1 Reading FoLiA

1.1 Loading a document .................................................. 3
  1.1.1 folia.main.Document ........................................ 3
1.2 Printing text .......................................................... 14
1.3 Index .................................................................. 15
1.4 Elements ................................................................ 15
  1.4.1 folia.main.AbstractElement ................................. 15
  1.4.2 folia.main.AbstractStructureElement ...................... 28
  1.4.3 folia.main.AbstractSpanAnnotation ......................... 42
  1.4.4 folia.main.AbstractAnnotationLayer ....................... 55
  1.4.5 folia.main.AbstractTextMarkup .............................. 68
1.5 Obtaining list of elements ............................................ 79
1.6 Select method .......................................................... 80
1.7 Selection Shortcuts ..................................................... 80
1.8 Navigating a document ............................................... 81
1.9 Structure Annotation Types ........................................ 82
  1.9.1 folia.main.Cell .................................................. 83
  1.9.2 folia.main.Definition .......................................... 97
  1.9.3 folia.main.Division ............................................ 111
  1.9.4 folia.main.Entry ............................................... 125
  1.9.5 folia.main.Event ............................................... 139
  1.9.6 folia.main.Example ........................................... 153
  1.9.7 folia.main.Figure ............................................. 167
  1.9.8 folia.main.Gap .................................................. 181
  1.9.9 folia.main.Head ................................................. 193
  1.9.10 folia.main.Linebreak ......................................... 207
  1.9.11 folia.main.List ................................................ 221
  1.9.12 folia.main.ListItem .......................................... 235
  1.9.13 folia.main.Note ............................................... 249
  1.9.14 folia.main.Paragraph ........................................ 263
  1.9.15 folia.main.Paragraph ........................................ 277
  1.9.16 folia.main.Quote ............................................. 291
  1.9.17 folia.main.Reference .......................................... 305
  1.9.18 folia.main.Row ................................................ 319
  1.9.19 folia.main.Sentence .......................................... 333
  1.9.20 folia.main.Table ............................................. 348
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9.21 folia.main.Term</td>
<td>362</td>
</tr>
<tr>
<td>1.9.22 folia.main.TableHead</td>
<td>376</td>
</tr>
<tr>
<td>1.9.23 folia.main.Text</td>
<td>390</td>
</tr>
<tr>
<td>1.9.24 folia.main.Whitespace</td>
<td>404</td>
</tr>
<tr>
<td>1.9.25 folia.main.Word</td>
<td>418</td>
</tr>
<tr>
<td>1.10 Common attributes</td>
<td>435</td>
</tr>
<tr>
<td>1.11 Annotations</td>
<td>436</td>
</tr>
<tr>
<td>1.11.1 Inline Annotation Types</td>
<td>437</td>
</tr>
<tr>
<td>1.11.2 Text and phonetic annotation</td>
<td>509</td>
</tr>
<tr>
<td>1.11.3 Span Annotation</td>
<td>533</td>
</tr>
<tr>
<td>1.11.4 Span Annotation Types</td>
<td>533</td>
</tr>
<tr>
<td>2 Editing FoLiA</td>
<td>841</td>
</tr>
<tr>
<td>2.1 Creating a new document</td>
<td>841</td>
</tr>
<tr>
<td>2.2 Declarations</td>
<td>841</td>
</tr>
<tr>
<td>2.3 Adding structure</td>
<td>842</td>
</tr>
<tr>
<td>2.4 Adding annotations</td>
<td>842</td>
</tr>
<tr>
<td>2.5 Provenance Information</td>
<td>844</td>
</tr>
<tr>
<td>2.6 Adding span annotation</td>
<td>845</td>
</tr>
<tr>
<td>2.7 Deleting annotations</td>
<td>846</td>
</tr>
<tr>
<td>2.8 Copying annotations</td>
<td>846</td>
</tr>
<tr>
<td>3 Searching in a FoLiA document</td>
<td>847</td>
</tr>
<tr>
<td>3.1 Corpus Query Language (CQL)</td>
<td>847</td>
</tr>
<tr>
<td>3.2 FoLiA Query Language (FQL)</td>
<td>848</td>
</tr>
<tr>
<td>3.2.1 folia.fql.Query</td>
<td>848</td>
</tr>
<tr>
<td>3.3 Streaming Reader</td>
<td>851</td>
</tr>
<tr>
<td>4 Higher-Order Annotations</td>
<td>853</td>
</tr>
<tr>
<td>4.1 Text Markup</td>
<td>853</td>
</tr>
<tr>
<td>4.2 Features</td>
<td>853</td>
</tr>
<tr>
<td>4.3 Alternatives</td>
<td>854</td>
</tr>
<tr>
<td>4.4 Corrections</td>
<td>855</td>
</tr>
<tr>
<td>4.5 Relations</td>
<td>856</td>
</tr>
<tr>
<td>4.6 Descriptions, Metrics</td>
<td>856</td>
</tr>
<tr>
<td>5 Metadata</td>
<td>857</td>
</tr>
<tr>
<td>6 Indices and tables</td>
<td>859</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>861</td>
</tr>
<tr>
<td>Index</td>
<td>863</td>
</tr>
</tbody>
</table>
This Python module provides an extensive library for parsing, creating and otherwise processing documents in the Format for Linguistic Annotation, aka FoLiA. It has been in active development since 2010 and used by numerous Natural Language Processing (NLP) tools.

This tutorial will introduce the FoLiA Python library. The FoLiA library provides an Application Programming Interface for the reading, creation and manipulation of FoLiA XML documents. The library is written for Python 3.5 and above.

Prior to reading this document, it is recommended to first read the FoLiA documentation itself and familiarise yourself with the format and underlying paradigm. It is especially important to understand the way FoLiA handles sets/classes, declarations, common attributes such as annotator/annotatortype 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.

Contents:
1.1 Loading a document

Any script that uses FoLiA starts with the import:

```python
import folia.main as 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.
```

