attribute), 965  
`XLINK` (`pynlpl.formats.folia.Event` attribute), 152  
`XLINK` (`pynlpl.formats.folia.Example` attribute), 165

- XLINK (pynlpl.formats.folia.Feature attribute), 863
- XLINK (pynlpl.formats.folia.Figure attribute), 178
- XLINK (pynlpl.formats.folia.Gap attribute), 191
- XLINK (pynlpl.formats.folia.Head attribute), 203
- XLINK (pynlpl.formats.folia.Headspan attribute), 788
- XLINK (pynlpl.formats.folia.LangAnnotation attribute), 451
- XLINK (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- XLINK (pynlpl.formats.folia.Linebreak attribute), 216
- XLINK (pynlpl.formats.folia.List attribute), 228
- XLINK (pynlpl.formats.folia.ListItem attribute), 241
- XLINK (pynlpl.formats.folia.Metric attribute), 1043
- XLINK (pynlpl.formats.folia.New attribute), 976
- XLINK (pynlpl.formats.folia.Note attribute), 254
- XLINK (pynlpl.formats.folia.Observation attribute), 565
- XLINK (pynlpl.formats.folia.ObservationLayer attribute), 694
- XLINK (pynlpl.formats.folia.Original attribute), 987
- XLINK (pynlpl.formats.folia.Paragraph attribute), 267
- XLINK (pynlpl.formats.folia.Part attribute), 280
- XLINK (pynlpl.formats.folia.PhonContent attribute), 506
- XLINK (pynlpl.formats.folia.PosAnnotation attribute), 440
- XLINK (pynlpl.formats.folia.Predicate attribute), 576
- XLINK (pynlpl.formats.folia.Quote attribute), 293
- XLINK (pynlpl.formats.folia.Reference attribute), 306
- XLINK (pynlpl.formats.folia.Row attribute), 319
- XLINK (pynlpl.formats.folia.SemanticRole attribute), 623
- XLINK (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- XLINK (pynlpl.formats.folia.SenseAnnotation attribute), 473
- XLINK (pynlpl.formats.folia.Sentence attribute), 333
- XLINK (pynlpl.formats.folia.Sentiment attribute), 588
- XLINK (pynlpl.formats.folia.SentimentLayer attribute), 706
- XLINK (pynlpl.formats.folia.Statement attribute), 600
- XLINK (pynlpl.formats.folia.StatementLayer attribute), 717
- XLINK (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- XLINK (pynlpl.formats.folia.Suggestion attribute), 998
- XLINK (pynlpl.formats.folia.SynsetFeature attribute), 874
- XLINK (pynlpl.formats.folia.SyntacticUnit attribute), 611
- XLINK (pynlpl.formats.folia.SyntaxLayer attribute), 729
- XLINK (pynlpl.formats.folia.Table attribute), 347
- XLINK (pynlpl.formats.folia.TableHead attribute), 373
- XLINK (pynlpl.formats.folia.Term attribute), 360
- XLINK (pynlpl.formats.folia.Text attribute), 386
- XLINK (pynlpl.formats.folia.TextContent attribute), 495
- XLINK (pynlpl.formats.folia.TextMarkupCorrection attribute), 841
- XLINK (pynlpl.formats.folia.TextMarkupError attribute), 851
- XLINK (pynlpl.formats.folia.TextMarkupGap attribute), 809
- XLINK (pynlpl.formats.folia.TextMarkupString attribute), 819
- XLINK (pynlpl.formats.folia.TextMarkupStyle attribute), 830
- XLINK (pynlpl.formats.folia.TimeSegment attribute), 635
- XLINK (pynlpl.formats.folia.TimingLayer attribute), 753
- XLINK (pynlpl.formats.folia.Whitespace attribute), 399
- XLINK (pynlpl.formats.folia.Word attribute), 413
- xml() (pynlpl.formats.folia.AbstractAnnotationLayer method), 83
- xml() (pynlpl.formats.folia.AbstractElement method), 34