1.1.1 `folia.main.Document`

```python
class folia.main.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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Start/load a FoLiA document:</td>
</tr>
<tr>
<td><code>add(text)</code></td>
<td>Alias for <code>Document.append()</code></td>
</tr>
<tr>
<td><code>alias(…)</code></td>
<td>Return the alias for a set (if applicable, returns the unaltered set otherwise iff fallback is enabled)</td>
</tr>
<tr>
<td><code>append(text)</code></td>
<td>Add a text (or speech) to the document:</td>
</tr>
</tbody>
</table>
### Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attachexternal</code>&lt;span&gt;(type, set, **kwargs)&lt;/span&gt;</td>
<td>See <code>AbstractElement.count()</code></td>
</tr>
<tr>
<td><code>count</code>&lt;span&gt;(Class[, set, recursive, ignore])&lt;/span&gt;</td>
<td>Create an element associated with this Document.</td>
</tr>
<tr>
<td><code>date</code>&lt;span&gt;([value])&lt;/span&gt;</td>
<td>Get or set the document’s date from/in the metadata.</td>
</tr>
<tr>
<td><code>declare</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Declare new annotation types, sets or annotators to be used in the document.</td>
</tr>
<tr>
<td><code>declared</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Checks if the annotation type is present (i.e.</td>
</tr>
<tr>
<td><code>defaultannotator</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Obtain the default annotator for the specified annotation type and set.</td>
</tr>
<tr>
<td><code>defaultannotortype</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Obtain the default annotator type for the specified annotation type and set.</td>
</tr>
<tr>
<td><code>defaultdatetime</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Obtain the default datetime for the specified annotation type and set.</td>
</tr>
<tr>
<td><code>defaultset</code>&lt;span&gt;(annotationtype[, set])&lt;/span&gt;</td>
<td>Obtain the default set for the specified annotation type.</td>
</tr>
<tr>
<td><code>done</code>&lt;span&gt;(annotationtype)&lt;/span&gt;</td>
<td>Signal that you are done editing the document, this will perform any pending post-processing operation.</td>
</tr>
<tr>
<td><code>erase</code>&lt;span&gt;(Class[, annotationset])&lt;/span&gt;</td>
<td>Erases all annotations of a particular type and annotation set (unless set is False in which case it applies to all elements regardless of set).</td>
</tr>
<tr>
<td><code>findwords</code>&lt;span&gt;(*args, **kwargs)&lt;/span&gt;</td>
<td>Get all annotators for the given annotationtype and set.</td>
</tr>
<tr>
<td><code>getannotators</code>&lt;span&gt;(annotationtype, annotationset)&lt;/span&gt;</td>
<td>Get all processors associated with the given annotationtype and set, generator yielding Processor instances, see also :meth:<code>AbstractElement.getannotators</code></td>
</tr>
<tr>
<td><code>getdefaultprocessor</code>&lt;span&gt;(annotationtype, . . . )&lt;/span&gt;</td>
<td>Does this annotationtype and set have associated processors/annotators? (FoLiA v2 provenance data)</td>
</tr>
<tr>
<td><code>getprocessors</code>&lt;span&gt;(annotationtype, annotationset)&lt;/span&gt;</td>
<td>Does this annotationtype and set have defaults? (new style FoLiA v2 with provenance data)</td>
</tr>
<tr>
<td><code>hasannotators</code>&lt;span&gt;(annotationtype, annotationset)&lt;/span&gt;</td>
<td>Does this annotationtype and set have associated processors/annotators? (FoLiA v1 without provenance data)</td>
</tr>
<tr>
<td><code>hasdefaultprocessor</code>&lt;span&gt;(annotationtype, . . . )&lt;/span&gt;</td>
<td>Does this annotationtype and set have associated processors/annotators? (FoLiA v2 provenance data)</td>
</tr>
<tr>
<td><code>hasdefaults</code>&lt;span&gt;(annotationtype, annotationset)&lt;/span&gt;</td>
<td>Does this annotationtype and set have defaults? (old style FoLiA v1 without provenance data)</td>
</tr>
<tr>
<td><code>hasprocessors</code>&lt;span&gt;(annotationtype, annotationset)&lt;/span&gt;</td>
<td>Does this annotationtype and set have associated processors/annotators? (FoLiA v2 provenance data)</td>
</tr>
<tr>
<td><code>items</code>&lt;span&gt;()&lt;/span&gt;</td>
<td>Returns a depth-first flat list of all items in the document</td>
</tr>
<tr>
<td><code>json</code>&lt;span&gt;()&lt;/span&gt;</td>
<td>Serialise the document to a dict ready for serialisation to JSON.</td>
</tr>
<tr>
<td><code>jsondeclarations</code>&lt;span&gt;()&lt;/span&gt;</td>
<td>Return all declarations in a form ready to be serialised to JSON.</td>
</tr>
<tr>
<td><code>jsonprovenance</code>&lt;span&gt;()&lt;/span&gt;</td>
<td>Internal method to serialize provenance data to JSON</td>
</tr>
<tr>
<td><code>language</code>&lt;span&gt;([value])&lt;/span&gt;</td>
<td>No arguments: Get the document’s language (ISO-639-3) from metadata Argument: Set the document’s language (ISO-639-3) in metadata</td>
</tr>
<tr>
<td><code>license</code>&lt;span&gt;([value])&lt;/span&gt;</td>
<td>No arguments: Get the document’s license from metadata Argument: Set the document’s license in metadata</td>
</tr>
<tr>
<td><code>load</code>&lt;span&gt;(filename)&lt;/span&gt;</td>
<td>Load a FoLiA XML file.</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Return a generator of all paragraphs found in the document.</td>
</tr>
<tr>
<td><code>parsemetadata(node)</code></td>
<td>Internal method to parse metadata</td>
</tr>
<tr>
<td><code>parsesubmetadata(node)</code></td>
<td></td>
</tr>
<tr>
<td><code>parsexml(node[, ParentClass])</code></td>
<td>Internal method.</td>
</tr>
<tr>
<td><code>parsexmldeclarations(node)</code></td>
<td>Internal method to parse XML declarations</td>
</tr>
<tr>
<td><code>parsexmlprovenance(node)</code></td>
<td></td>
</tr>
<tr>
<td><code>pendingsort([warnonly])</code></td>
<td>Perform any pending sorts on span annotation elements (per layer, in turn recurses into all span annotations)</td>
</tr>
<tr>
<td><code>pendingvalidation([warnonly])</code></td>
<td>Perform any pending validations</td>
</tr>
<tr>
<td><code>publisher([value])</code></td>
<td>No arguments: Get the document’s publisher from metadata. Argument: Set the document’s publisher in metadata</td>
</tr>
<tr>
<td><code>save([filename])</code></td>
<td>Save the document to file.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore])</code></td>
<td>See <code>AbstractElement.select()</code></td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Return a generator of all sentence found in the document.</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, hidden])</code></td>
<td>Returns the text of the entire document (returns a unicode instance)</td>
</tr>
<tr>
<td><code>title([value])</code></td>
<td>Get or set the document’s title from/in the metadata</td>
</tr>
<tr>
<td><code>unalias(annotationtype, alias)</code></td>
<td>Return the set for an alias (if applicable, raises an exception otherwise)</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Return a generator of all active words found in the document.</td>
</tr>
<tr>
<td><code>xml()</code></td>
<td>Serialise the document to XML.</td>
</tr>
<tr>
<td><code>xmldeclarations()</code></td>
<td>Internal method to generate XML nodes for all declarations</td>
</tr>
<tr>
<td><code>xmlmetadata()</code></td>
<td>Internal method to serialize metadata to XML</td>
</tr>
<tr>
<td><code>xmlprovenance()</code></td>
<td>Internal method to serialize provenance data to XML</td>
</tr>
<tr>
<td><code>xmlstring()</code></td>
<td>Return the XML representation of the document as a string.</td>
</tr>
<tr>
<td><code>xpath(query)</code></td>
<td>Run Xpath expression and parse the resulting elements.</td>
</tr>
</tbody>
</table>

**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:
```

1.1. Loading a document
3) Load a document from an XML string:
```
doc = folia.Document(string='<FoLiA>....</FoLiA>')</n```

4) Load a document by passing a parse xml tree (xmltree):
```
    doc = folia.Document(tree=xmltree)
```

You will often want to associate a Processor when you instantiate a document, the processor encapsulates information regarding the tool that is processing a document (i.e. your script), and adds this to the document’s provenance chain. Any new annotations you add to this document will be automatically related to the processor:
```
doc = folia.Document(id="example", processor=Processor.create(name="my-tool", version="0.1"))
```

**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), this value is always forced to True to FoLiA v1.5 and above
- **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)
- **keepversion** (bool) – attempt to keep the FoLiA version (use with caution)
- **version** (str) – force a particular FoLiA version when creating a new document (use with caution)
- **declare** (list) – Declare the specifies annotation types. Consists of a list or tuple of annotation types or (annotation, set) tuples or (annotation type, set, processor) tuples
- **processor** (Processor) – Register the current processor in the provenance data and use this processor in all subsequent declarations.
- **reprocessor** (Processor) – As above, but will take pro-active ownership of any declarations already present but not tied to a processor yet.
- **debug** (bool) – Boolean to enable/disable debug
- **autodeclare** (bool) – Automatically declare annotation types and annotators whenever possible (enabled by default for FoLiA v2)
• **mode** – The mode for loading a document, is either `folia.Mode.MEMORY`, in which 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. Other mode is `folia.Mode.XPATH`, in which 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.

• **fixunassignedprocessor** (bool) – If set, fixes invalid FoLiA that does not explicitly assign a processor to an annotation when multiple processors are possible (and there is therefore no default). The last processor will be used in this case.

`__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:

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

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

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

3) Load a document from an XML string:

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

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

```python
doc = folia.Document(tree=xmltree)
```

You will often want to associate a Processor when you instantiate a document, the processor encapsulates information regarding the tool that is processing a document (i.e. your script), and adds this to the document’s provenance chain. Any new annotations you add to this document will be automatically related to the processor:

```python
doc = folia.Document(id="example", processor=Processor.create(name="my-tool", version="0.1"))
```

**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), this value is always forced to True to FoLiA v1.5 and above
- **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)

### 1.1. Loading a document
• **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)

• **keepversion** (*bool*) – attempt to keep the FoLiA version (use with caution)

• **version** (*str*) – force a particular FoLiA version when creating a new document (use 
with caution)

• **declare** (*list*) – Declare the specifies annotation types. Consists of a list or tuple of 
annotationtypes or (annotation,set) tuples or (annotationtype,set,processor) tuples

• **processor** (*Processor*) – Register the current processor in the provenance data and 
use this processor in all subsequent declarations.

• **reprocessor** (*Processor*) – As above, but will take pro-active ownership of any 
declarations already present but not tied to a processor yet.

• **debug** (*bool*) – Boolean to enable/disable debug

• **autodeclare** (*bool*) – Automatically declare annotation types and annotators whenever 
possible (enabled by default for FoLiA v2)

• **mode** – The mode for loading a document, is either folia.Mode.MEMORY, in which 
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, in which 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.

• **fixunassignedprocessor** (*bool*) – If set, fixes invalid FoLiA that does not explicitly 
assign a processor to an annotation when multiple processors are possible (and there 
is therefore no default). The last processor will be used in this case.