- xml() (pynlpl.formats.folia.AbstractSpanAnnotation method), 60
- xml() (pynlpl.formats.folia.AbstractStructureElement method), 47
- xml() (pynlpl.formats.folia.AbstractTextMarkup method), 94
- xml() (pynlpl.formats.folia.AbstractTokenAnnotation method), 71
- xml() (pynlpl.formats.folia.ActorFeature method), 893
- xml() (pynlpl.formats.folia.Alignment method), 1017
- xml() (pynlpl.formats.folia.AlignReference method), 1028
- xml() (pynlpl.formats.folia.Alternative method), 928
- xml() (pynlpl.formats.folia.AlternativeLayers method), 939
- xml() (pynlpl.formats.folia.BegindatetimeFeature method), 904
- xml() (pynlpl.formats.folia.Cell method), 110
- xml() (pynlpl.formats.folia.Chunk method), 526
- xml() (pynlpl.formats.folia.ChunkingLayer method), 655
- xml() (pynlpl.formats.folia.CoreferenceChain method), 538
- xml() (pynlpl.formats.folia.CoreferenceLayer method), 667
- xml() (pynlpl.formats.folia.CoreferenceLink method), 773
- xml() (pynlpl.formats.folia.Correction method), 950
- xml() (pynlpl.formats.folia.Current method), 961
- xml() (pynlpl.formats.folia.Definition method), 123
- xml() (pynlpl.formats.folia.DependenciesLayer method), 679
- xml() (pynlpl.formats.folia.Dependency method), 550
- xml() (pynlpl.formats.folia.DependencyDependent method), 785
- xml() (pynlpl.formats.folia.Description method), 1039
- xml() (pynlpl.formats.folia.Division method), 136

- `xml()` (`pynlpl.formats.folia.Document` method), 21
- `xml()` (`pynlpl.formats.folia.DomainAnnotation` method), 437
- `xml()` (`pynlpl.formats.folia.EnddatetimeFeature` method), 915
- `xml()` (`pynlpl.formats.folia.EntitiesLayer` method), 691
- `xml()` (`pynlpl.formats.folia.Entity` method), 561
- `xml()` (`pynlpl.formats.folia.Entry` method), 149
- `xml()` (`pynlpl.formats.folia.ErrorDetection` method), 973
- `xml()` (`pynlpl.formats.folia.Event` method), 162
- `xml()` (`pynlpl.formats.folia.Example` method), 175
- `xml()` (`pynlpl.formats.folia.Feature` method), 871
- `xml()` (`pynlpl.formats.folia.Figure` method), 188
- `xml()` (`pynlpl.formats.folia.Gap` method), 199
- `xml()` (`pynlpl.formats.folia.Head` method), 212
- `xml()` (`pynlpl.formats.folia.Headspan` method), 796
- `xml()` (`pynlpl.formats.folia.LangAnnotation` method), 459
- `xml()` (`pynlpl.formats.folia.LemmaAnnotation` method), 470
- `xml()` (`pynlpl.formats.folia.Linebreak` method), 225
- `xml()` (`pynlpl.formats.folia.List` method), 238
- `xml()` (`pynlpl.formats.folia.ListItem` method), 251
- `xml()` (`pynlpl.formats.folia.Metric` method), 1051
- `xml()` (`pynlpl.formats.folia.New` method), 984
- `xml()` (`pynlpl.formats.folia.Note` method), 264
- `xml()` (`pynlpl.formats.folia.Observation` method), 573
- `xml()` (`pynlpl.formats.folia.ObservationLayer` method), 702
- `xml()` (`pynlpl.formats.folia.Original` method), 995
- `xml()` (`pynlpl.formats.folia.Paragraph` method), 277
- `xml()` (`pynlpl.formats.folia.Part` method), 290
- `xml()` (`pynlpl.formats.folia.PhonContent` method), 513
- `xml()` (`pynlpl.formats.folia.PosAnnotation` method), 448
- `xml()` (`pynlpl.formats.folia.Predicate` method), 585