```
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:
   ```python
   doc.append(folia.Text)
   ```
   Example 2::
   ```python
   doc.append( folia.Text(doc, id='example.text') )
   ```
   Example 3:
   ```python
   doc.append(folia.Speech)
   ```

attachexternal(type, set, **kwargs)

count(Class, set=False, 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=None, *args, **kwargs)**

Declare new annotation types, sets or annotators to be used in the document.

This typically done by associating an annotationtype and set with a processor, the processor contains annotator information and will be recorded in the provenance data.

### 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

#### Positional Arguments:

- `processor` *(Processor or str)*: A processor to declare, can be a processor instance or an ID of an existing processor. The processor encapsulates all information of an annotator. If you specify multiple processors then they are parsed as a hierarchy, the first one being the root and the others subprocessors.

#### Keyword Arguments

- **alias** *(str)* – Defines alias that may be used in set attribute of elements instead of the full set name

- **generator** *(bool)* – Automatically append a subprocessor with generator information on the FoLiA library used? (default: True)

#### Keyword Arguments (<= FoLiA 1.5 behaviour, i.e. without provenance data):

- **annotator** *(str)*: Sets a default annotator old-style, i.e. without full provenance annotator. The processor encapsulates all information of an annotator. Time to reconsider. If you specify multiple processors then they are parsed as a hierarchy, the first one being the root and the others subprocessors.

- **datetime** *(datetime)*: Sets the default datetime

Example 1 (with provenance):

```python
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', Processor(name="mytagger"))
```

Example 2 (with provenance; nested processors):

```python
main_processor = Processor(name="myNLPtool", version="2.2")
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', main_processor, Processor(name="mytagger"))
doc.declare(folia.LemmaAnnotation, 'http://some/set', main_processor, Processor(name="mylemmatiser"))
```

Example 2b (with provenance; nested processors, same as above but setting main processor on Document instantiation instead):
Example 3 (with provenance; nested processors):

```
main_processor = Processor(name="myEditor", version="1.2")
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', main_processor, Processor(name="alice", type=AnnotatorType.MANUAL))
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', main_processor, Processor(name="bob", type=AnnotatorType.MANUAL))
doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', main_processor, Processor(name="john", type=AnnotatorType.MANUAL))
```

Example 4 (without provenance, for backward compatibility, the use of proper provenance is always preferred!):

```
```

Returns:: Processor instance of the last processor added (or None if no provenance is used)

`declared(annotationtype, set=False)`

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/None/False)** – the set, should formally be a URL pointing to the set definition (aliases are also supported). If set to False, checks regardless of set (i.e. matching any set). If set to None, there is no associated set.

Example:

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

Returns  bool

`defaultannotator(annotationtype, set=False)`

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/None/False)** – the set, should formally be a URL pointing to the set definition or None for setless annotations. If set to False, the default set will be inferred automatically, but an exception will occur if there is none!
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=False)

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/None/False) – the set, should formally be a URL pointing to the set definition or None for setless annotations. If set to False, the default set will be inferred automatically, but an exception will occur if there is none!

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=False)

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/None/False) – the set, should formally be a URL pointing to the set definition or None for setless annotations. If set to False, the default set will be inferred automatically, but an exception will occur if there is none!

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 or None), or False if there is no default set. Take care to explicitly distinguish between False and None!

Raises NoSuchAnnotation if the annotation type does not exist or if there is ambiguity (multiple sets for the same type). Or returns False instead if raiseexception = False

done ()

Signal that you are done editing the document, this will perform any pending post-processing operation

erase (Class, annotationset=False)

Erases all annotations of a particular type and annotation set (unless set is False in which case it applies to all elements regardless of set). Also removed the declarations (i.e. the opposite of declare())

findwords (*args, **kwargs)
getannotators (annotationtype, annotationset)
Get all annotators for the given annotationtype and set. This is a generator that yields Annotator instances, these resolve to a Processor when called. See also :meth:AbstractElement.getprocessors to obtain processors directly, which is most likely what you want.

getdefaultprocessor (annotationtype, annotationset)

getprocessors (annotationtype, annotationset)
Get all processors associated with the given annotationtype and set, generator yielding Processor instances, see also :meth:AbstractElement.getannotators

hasannotators (annotationtype, annotationset)
Alias for Document.hasprocessors(): Does this annotationtype and set have associated processors/annotators? (FoLiA v2 provenance data)

hasdefaultprocessor (annotationtype, annotationset)
Does this annotationtype and set have defaults? (new style FoLiA v2 with provenance data)

hasdefaults (annotationtype, annotationset)
Does this annotationtype and set have associated defaults? (old style FoLiA v1 without provenance data)

hasprocessors (annotationtype, annotationset)
Does this annotationtype and set have associated processors/annotators? (FoLiA v2 provenance data)

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:

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

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

    Returns  list of dict

jsonprovenance ()
Internal method to serialize provenance data to JSON

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)

parsecxml (node, ParentClass=None)
Internal method.
This is the main XML parser, will invoke class-specific XML parsers.

parsecxmldeclarations (node)
Internal method to parse XML declarations

parsexmlprovenance (node)

pendingsort (warnonly=None)
Perform any pending sorts on span annotation elements (per layer, in turn recurses into all span annotations)

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=False, 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)

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

1.1. Loading a document
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

xmlprovenance()  
Internal method to serialize provenance data 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 namesapace 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.

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()).

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)

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

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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AbstractElement</strong></td>
<td>Abstract base class from which all FoLiA elements are derived.</td>
</tr>
<tr>
<td><strong>AbstractStructureElement</strong></td>
<td>Abstract element, all structure elements inherit from this class.</td>
</tr>
<tr>
<td><strong>AllowTokenAnnotation</strong></td>
<td>Abstract element, all span annotation elements are derived from this class</td>
</tr>
<tr>
<td><strong>AbstractSpanAnnotation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AbstractTokenAnnotation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AbstractAnnotationLayer</strong></td>
<td>Annotation layers for Span Annotation are derived from this abstract base class</td>
</tr>
<tr>
<td><strong>AbstractTextMarkup</strong></td>
<td>Abstract class for text markup elements, elements that appear with the TextContent (t) element.</td>
</tr>
</tbody>
</table>

1.4.1 folia.main.AbstractElement

```
class folia.main.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 of 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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code></td>
<td></td>
</tr>
<tr>
<td><code>add</code></td>
<td></td>
</tr>
<tr>
<td><code>addable</code></td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>addidsuffix</code></td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addtoindex([norecurse])</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>ancestor(*Classes)</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors([Class])</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotator2processor([annotator, ...])</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append(child, *args, **kwargs)</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore, node])</code></td>
<td>Get the index at which an element occurs, recursively by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>originaltext(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext(cls)</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```python
ACCEPTED_DATA = (<class 'folia.main.Description'>, <class 'folia.main.Comment'>)
```

```python
ANNOTATIONTYPE = None
```

```python
AUTH = True
```

```python
AUTO_GENERATE_ID = False
```

```python
HIDDEN = False
```

```python
OCCURRENCES = 0
```

```python
OCCURRENCES_PER_SET = 0
```

```python
OPTIONAL_ATTRIBS = None
```

```python
PHONCONTAINER = False
```

```python
PRIMARYELEMENT = True
```

```python
PRINTABLE = False
```

```python
REQUIRED_ATTRIBS = None
```

```python
REQUIRED_DATA = None
```

```python
SETONLY = False
```

```python
SPEAKABLE = False
```

```python
SUBSET = None
```

```python
TEXTCONTAINER = False
```

```python
TEXTDELIMITER = None
```

```python
WREFABLE = False
```

```python
XLINK = False
```

```python
XMLTAG = None
```

Method Details

```python
__init__ (doc, *args, **kwargs)
```

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

```python
__init__ (doc, *args, **kwargs)
```

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

```python
classmethod accepts (Class, raiseexceptions=True, parentinstance=None)
```

1.4. Elements
add(child, *args, **kwargs)

classmethod addable (parent, set=False, 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, None, or False) – 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. Raise a NoSuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append(child, *args, **kwargs)

checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

`depthfirstsearch (function)`
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

`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:

```python
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=False, **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 TEXTDELMITER attribute but may return a customised one instead.

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

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, hidden=False )

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:

```python
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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
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 outputed, 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.

• **hidden** *(bool)* – Include hidden elements, defaults to *False*.

Example:

```python
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, hidden=False)*

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards()**

Internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, Alternative Layers, 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:
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()`

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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`. 

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

```
• **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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.

• **hidden** *(bool)* – Include hidden elements, defaults to False.

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
update(text) ()  
   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)  
   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()

1.4.2 folia.main.AbstractStructureElement
class folia.main.AbstractStructureElement (doc, *args, **kwargs)  
   Abstract element, all structure elements inherit from this class. Never instantiated directly.

Method Summary

   __init__ (doc, *args, **kwargs) Initialize self.
accepts (Class[
   raiseexceptions, parentinstance])
add (child, *args, **kwargs)
addable (parent[], set, raiseexceptions)  
   Tests whether a new element of this class can be added to the parent.
addidsuffix (idsuffix[], recursive)  
   Appends a suffix to this element’s ID, and optionally to all child IDs as well.
Continued on next page
### Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addtoindex([norecurse])</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>alternatives([Class, set, returnelements])</code></td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><code>ancestor(*Classes)</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors([Class])</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation(type[, set])</code></td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><code>annotations(Class[, set])</code></td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><code>annotator2processor([annotator, ...])</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append(child, *args, **kwargs)</code></td>
<td>See <code>AbstractElement.append()</code></td>
</tr>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
</tbody>
</table>

Continued on next page
Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes


`ANNOTATIONTYPE` = None

`AUTH` = True
`AUTOGENERATE_ID` = True
`HIDDEN` = False
`OCCURRENCES` = 0
`OCCURRENCES_PER_SET` = 0
`OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 5, 6, 7, 8, 9, 11, 13)
`PHONCONTAINER` = False
`PRIMARYELEMENT` = True
`PRINTABLE` = True
`REQUIRED_ATTRIBS` = None
`REQUIRED_DATA` = None
`SETONLY` = False
`SPEAKABLE` = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELMITER = '\n\n'
WREFABLE = False
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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.
Yields Alternative elements

ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
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 False (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:
AllowInlineAnnotation.annotations() AbstractElement.select()

Raises NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)
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 False (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

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)

See AbstractElement.append()

checkdeclaration ()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context (size, placeholde=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=False, 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

depthfirstsearch (function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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=False, **kwargs)
Internal method to find replaceable elements. Auxiliary function used by AbstractElement.replace(). Can be overriden 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=False)
Returns an integer indicating whether such an annotation exists, and if so, how many.

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

hasannotationlayer(annnotationtype=None, set=False)
Does the specified annotation layer exist?

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

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:

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

**Returns**: dict

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

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

**parsecommonarguments** (`doc, **kwargs`)  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False`)  
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 outputed, 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.

• **hidden** (`bool`) – Include hidden elements, defaults to `False`.

Example:

```python
word.phon()
```
Returns  The phonetic content of the element (\texttt{unicode} instance in Python 2, \texttt{str} in Python 3)

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

See also:

\texttt{phoncontent()}: Retrieves the phonetic content as an element rather than a string
\texttt{text()}

\texttt{phoncontent (cls='current', correctionhandling=1, hidden=False)}
Get the phonetic content explicitly associated with this element (of the specified class).

Unlike \texttt{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

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

Returns  The phonetic content (\texttt{PhonContent})

Raises  NoSuchPhon if there is no phonetic content for the element

See also:

\texttt{phon()} \texttt{textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes (other)}
Returns a boolean indicating whether this element precedes the other element

\texttt{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

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

\texttt{classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None)}
Returns a RelaxNG definition for this element (as an XML element (\texttt{ lxml.etree}) rather than a string)

\texttt{classmethod relaxng_backwards ()}
internal helper function for backward compatibility
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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

resolwword(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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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 \texttt{(newdoc)}
Set a different document. Usually no need to call this directly, invoked implicitly by \texttt{copy()}

setdocument \texttt{(doc)}
Associate a document with this element.

**Parameters**

- \texttt{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 \texttt{copy()}

setprocessor \texttt{(processor)}
Sets the processor for this element, taking care of adding an annotator in the declarations

settext \texttt{(text, cls='current')}
Set the text for this element.

**Parameters**

- \texttt{text (str)} – The text
- \texttt{cls (str)} – The class of the text, defaults to \texttt{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 \texttt{src} attribute directly.

**Returns**

- \texttt{str} or \texttt{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 \texttt{src} attribute directly.

**Returns**

- \texttt{str} or \texttt{None} if not found

stricttext \texttt{(cls='current')}
Alias for \texttt{text()} with \texttt{strict=True}

**text** \texttt{(cls='current', retaintokenisation=False, previousdelimiter='\”, strict=False, correctionhandling=1, normalize_spaces=False, hidden=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**

- \texttt{cls (str)} – The class of the text content to obtain, defaults to \texttt{current}.
- \texttt{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 \texttt{False}.
- \texttt{previousdelimiter (str)} – Can be set to a delimiter that was last outputed, useful when chaining calls to \texttt{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

• **hidden** (bool) – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)
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.

• **hidden** (bool) – Include hidden elements, defaults to False.

**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 words.

xml(attribs=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()

1.4.3 folia.main.AbstractSpanAnnotation

class folia.main.AbstractSpanAnnotation(doc, *args, **kwargs)

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

Method Summary

__init__(doc, *args, **kwargs) Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addidsuffix idsuffix[, recursive]</code></td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addtoindex [norecurse]</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td><code>ancestor *Classes</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors [Class]</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation type[, set]</code></td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotator2processor [annotator, . . . ]</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append child, *args, **kwargs</code></td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context size[, placeholder, scope]</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><code>copy [newdoc, idsuffix]</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren [newdoc, idsuffix]</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct **kwargs</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count Class[, set, recursive, ignore, node]</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch function</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat subset</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling cls</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables parent[, set]</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id cls</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata [key]</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter [retaintokenisation]</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation Class[, set]</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon cls, strict, correctionhandling, . . . ]</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext cls, strict, correctionhandling, . . . ]</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, . . .])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, . . .])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, . . .])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 7 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

#### Class Attributes

- **ACCEPTED_DATA** = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ..., <class 'folia.main.LinkReference'>, <class 'folia.main.Metric'>, <class 'folia.main.Relation'>)
- **ANNOTATIONTYPE** = None
- **AUTH** = True
- **AUTO_GENERATE_ID** = False
- **HIDDEN** = 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

---

1.4. Elements
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)

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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses). Not instances!
Yields elements (instances derived from AbstractElement)

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

**annotations (Class, set=False)**
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.

**annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)**
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append (child, *args, **kwargs)**
See AbstractElement.append()

**checkdeclaration ()**
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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 DeepValidationException
depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

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:
```python
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=False, **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=False)
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, hidden=False)
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, hidden=False)

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:

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

Returns dict

layer ()

Return the annotation layer this annotation pertains to

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* (*str*) – 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*

**parsecommonarguments** (*doc, **kwargs*)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

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

**Parameters**

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

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

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

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*.
- *realtaintokenisation* (*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 outputed, useful when chaining calls to *phon()*.
- *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.
- *hidden* (*bool*) – Include hidden elements, defaults to *False*.

Example:

```python
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, hidden=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

remove (child)
Removes the child element

1.4. Elements
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 to be an alternative.

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

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
   Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'cornetto', True, [folio.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().

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations.

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.

sort(force=False)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage.

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, hidden=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 outputed, 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

• **hidden** (bool) – Include hidden elements, defaults to False.

Example:

```python
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, *hidden*=False)

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.

• **hidden** (bool) – Include hidden elements, defaults to False.

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

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

See also:

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

`tkttext`(*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:**

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

**__len__**()
Returns the number of child elements under the current element.

**__str__**()
Alias for text()

### 1.4.4 folia.main.AbstractAnnotationLayer

class folia.main.AbstractAnnotationLayer(doc, *args, **kwargs)

- **Bases:** folia.main.AbstractElement, folia.main.AllowGenerateID, folia.main.AllowCorrections

Annotation layers for Span Annotation are derived from this abstract base class

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>(doc, *args, **kwargs) Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex</td>
<td>(norecurs) Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes) Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ancestors()</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation(type[, set])</code></td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations(Class[, set])</code></td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor([annotator, ...])</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append(child, *args, **kwargs)</code></td>
<td>See <code>AbstractElement.append()</code></td>
</tr>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan(*words)</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
</tbody>
</table>

Continued on next page
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>json()</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext()</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext()</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments()</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml()</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon()</code></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>phoncontent()</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes()</code></td>
<td>Returns a boolean indicating after an element is added whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous()</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng()</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><code>remove()</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace()</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets()</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword()</code></td>
<td></td>
</tr>
</tbody>
</table>
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>textcontent</strong> ([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textvalidation</strong> ([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><strong>toktext</strong> ([cls])</td>
<td>Alias for <strong>text()</strong> with retainTokenisation=True</td>
</tr>
<tr>
<td><strong>updatetext</strong> ()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><strong>xml</strong> ([attribs, elements, skipchildren])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td><strong>xmlstring</strong> ([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong><strong>iter</strong></strong> ()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong><strong>len</strong></strong> ()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong><strong>str</strong></strong> ()</td>
<td>Alias for <strong>text()</strong></td>
</tr>
</tbody>
</table>

**Class Attributes**

- ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>)
- ANNOTATIONTYPE = None
- AUTH = True
- AUTO_GENERATE_ID = False
- HIDDEN = False
- OCCURRENCES = 0
- OCCURRENCES_PER_SET = 0
- OPTIONAL_ATTRIBS = (0,)
- PHONCONTAINER = False
- PRIMARYELEMENT = True
- PRINTABLE = False
- REQUIRED_ATTRIBS = None
- REQUIRED_DATA = None
- SETONLY = True
- SPEAKABLE = False
- SUBSET = None
- TEXTCONTAINER = False
- TEXTDELMITER = None
- WREFABLE = False
- XKLINK = 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=False, 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, None, or False) – 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 sually 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. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (*+) – The possible classes (AbstractElement or subclasses) to se-
lect 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 (tuple of) classes (AbstractElement or subclasses).

Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if
none was found

annotations (Class, set=False)
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.