- `xml()` (`pynlpl.formats.folia.Quote` method), 303
- `xml()` (`pynlpl.formats.folia.Reference` method), 316
- `xml()` (`pynlpl.formats.folia.Row` method), 329
- `xml()` (`pynlpl.formats.folia.SemanticRole` method), 631
- `xml()` (`pynlpl.formats.folia.SemanticRolesLayer` method), 749
- `xml()` (`pynlpl.formats.folia.SenseAnnotation` method), 481
- `xml()` (`pynlpl.formats.folia.Sentence` method), 344
- `xml()` (`pynlpl.formats.folia.Sentiment` method), 596
- `xml()` (`pynlpl.formats.folia.SentimentLayer` method), 714
- `xml()` (`pynlpl.formats.folia.Statement` method), 608
- `xml()` (`pynlpl.formats.folia.StatementLayer` method), 726
- `xml()` (`pynlpl.formats.folia.SubjectivityAnnotation` method), 492
- `xml()` (`pynlpl.formats.folia.Suggestion` method), 1006
- `xml()` (`pynlpl.formats.folia.SynsetFeature` method), 882
- `xml()` (`pynlpl.formats.folia.SyntacticUnit` method), 620
- `xml()` (`pynlpl.formats.folia.SyntaxLayer` method), 738
- `xml()` (`pynlpl.formats.folia.Table` method), 357
- `xml()` (`pynlpl.formats.folia.TableHead` method), 383
- `xml()` (`pynlpl.formats.folia.Term` method), 370
- `xml()` (`pynlpl.formats.folia.Text` method), 396
- `xml()` (`pynlpl.formats.folia.TextContent` method), 502
- `xml()` (`pynlpl.formats.folia.TextMarkupCorrection` method), 848
- `xml()` (`pynlpl.formats.folia.TextMarkupError` method), 859
- `xml()` (`pynlpl.formats.folia.TextMarkupGap` method), 816
- `xml()` (`pynlpl.formats.folia.TextMarkupString` method), 827
- `xml()` (`pynlpl.formats.folia.TextMarkupStyle` method), 837
- `xml()` (`pynlpl.formats.folia.TimeSegment` method), 643
- `xml()` (`pynlpl.formats.folia.TimingLayer` method), 761
- `xml()` (`pynlpl.formats.folia.Whitespace` method), 409
- `xml()` (`pynlpl.formats.folia.Word` method), 424
- `xmldeclarations()` (`pynlpl.formats.folia.Document` method), 21
- `xmlmetadata()` (`pynlpl.formats.folia.Document` method), 21
- `xmlstring()` (`pynlpl.formats.folia.AbstractAnnotationLayer` method), 83
- `xmlstring()` (`pynlpl.formats.folia.AbstractElement` method), 34
- `xmlstring()` (`pynlpl.formats.folia.AbstractSpanAnnotation` method), 60
- `xmlstring()` (`pynlpl.formats.folia.AbstractStructureElement` method), 47
- `xmlstring()` (`pynlpl.formats.folia.AbstractTextMarkup` method), 94
- `xmlstring()` (`pynlpl.formats.folia.AbstractTokenAnnotation` method), 71
- `xmlstring()` (`pynlpl.formats.folia.ActorFeature` method), 893
- `xmlstring()` (`pynlpl.formats.folia.Alignment` method), 1017
- `xmlstring()` (`pynlpl.formats.folia.AlignReference` method), 1028
- `xmlstring()` (`pynlpl.formats.folia.Alternative` method), 928
- `xmlstring()` (`pynlpl.formats.folia.AlternativeLayers` method), 939
- `xmlstring()` (`pynlpl.formats.folia.BegindatetimeFeature` method), 904
- `xmlstring()` (`pynlpl.formats.folia.Cell` method), 110
- `xmlstring()` (`pynlpl.formats.folia.Chunk` method), 526
- `xmlstring()` (`pynlpl.formats.folia.ChunkingLayer` method), 655
- `xmlstring()` (`pynlpl.formats.folia.CoreferenceChain` method), 538
- `xmlstring()` (`pynlpl.formats.folia.CoreferenceLayer` method), 667
- `xmlstring()` (`pynlpl.formats.folia.CoreferenceLink`