```python
annotator2processor(annotator=None, annotatortype=None, parentprocessor=None)
```

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

```python
append(child, *args, **kwargs)
```

See `AbstractElement.append()`.

```python
checkdeclaration()
```

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

```python
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.

```python
copy(newdoc=None, idsuffix=None)
```

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 appended 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

```python
copychildren(newdoc=None, idsuffix=None)
```

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

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

```python
correct(**kwargs)
```

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

```python
count(Class, set=False, recursive=True, ignore=True, node=None)
```

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

**Returns** int

```python
deepvalidation()
```

Perform deep validation of this element.

**Raises** DeepValidationException

```python
depthfirstsearch(function)
```

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

```python
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:

```python
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=False, **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=False)
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, hidden=False)
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, hidden=False)`

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:

```python
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

parsecommonarguments (doc, **kwargs)
   Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
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 outputed, 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.
   • hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)
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()
• **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 to be an alternative.

• **be an alternative.**(to) –

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

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:

```python
    ..
```

**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()`
setprocessor *(processor)*
Sets the processor for this element, taking care of adding an annotator in the declarations

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.

sort ()
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, hidden=False)*
Get the text associated with this element (of the specified class)

The text will be constructed from child-elements whereever 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 outputed, useful when chaining calls to *text()*.
- **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
• **hidden** (*bool*) – Include hidden elements, defaults to `False`.

Example:

```python
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, *hidden*=False)

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.

- **hidden** (*bool*) – Include hidden elements, defaults to `False`.

**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:

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

__len__
Returns the number of child elements under the current element.

__str__
Alias for text

### 1.4.5 folia.main.AbstractTextMarkup

class folia.main.AbstractTextMarkup

Bases: folia.main.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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>(doc, *args, **kwargs) See AbstractElement.<strong>init</strong>(), text is passed as a string in *args.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addsuffix</td>
<td>(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex</td>
<td>([norecurse]) Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes) Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors</td>
<td>([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>([annotator, ...]) Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td>() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context</td>
<td>(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy</td>
<td>([newdoc, idsuffix]) Make a deep copy of this element and all its children.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadatad([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasphon([cls, previousdelimiter, strict, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>See <code>AbstractElement.json()</code></td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, . . .])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>relaxng()</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolve()</code></td>
<td></td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>rightcontext(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text)</code></td>
<td>Sets the text content of the markup element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

- `ACCEPTED_DATA = (<class 'folia.main.AbstractTextMarkup'>, <class 'folia.main.Comment'>)`
- `ANNOTATIONTYPE = None`
- `AUTH = True`
- `AUTO_GENERATE_ID = False`
- `HIDDEN = 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 = ''
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:
```python
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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields  
elements (instances derived from AbstractElement)

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)  
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)  

checkdeclaration ()  
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)  
Like AbstractElement.select (), but instead of returning the elements, it merely counts them.

Returns  
int
depdeepvalidation ()  
Perform deep validation of this element.

Raises  
DeepValidationError
**depthfirstsearch** (*function*)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

**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:
```python
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=False, **kwargs*)
Internal method to find replaceable elements. Auxiliary function used by AbstractElement.replace(). Can be overriden 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, hidden=False*)
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, hidden=False)*

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`

**parsecommonarguments**(doc, **kwargs)**

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)`

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 outputed, 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.

• **hidden (bool)** – Include hidden elements, defaults to `False`.

**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

`textcontent ()`

`phoncontent (cls='current', correctionhandling=1, hidden=False)`

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.

**precedes(other)**

Returns a boolean indicating whether this element precedes the other element

**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’, 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 el-
  ement. 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)

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.(to)** –

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

**resolve()**

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls=’current’)

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

- **hidden (bool)** – Include hidden elements, defaults to False.

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, hidden=False)**  
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.

• **hidden (bool)** – Include hidden elements, defaults to False.

  **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:

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

  **__len__** ()

  Returns the number of child elements under the current element.

  **__str__** ()

  Alias for `text()`

1.5 Obtaining list of elements

The aforementioned index is useful only if you know the ID of the element. This if 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:

```python
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.

### 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:

```python
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:

```python
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:

```python
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.

### 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()`:

```python
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()`:

```python
for sentence in doc.sentences():
    for word in sentence.words():
        print(word)
```

Or including paragraphs, assuming the document has them:

```python
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:

```python
word = sentence.words(3)  #retrieves the fourth word
```

### 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 to go upwards or sideways. The `AbstractElement.next()` and `AbstractElement.previous()` methods can be used for side navigation, they return the next or previous element, respectively:

```python
nextelement = element.next()
previouselement = element.previous()
```

You can explicitly filter by passing an element type:

```python
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:

```python
nextword = word.next(folia.Word, [folia.Sentence])
```

If you do not want any constraints, pass `None`:

```python
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:

```python
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)

### 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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>A cell in a Row in a Table</td>
</tr>
<tr>
<td>Definition</td>
<td>Element used in Entry for the portion that provides a definition for the entry.</td>
</tr>
<tr>
<td>Division</td>
<td>Structure element representing some kind of division.</td>
</tr>
<tr>
<td>Entry</td>
<td>Represents an entry in a glossary/lexicon/dictionary.</td>
</tr>
<tr>
<td>Event</td>
<td>Structural element representing events, often used in new media contexts for things such as tweets, chat messages and forum posts.</td>
</tr>
<tr>
<td>Example</td>
<td>Element that provides an example.</td>
</tr>
<tr>
<td>Figure</td>
<td>Element for the representation of a graphical figure.</td>
</tr>
<tr>
<td>Gap</td>
<td>Gap element, represents skipped portions of the text.</td>
</tr>
<tr>
<td>Head</td>
<td>Head element; a structure element that acts as the header/title of a Division.</td>
</tr>
<tr>
<td>Linebreak</td>
<td>Line break element, signals a line break.</td>
</tr>
<tr>
<td>List</td>
<td>Element for enumeration/itemisation.</td>
</tr>
<tr>
<td>ListItem</td>
<td>Single element in a List.</td>
</tr>
<tr>
<td>Note</td>
<td>Element used for notes, such as footnotes or warnings or notice blocks.</td>
</tr>
<tr>
<td>Paragraph</td>
<td>Paragraph element.</td>
</tr>
<tr>
<td>Part</td>
<td>Generic structure element used to mark a part inside another block.</td>
</tr>
<tr>
<td>Quote</td>
<td>Quote: a structure element.</td>
</tr>
<tr>
<td>Reference</td>
<td>A structural element that denotes a reference, internal or external.</td>
</tr>
<tr>
<td>Row</td>
<td>A row in a Table</td>
</tr>
<tr>
<td>Sentence</td>
<td>Sentence element.</td>
</tr>
<tr>
<td>Table</td>
<td>A table consisting of Row elements that in turn consist of Cell elements</td>
</tr>
<tr>
<td>Term</td>
<td>A term, often used in context of Entry</td>
</tr>
</tbody>
</table>

Continued on next page
Table 10 – continued from previous page

<table>
<thead>
<tr>
<th>TableHead</th>
<th>Encapsulated the header of a table, contains Cell elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>A full text.</td>
</tr>
<tr>
<td>Whitespace</td>
<td>Whitespace element, signals a vertical whitespace</td>
</tr>
<tr>
<td>Word</td>
<td>Word (aka token) element.</td>
</tr>
</tbody>
</table>

1.9.1 folia.main.Cell

class folia.main.Cell(doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement

A cell in a Row in a Table

Method Summary

<table>
<thead>
<tr>
<th><strong>init</strong></th>
<th>Initialize self.</th>
</tr>
</thead>
<tbody>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([Founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, . . .])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolvelinear(begin, end[,...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True.</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True.</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

---

1.9. Structure Annotation Types

85
Table 11 – continued from previous page

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

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...

ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Cell'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' | '
WREFABLE = False
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=False, 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, None, or False) – 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=False, returnelements=False)
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)
- returnelements – Return the actual matches within the alternatives, will return two-
tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (*+) – The possible classes (AbstractElement or subclasses) to se-
lect from. Not instances!

Example:

paragraph = word.ancestor(folia.Paragraph)

ancestors (Class=func)
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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)
annotation (type, set=False)

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 False (default), all elements regardless of set will be returned.

Returns An element (instance derived from AbstractElement)

Example:

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

See also:

- AllowInlineAnnotation.annotations()
- AbstractElement.select()

Raises NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)

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 False (default), all elements regardless of set will be returned.

Yields Elements (instances derived from AbstractElement)

Example:

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

See also:

- AbstractElement.select()

Raises

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

```python
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
class method `findreplaceables` *(parent, set=False, **kwargs)*

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

`generate_id` *(cls)*

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

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

`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=False)*

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

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

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

Does the specified annotation layer exist?

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

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.

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

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:

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

Returns dict

layers(annotationtype=None, set=False)
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

parsecommonarguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

classmethod parsexml(node, doc, **kwargs)
Internal class method used for turning an XML element into an instance of the Class.

1.9. Structure Annotation Types
Parameters

- **node** — XML Element (* *)
- **doc** — Document (*)

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

```python
phon(cls='current', previousdelimiter=", strict=False, correctionhandling=1, hidden=False)
```

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 outputed, 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.
- **hidden**(bool) – Include hidden elements, defaults to False.

Example:

```python
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()`

```python
phoncontent(cls='current', correctionhandling=1, hidden=False)
```

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 phonetic 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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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
- **to** –

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

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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 fol-
  lowing elements: Alternative, AlternativeLayers, 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,
```

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to *False*.

**Example:**

```python
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, hidden=False)*

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.

- **hidden (bool)** – Include hidden elements, defaults to **False**.

**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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

1.9.2 folia.main.Definition

class folia.main.Definition(doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement

Element used in Entry for the portion that provides a definition for the entry.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>()</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Test whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Append a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>ancestors((Class))</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Append an ID suffix to this element and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>copychildren</strong></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>**correct(<strong>kwargs</strong>))</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count(Class[, set, recursive, ignore, node])</strong></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation()</strong></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch(function)</strong></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><strong>description()</strong></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat(subset)</strong></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling(cls)</strong></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><strong>findreplaceables(parent[, set])</strong></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id(cls)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getindex(child[, recursive, ignore])</strong></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata([key])</strong></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter([retaintokenisation])</strong></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation(Class[, set])</strong></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasannotationlayer([annotationtype, set])</strong></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><strong>hasphon([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection()</strong></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>**insert(index, child, *args, <strong>kwargs)</strong></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><strong>items([founditems])</strong></td>
<td></td>
</tr>
<tr>
<td><strong>json([attribs, recurse, ignorelist])</strong></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layers([annotationtype, set])</strong></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><strong>leftcontext(size[, placeholder, scope])</strong></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next([Class, scope, reverse])</strong></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext([cls])</strong></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>paragraphs([index])</strong></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td>**parsecommonarguments(doc, <strong>kwargs)</strong></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>**parsesxml(node, doc, <strong>kwargs)</strong></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
Table 12 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code>.</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code>.</td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attrs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 12 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring()</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

- `ACCEPTED_DATA` = (`<class 'folia.main.AbstractAnnotationLayer'>`, `<class 'folia.main.AbstractInlineAnnotation'>`, `<class 'folia.main.TextContent'>`, `<class 'folia.main.Utterance'>`, `<class 'folia.main.Whitespace'>`, `<class 'folia.main.Word'>`)
- `ANNOTATIONTYPE` = 42
- `AUTH` = True
- `AUTO_GENERATE_ID` = True
- `HIDDEN` = False
- `LABEL` = 'Definition'
- `OCCURRENCES` = 0
- `OCCURRENCES_PER_SET` = 0
- `OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
- `PHONCONTAINER` = False
- `PRIMARYELEMENT` = True
- `PRINTABLE` = True
- `REQUIRED_ATTRIBS` = None
- `REQUIRED_DATA` = None
- `SETONLY` = False
- `SPEAKABLE` = True
- `SUBSET` = None
- `TEXTCONTAINER` = False
- `TEXTDELIMITER` = '\n\n'
- `WREFABLE` = False
- `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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-
tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements
ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (*+) – The possible classes (AbstractElement or subclasses) to se-
lect 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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)
**annotation** (*type, set=False*)
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 False (default), all elements regardless of set will be returned.

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

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

See also:
`AllowInlineAnnotation.annotations()` `AbstractElement.select()`

Raises `NoSuchAnnotation` if no such annotation exists

**annotations** (*Class, set=False*)
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 False (default), all elements regardless of set will be returned.

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

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

See also:
`AbstractElement.select()`

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

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append** (*child, *args, **kwargs*)
See `AbstractElement.append()`
checkdeclaration()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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 appended 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=False, 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

depthfirstsearch (function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

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:

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

Returns str or list

findcorrectionhandling (cls)

Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.
classmethod findreplaceables (parent, set=False, **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=False)
   Returns an integer indicating whether such as annotation exists, and if so, how many.
   See AllowInlineAnnotation.annotations() for a description of the parameters.

hasannotationlayer (annotationtype=None, set=False)
   Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
   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, hidden=False)
   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 en-
     countered. 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:

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

Returns dict

layers(annotsontype=none, set=false)
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 el-
ment. 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

parseCommonArguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

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, hidden=False)`

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

- **hidden** (bool) – Include hidden elements, defaults to `False`.

Example:

```python
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
- `textcontent()`

`phoncontent(cls='current', correctionhandling=1, hidden=False)`

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 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.
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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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`, 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 el-
  ement. 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)

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.(to)** –

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

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'c9netto', True, [folia.Original,
```

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()

setprocessor (processor)
    Sets the processor for this element, taking care of adding an annotator in the declarations.

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, hidden=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to *False*.

**Example:**

```python
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, *hidden*=False)

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.

- **hidden** *(bool)* – Include hidden elements, defaults to **False**.

**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** (*attribs=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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for `text()`

1.9.3 folia.main.Division

class folia.main.Division(doc, *args, **kwargs)
Bases: folia.main.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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, . . .])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
</tbody>
</table>

Continued on next page

1.9. Structure Annotation Types

111
Table 13 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannnotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>head()</code></td>
<td></td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>parsexml</strong>&lt;br&gt;(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong>&lt;br&gt;(cls, previousdelimiter, strict, ...)</td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>phoncontent</strong>&lt;br&gt;(cls, correctionhandling, hidden)</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend</strong>()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>previous</strong>&lt;br&gt;(Class, scope)</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng</strong>&lt;br&gt;(includechildren, extraattrs, ...)</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><strong>relaxng_backwards</strong>()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><strong>remove</strong>&lt;br&gt;(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td><strong>replace</strong>&lt;br&gt;(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><strong>resolveoffsets</strong>&lt;br&gt;(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><strong>resolveword</strong>&lt;br&gt;(id)</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><strong>select</strong>&lt;br&gt;(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><strong>sentences</strong>&lt;br&gt;([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><strong>setdoc</strong>&lt;br&gt;(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><strong>setdocument</strong>&lt;br&gt;(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><strong>setparents</strong>()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><strong>setprocessor</strong>&lt;br&gt;(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><strong>settext</strong>&lt;br&gt;(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><strong>speech_speaker</strong>()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>speech_src</strong>()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>stricttext</strong>&lt;br&gt;(cls)</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td><strong>text</strong>&lt;br&gt;(cls, retaintokenisation, ...)</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>textcontent</strong>&lt;br&gt;(cls, correctionhandling, hidden)</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textvalidation</strong>&lt;br&gt;(warnonly)</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><strong>toktext</strong>&lt;br&gt;(cls)</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td><strong>updatetext</strong>()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><strong>words</strong>&lt;br&gt;([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
</tbody>
</table>
**Table 13 – continued from previous page**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