- method), 773
- xmlstring() (pynlpl.formats.folia.Correction method), 951
- xmlstring() (pynlpl.formats.folia.Current method), 962
- xmlstring() (pynlpl.formats.folia.Definition method), 123
- xmlstring() (pynlpl.formats.folia.DependenciesLayer method), 679
- xmlstring() (pynlpl.formats.folia.Dependency method), 550
- xmlstring() (pynlpl.formats.folia.DependencyDependent method), 785
- xmlstring() (pynlpl.formats.folia.Description method), 1040
- xmlstring() (pynlpl.formats.folia.Division method), 136
- xmlstring() (pynlpl.formats.folia.Document method), 21
- xmlstring() (pynlpl.formats.folia.DomainAnnotation method), 437
- xmlstring() (pynlpl.formats.folia.EnddatetimeFeature method), 915
- xmlstring() (pynlpl.formats.folia.EntitiesLayer method), 691
- xmlstring() (pynlpl.formats.folia.Entity method), 561
- xmlstring() (pynlpl.formats.folia.Entry method), 149
- xmlstring() (pynlpl.formats.folia.ErrorDetection method), 973
- xmlstring() (pynlpl.formats.folia.Event method), 162
- xmlstring() (pynlpl.formats.folia.Example method), 175
- xmlstring() (pynlpl.formats.folia.Feature method), 871
- xmlstring() (pynlpl.formats.folia.Figure method), 188
- xmlstring() (pynlpl.formats.folia.Gap method), 199
- xmlstring() (pynlpl.formats.folia.Head method), 212
- xmlstring() (pynlpl.formats.folia.Headspan method), 796
- xmlstring() (pynlpl.formats.folia.LangAnnotation method), 459
- xmlstring() (pynlpl.formats.folia.LemmaAnnotation method), 470
- xmlstring() (pynlpl.formats.folia.Linebreak method), 225
- xmlstring() (pynlpl.formats.folia.List method), 238
- xmlstring() (pynlpl.formats.folia.ListItem method), 251
- xmlstring() (pynlpl.formats.folia.Metric method), 1051
- xmlstring() (pynlpl.formats.folia.New method), 984
- xmlstring() (pynlpl.formats.folia.Note method), 264
- xmlstring() (pynlpl.formats.folia.Observation method), 573
- xmlstring() (pynlpl.formats.folia.ObservationLayer method), 702
- xmlstring() (pynlpl.formats.folia.Original method), 995
- xmlstring() (pynlpl.formats.folia.Paragraph method), 277
- xmlstring() (pynlpl.formats.folia.Part method), 290
- xmlstring() (pynlpl.formats.folia.PhonContent method), 513
- xmlstring() (pynlpl.formats.folia.PosAnnotation method), 448
- xmlstring() (pynlpl.formats.folia.Predicate method), 585
- xmlstring() (pynlpl.formats.folia.Quote method), 303
- xmlstring() (pynlpl.formats.folia.Reference method), 316
- xmlstring() (pynlpl.formats.folia.Row method), 329
- xmlstring() (pynlpl.formats.folia.SemanticRole method), 631
- xmlstring() (pynlpl.formats.folia.SemanticRolesLayer method), 749
- xmlstring() (pynlpl.formats.folia.SenseAnnotation method), 481
- xmlstring() (pynlpl.formats.folia.Sentence method), 344
- xmlstring() (pynlpl.formats.folia.Sentiment method), 596
- xmlstring() (pynlpl.formats.folia.SentimentLayer method), 714
- xmlstring() (pynlpl.formats.folia.Statement method), 608
- xmlstring() (pynlpl.formats.folia.StatementLayer method), 726
- xmlstring() (pynlpl.formats.folia.SubjectivityAnnotation method), 492
- xmlstring() (pynlpl.formats.folia.Suggestion method), 1006
- xmlstring() (pynlpl.formats.folia.SynsetFeature method), 882
- xmlstring() (pynlpl.formats.folia.SyntacticUnit method), 620
- xmlstring() (pynlpl.formats.folia.SyntaxLayer method), 738
- xmlstring() (pynlpl.formats.folia.Table method), 357
- xmlstring() (pynlpl.formats.folia.TableHead method), 383
- xmlstring() (pynlpl.formats.folia.Term method), 370
- xmlstring() (pynlpl.formats.folia.Text method), 396
- xmlstring() (pynlpl.formats.folia.TextContent method), 502
- xmlstring() (pynlpl.formats.folia.TextMarkupCorrection method), 848
- xmlstring() (pynlpl.formats.folia.TextMarkupError method), 859
- xmlstring() (pynlpl.formats.folia.TextMarkupGap method), 816
- xmlstring() (pynlpl.formats.folia.TextMarkupString method), 827
- xmlstring() (pynlpl.formats.folia.TextMarkupStyle method), 837
- xmlstring() (pynlpl.formats.folia.TimeSegment method), 643
- xmlstring() (pynlpl.formats.folia.TimingLayer method), 761
- xmlstring() (pynlpl.formats.folia.Whitespace method), 409
- xmlstring() (pynlpl.formats.folia.Word method), 424
- XMLTAG (pynlpl.formats.folia.AbstractAnnotationLayer attribute), 75
- XMLTAG (pynlpl.formats.folia.AbstractElement attribute), 26
- XMLTAG (pynlpl.formats.folia.AbstractSpanAnnotation

- attribute), 52
- XMLTAG (pynlpl.formats.folia.AbstractStructureElement attribute), 37
- XMLTAG (pynlpl.formats.folia.AbstractTextMarkup attribute), 86
- XMLTAG (pynlpl.formats.folia.AbstractTokenAnnotation attribute), 63
- XMLTAG (pynlpl.formats.folia.ActorFeature attribute), 885
- XMLTAG (pynlpl.formats.folia.Alignment attribute), 1009
- XMLTAG (pynlpl.formats.folia.AlignReference attribute), 1020
- XMLTAG (pynlpl.formats.folia.Alternative attribute), 919
- XMLTAG (pynlpl.formats.folia.AlternativeLayers attribute), 931
- XMLTAG (pynlpl.formats.folia.BeginatetimeFeature attribute), 896
- XMLTAG (pynlpl.formats.folia.Cell attribute), 101
- XMLTAG (pynlpl.formats.folia.Chunk attribute), 518
- XMLTAG (pynlpl.formats.folia.ChunkingLayer attribute), 647
- XMLTAG (pynlpl.formats.folia.CoreferenceChain attribute), 530
- XMLTAG (pynlpl.formats.folia.CoreferenceLayer attribute), 658
- XMLTAG (pynlpl.formats.folia.CoreferenceLink attribute), 764
- XMLTAG (pynlpl.formats.folia.Correction attribute), 944
- XMLTAG (pynlpl.formats.folia.Current attribute), 954
- XMLTAG (pynlpl.formats.folia.Definition attribute), 114
- XMLTAG (pynlpl.formats.folia.DependenciesLayer attribute), 670
- XMLTAG (pynlpl.formats.folia.Dependency attribute), 541
- XMLTAG (pynlpl.formats.folia.DependencyDependent attribute), 776
- XMLTAG (pynlpl.formats.folia.Description attribute), 1032
- XMLTAG (pynlpl.formats.folia.Division attribute), 127
- XMLTAG (pynlpl.formats.folia.DomainAnnotation attribute), 429
- XMLTAG (pynlpl.formats.folia.EnddatetimeFeature attribute), 907
- XMLTAG (pynlpl.formats.folia.EntitiesLayer attribute), 682
- XMLTAG (pynlpl.formats.folia.Entity attribute), 553
- XMLTAG (pynlpl.formats.folia.Entry attribute), 139
- XMLTAG (pynlpl.formats.folia.ErrorDetection attribute), 965
- XMLTAG (pynlpl.formats.folia.Event attribute), 153
- XMLTAG (pynlpl.formats.folia.Example attribute), 165
- XMLTAG (pynlpl.formats.folia.Feature attribute), 863