- `ACCEPTED_DATA` = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
- `ANNOTATIONTYPE` = 2
- `AUTH` = True
- `AUTO_GENERATE_ID` = True
- `HIDDEN` = False
- `LABEL` = 'Division'
- `OCCURRENCES` = 0
- `OCCURRENCES_PER_SET` = 0
- `OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 5, 8, 6, 7, 9, 11, 13)
- `PHONCONTAINER` = False
- `PRIMARYELEMENT` = True
- `PRINTABLE` = True
- `REQUIRED_ATTRIBS` = None
- `REQUIRED_DATA` = None
- `SETONLY` = False
- `SPEAKABLE` = True
- `SUBSET` = None
- `TEXTCONTAINER` = False
- `TEXTDELIMITER` = '\n


'
- `WREFABLE` = False
- `XLINK` = False
- `XMLTAG` = 'div'

**Method Details**

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

```python
__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=False, 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, None, or False)* – 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=False, returnelements=False)

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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

**Yields** Alternative elements

**ancestor**(Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

**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 (tuple of) classes (AbstractElement or subclasses). Not instances!*
Yields elements (instances derived from AbstractElement)

**annotation** (*type, set=False*)

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 False (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:**

AllowInlineAnnotation.annotations() AbstractElement.select()

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** (*Class, set=False*)

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 False (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**

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int

depervalidation()
Perform deep validation of this element.

Raises DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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=False, **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

gettxtdelimiter(retaintokenisation=False)

Return the text delimiter for this class.

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

hasannotation(Class, set=False)

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

See AllowInlineAnnotation.annotations() for a description of the parameters.

hasannotationlayer(annocationtype=None, set=False)

Does the specified annotation layer exist?

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

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

hastxt(cls='current', strict=True, correctionhandling=1, hidden=False)

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.
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(attribs=None, recurse=True, ignorelist=False)
Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.
Example:

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

Returns dict

layers(annotType=None, set=False)
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

parseCommonArguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.
**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, hidden=False)**

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 outputed, 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.
- `hidden` (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)**

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.

**precedes**(other)
Returns a boolean indicating whether this element precedes the other element

**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’, 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.

**classmethod relaxng**(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**classmethod relaxng_backwards()**
internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

1.9. Structure Annotation Types 121
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=False, 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 False (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 follow-
  ing elements: Alternative, AlternativeLayers, 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:

```
...
```

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 sually no need to call this directly,
invoked implicitly by copy()

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, useful when chaining calls to `text()`.

- **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

- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```
word.text()
```

**Returns**  
The text of the element *(unicode instance in Python 2, str in Python 3)*
textcontent (cls='current', correctionhandling=1, hidden=False)
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.
- **hidden**(bool) – Include hidden elements, defaults to False.

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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for `text()`

1.9.4 folia.main.Entry

class folia.main.Entry(doc, *args, **kwargs)

Bases: folia.main.AbstractStructureElement

Represents an entry in a glossary/lexicon/dictionary.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>(doc, *args, **kwargs) Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>(idsuffix[, recursive])</td>
</tr>
<tr>
<td>addtoindex</td>
<td>([norecurse])</td>
</tr>
<tr>
<td>alternatives</td>
<td>([Class, set, returnelements])</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes)</td>
</tr>
<tr>
<td>ancestors</td>
<td>([Class])</td>
</tr>
<tr>
<td>annotation</td>
<td>(type[, set])</td>
</tr>
<tr>
<td>annotations</td>
<td>(Class[, set])</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>([annotator, ...])</td>
</tr>
<tr>
<td>append</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td>()</td>
</tr>
</tbody>
</table>

Continued on next page
Table 14 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>context</strong>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><strong>copy</strong>(newdoc, idsuffix)</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren</strong>(newdoc, idsuffix)</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>correct</strong>(kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count</strong>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><strong>description</strong>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong>(cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><strong>findreplaceables</strong>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>getindex</strong>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong>(key)</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong>(retaintokenisation)</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong>(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasannotationlayer</strong>(annotationtype, set)</td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><strong>hasphon</strong>(cls, strict, correctionhandling, . . .)</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext</strong>(cls, strict, correctionhandling, . . .)</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection</strong>()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><strong>insert</strong>(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><strong>items</strong>(founditems)</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layers</strong>(annotationtype, set)</td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><strong>leftcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong>(Class, scope, reverse)</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong>(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>paragraphs</strong>(index)</td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong>(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
</tbody>
</table>
Table 14 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 14 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.Alternative'>, ...
ANNOTATIONTYPE = 40
AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Entry'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
WREFABLE = False
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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements
ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
   Not instances!
Yields elements (instances derived from `AbstractElement`)

`annotation(type, set=False)`
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 False (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:
- `AllowInlineAnnotation.annotations()` `AbstractElement.select()`

Raises `NoSuchAnnotation` if no such annotation exists

`annotations(Class, set=False)`
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 False (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
- `NoSuchAnnotation` if no such annotation exists

`annotator2processor(annotator=None, annotatortype=None, parentprocessor=None)`
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

`append(child, *args, **kwargs)`
See `AbstractElement.append()`
checkdeclaration()
   Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch (function)
   Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

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

   Returns str or list

findcorrectionhandling (cls)
   Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused
classmethod `findreplaceables`

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

`generate_id(cls)`

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

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

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=False)`

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

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

`hasannotationlayer(annnotationtype=None, set=False)`

Does the specified annotation layer exist?

`hasphon(cls='current', strict=True, correctionhandling=1, hidden=False)`

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, hidden=False)`

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 content prior to correction, and `CorrectionHandling.EITHER` if you don’t care.
want the text prior to correction, and \texttt{CorrectionHandling.EITHER} if you don’t care.

\textbf{Returns} bool

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

\texttt{insert(index, child, *args, **kwargs)}

\texttt{items(founditems=[])}
Returns a depth-first flat list of all items below this element (not limited to AbstractElement)

\texttt{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:

\begin{python}
import json
json.dumps(word.json())
\end{python}

\textbf{Returns} dict

\texttt{layers(annnotationtype=\texttt{None}, set=False)}
Returns a list of annotation layers found directly under this element, does not include alternative layers

\texttt{leftcontext(size, placeholder=\texttt{None}, scope=\texttt{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

\texttt{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.

\textbf{Parameters}

\begin{itemize}
\item \texttt{Class} (\texttt{\ast}) – The class to select; any python class subclassed off \texttt{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 \texttt{None} to not constrain at all
\item \texttt{scope} (\texttt{\ast}) – 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 \texttt{None} to not constrain at all
\end{itemize}

\texttt{originaltext(cls=\texttt{\textquote{\text{'original'}}})}
Alias for retrieving the original uncorrect text.

A call to \texttt{text()} with \texttt{correctionhandling=CorrectionHandling.ORIGINAL}

\texttt{paragraphs(index=None)}
Returns a generator of Paragraph elements found (recursively) under this element.

\textbf{Parameters} \texttt{index} (\texttt{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

\texttt{parsecommonarguments(doc, **kwargs)}
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

\texttt{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, hidden=False)

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 outputed, 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.
- hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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

phoncontent (cls='current', correctionhandling=1, hidden=False)

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**resolveword**(id)
rightcontext \( \text{(size, placeholder=\text{None}, scope=\text{None})} \)

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 \( \text{(Class, set=\text{False}, recursive=\text{True}, ignore=\text{True}, node=\text{None})} \)

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 False (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`, `AlternativeLayers`, `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:

```python
...
```

sentences \( \text{(index=\text{None})} \)

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 \( \text{(newdoc)} \)

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

setdocument \( \text{(doc)} \)

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()`

setprocessor \( \text{(processor)} \)

Sets the processor for this element, taking care of adding an annotator in the declarations

settext \( \text{(text, cls='current')} \)

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, **hidden**=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)
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.

- **hidden** (bool) – Include hidden elements, defaults to `False`.

**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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

1.9.5 folia.main.Event

class folia.main.Event (doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement

Structural element representing events, often used in new media contexts for things such as tweets, chat messages and forum posts.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accept (Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>_addable (parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex (norecurse)</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives ([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors ([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation (type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations (Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor ([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append (child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context (size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to <code>AbstractElement</code>)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 15 – continued from previous page

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>tokttext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 15 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__() </code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

- ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
- ANNOTATIONTYPE = 22
- AUTH = True
- AUTO_GENERATE_ID = True
- HIDDEN = False
- LABEL = 'Event'
- OCCURRENCES = 0
- OCCURRENCES_PER_SET = 0
- OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
- PHONCONTAINER = False
- PRIMARYELEMENT = True
- PRINTABLE = True
- REQUIRED_ATTRIBS = None
- REQUIRED_DATA = None
- SETONLY = False
- SPEAKABLE = True
- SUBSET = None
- TEXTCONTAINER = False
- TEXTDELIMITER = '\n\n'
- WREFABLE = False
- 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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

1.9. Structure Annotation Types 143
**annotation** *(type, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

**See also:**

- `AllowInlineAnnotation.annotations()`  
- `AbstractElement.select()`

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

**See also:**

- `AbstractElement.select()`

**Raises**

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

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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 appened 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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **kwargs)
Internal method to find replaceable elements. Auxiliary function used by AbstractElement.replace(). Can be overriden 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=False)
Returns an integer indicating whether such as annotation exists, and if so, how many.
See AllowInlineAnnotation.annotations() for a description of the parameters.

hasannotationlayer (annotationtype=None, set=False)
Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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:

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

Returns dict

layers(annotationToken=None, set=False)

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

parseCommonArguments(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)`

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden (bool)** – Include hidden elements, defaults to `False`.