- XMLTAG (pynlpl.formats.folia.Figure attribute), 179
- XMLTAG (pynlpl.formats.folia.Gap attribute), 191
- XMLTAG (pynlpl.formats.folia.Head attribute), 203
- XMLTAG (pynlpl.formats.folia.Headspeed attribute), 788
- XMLTAG (pynlpl.formats.folia.LangAnnotation attribute), 451
- XMLTAG (pynlpl.formats.folia.LemmaAnnotation attribute), 462
- XMLTAG (pynlpl.formats.folia.Linebreak attribute), 216
- XMLTAG (pynlpl.formats.folia.List attribute), 228
- XMLTAG (pynlpl.formats.folia.ListItem attribute), 241
- XMLTAG (pynlpl.formats.folia.Metric attribute), 1043
- XMLTAG (pynlpl.formats.folia.New attribute), 976
- XMLTAG (pynlpl.formats.folia.Note attribute), 254
- XMLTAG (pynlpl.formats.folia.Observation attribute), 565
- XMLTAG (pynlpl.formats.folia.ObservationLayer attribute), 694
- XMLTAG (pynlpl.formats.folia.Original attribute), 987
- XMLTAG (pynlpl.formats.folia.Paragraph attribute), 268
- XMLTAG (pynlpl.formats.folia.Part attribute), 281
- XMLTAG (pynlpl.formats.folia.PhonContent attribute), 506
- XMLTAG (pynlpl.formats.folia.PosAnnotation attribute), 440
- XMLTAG (pynlpl.formats.folia.Predicate attribute), 576
- XMLTAG (pynlpl.formats.folia.Quote attribute), 293
- XMLTAG (pynlpl.formats.folia.Reference attribute), 307
- XMLTAG (pynlpl.formats.folia.Row attribute), 319
- XMLTAG (pynlpl.formats.folia.SemanticRole attribute), 623
- XMLTAG (pynlpl.formats.folia.SemanticRolesLayer attribute), 741
- XMLTAG (pynlpl.formats.folia.SenseAnnotation attribute), 473
- XMLTAG (pynlpl.formats.folia.Sentence attribute), 333
- XMLTAG (pynlpl.formats.folia.Sentiment attribute), 588
- XMLTAG (pynlpl.formats.folia.SentimentLayer attribute), 706
- XMLTAG (pynlpl.formats.folia.Statement attribute), 600
- XMLTAG (pynlpl.formats.folia.StatementLayer attribute), 717
- XMLTAG (pynlpl.formats.folia.SubjectivityAnnotation attribute), 484
- XMLTAG (pynlpl.formats.folia.Suggestion attribute), 998
- XMLTAG (pynlpl.formats.folia.SynsetFeature attribute), 874
- XMLTAG (pynlpl.formats.folia.SyntacticUnit attribute), 611
- XMLTAG (pynlpl.formats.folia.SyntaxLayer attribute), 729
- XMLTAG (pynlpl.formats.folia.Table attribute), 347
- XMLTAG (pynlpl.formats.folia.TableHead attribute), 373

XMLTAG (pynlpl.formats.folia.Term attribute), [360](#)  
XMLTAG (pynlpl.formats.folia.Text attribute), [386](#)  
XMLTAG (pynlpl.formats.folia.TextContent attribute),  
[495](#)  
XMLTAG (pynlpl.formats.folia.TextMarkupCorrection  
attribute), [841](#)  
XMLTAG (pynlpl.formats.folia.TextMarkupError at-  
tribute), [851](#)  
XMLTAG (pynlpl.formats.folia.TextMarkupGap at-  
tribute), [809](#)  
XMLTAG (pynlpl.formats.folia.TextMarkupString  
attribute), [819](#)  
XMLTAG (pynlpl.formats.folia.TextMarkupStyle at-  
tribute), [830](#)  
XMLTAG (pynlpl.formats.folia.TimeSegment attribute),  
[635](#)  
XMLTAG (pynlpl.formats.folia.TimingLayer attribute),  
[753](#)  
XMLTAG (pynlpl.formats.folia.Whitespace attribute),  
[399](#)  
XMLTAG (pynlpl.formats.folia.Word attribute), [413](#)  
xpath() (pynlpl.formats.folia.Document method), [21](#)  
xpath() (pynlpl.formats.sonar.CorpusDocumentX  
method), [1055](#)