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

`textContent ()`

`phoncontent (cls='current', correctionhandling=1, hidden=False)`

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 phonetic 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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 to be an alternative.

- **be an alternative.** (to)

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

**resolveword**(id)

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!
rightcontext(\texttt{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(\texttt{Class, set=False, recursive=True, ignore=True, node=None})
- Select child elements of the specified class.
  A further restriction can be made based on set.

  \textbf{Parameters}
  
  - \texttt{Class (class)} – The class to select; any python class (not instance) subclassed off \texttt{AbstractElement}.
  - \texttt{Set (str)} – The set to match against, only elements pertaining to this set will be returned. If set to False (default), all elements regardless of set will be returned.
  - \texttt{recursive (bool)} – Select recursively? Descending into child elements? Defaults to True.
  - \texttt{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: \texttt{Alternative}, \texttt{AlternativeLayers}, \texttt{Suggestion}, and \texttt{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.
  - \texttt{node (+)} – Reserved for internal usage, used in recursion.

  \textbf{Yields} Elements (instances derived from \texttt{AbstractElement})

  \textbf{Example:}

  ```python
  for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original, __
  \texttt{\textbullet folia.Suggestion, folia.Alternative}]):
  ...
  ```

sentences(\texttt{index=None})
- Returns a generator of Sentence elements found (recursively) under this element.

  \textbf{Parameters} \texttt{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(\texttt{newdoc})
- Set a different document. Usually no need to call this directly, invoked implicitly by \texttt{copy()}

setdocument(\texttt{doc})
- Associate a document with this element.

  \textbf{Parameters} \texttt{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 \texttt{copy()}

setprocessor(\texttt{processor})
- Sets the processor for this element, taking care of adding an annotator in the declarations.

settext(\texttt{text, cls=’current’})
- Set the text for this element.

  \textbf{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, hidden=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 outputed, 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

• **hidden (bool)** – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)*

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.

- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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:

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

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

1.9.6 folia.main.Example

class folia.main.Example (doc, *args, **kwargs)

    Bases: folia.main.AbstractStructureElement

Element that provides an example. Used for instance in the context of Entry

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts (Class, raiseexceptions, parentinstance)</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex [norecurse]</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives (Class, set, returnelements)</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors ([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation (type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations (Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor ([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append (child, *args, **kwargs)</td>
<td>See AbstractElement.append ()</td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context (size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
<tr>
<td>copy ([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
</tbody>
</table>

Continued on next page

1.9. Structure Annotation Types
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren(newdoc, idsuffix)</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hasText([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsetext(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace</td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword</td>
<td>Retrieve the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>rightcontext</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences</td>
<td>Returns a generator of Sentence elements found recursively under this element.</td>
</tr>
<tr>
<td>setdoc</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speechSpeaker</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext</td>
<td>Alias for <code>text()</code> with <code>strict=True</code>.</td>
</tr>
<tr>
<td>text</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>tokenize</td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code>.</td>
</tr>
<tr>
<td>updatetext</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words</td>
<td>Returns a generator of Word elements found recursively under this element.</td>
</tr>
<tr>
<td>xml</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 16 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring()</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

- `ACCEPTED_DATA` = (`<class 'folia.main.AbstractAnnotationLayer'>`, `<class 'folia.main.AbstractInlineAnnotation'>`, `<class 'folia.main.TextContent'>`, `<class 'folia.main.Utterance'>`, `<class 'folia.main.Whitespace'>`, `<class 'folia.main.Word'>`)
- `ANNOTATIONTYPE` = 43
- `AUTH` = True
- `AUTO_GENERATE_ID` = True
- `HIDDEN` = False
- `LABEL` = 'Example'
- `OCCURRENCES` = 0
- `OCCURRENCES_PER_SET` = 0
- `OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
- `PHONCONTAINER` = False
- `PRIMARYELEMENT` = True
- `PRINTABLE` = True
- `REQUIRED_ATTRIBS` = None
- `REQUIRED_DATA` = None
- `SETONLY` = False
- `SPEAKABLE` = True
- `SUBSET` = None
- `TEXTCONTAINER` = False
- `TEXTDELIMITER` = '\n\n'
- `WREFABLE` = False
- `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=False, 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 overriden by subclasses for more customised behaviour.

Parameters

• parent (AbstractElement) – The element that is being added to
• set (str, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)
**annotation**(type, set=False)

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 False (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:**

AllowInlineAnnotation.annotations() AbstractElement.select()

**Raises** NoSuchAnnotation if no such annotation exists

**annotations**(Class, set=False)

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 False (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**

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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
depthfirstsearch(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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=False, **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=False)
Returns an integer indicating whether such an annotation exists, and if so, how many.
See AllowInlineAnnotation. annotations() for a description of the parameters.

hasannotationlayer (annotationtype=None, set=False)
Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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:

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

Returns dict

layers (annotationtype=None, set=False)
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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
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 outputed, 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.
• hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)
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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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 ele-
ment. 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)
classmethod relaxng_backwards()
internal helper function for backward compatibility
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 alter-
native. Simply use AbstractElement.append() if you want the added element
• be an alternative. (to) –
See AbstractElement.append() for more information and all parameters.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

resolveword(id)
rightcontext\( (size, \text{placeholder}=None, \text{scope}=None) \)

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

select\( (\text{Class}, \text{set}=\text{False}, \text{recursive}=\text{True}, \text{ignore}=\text{True}, \text{node}=\text{None}) \)

Select child elements of the specified class.

A further restriction can be made based on \( \text{set} \).

**Parameters**

- **Class**\( (\text{class}) \) – The class to select; any python class (not instance) subclassed off \texttt{AbstractElement}.
- **Set**\( (\text{str}) \) – The set to match against, only elements pertaining to this set will be returned. If set to \text{False} (default), all elements regardless of set will be returned.
- **recursive**\( (\text{bool}) \) – Select recursively? Descending into child elements? Defaults to \text{True}.
- **ignore** – A list of Classes to ignore, if set to \text{True} instead of a list, all non-authoritative elements will be skipped (this is the default behaviour and corresponds to the following elements: \texttt{Alternative}, \texttt{AlternativeLayers}, \texttt{Suggestion}, and \texttt{folia.Original}. These elements and those contained within are never \textit{authoritative}. You may also include the boolean \text{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 \texttt{AbstractElement})

**Example:**

```python
for sense in text.select(folia.Sense, 'cornoett\textbackslash o', True, [folia.Original, ...
\texttt{\rightarrow folia.Suggestion, folia.Alternative} ]):
    ...
```

sentences\( (\text{index}=\text{None}) \)

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

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

setdoc\( (\text{newdoc}) \)

Set a different document. Usually no need to call this directly, invoked implicitly by \texttt{copy()}

setdocument\( (\text{doc}) \)

Associate a document with this element.

**Parameters** **doc**\( (\text{Document}) \) – 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 \texttt{copy()}

setprocessor\( (\text{processor}) \)

Sets the processor for this element, taking care of adding an annotator in the declarations.

settext\( (\text{text, cls='current'}) \)

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to *False*.

**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.

---

1.9. Structure Annotation Types 165
textcontent (cls='current', correctionhandling=1, hidden=False)
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.

- **hidden (bool)** – Include hidden elements, defaults to False.

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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

1.9.7 folia.main.Figure

class folia.main.Figure(doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement

Element for the representation of a graphical figure. Structure element.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class, raiseexceptions, parentinstance)</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex[norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator,...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>caption()</td>
<td></td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 17 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select([Class[, set, recursive, ignore, node]])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for <strong>text()</strong> with <strong>strict=True</strong></td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>tokei([cls])</td>
<td>Alias for <strong>text()</strong> with <strong>retaintokenisation=True</strong></td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

**Continued on next page**

### 1.9. Structure Annotation Types
Table 17 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.Alternative'>, ...

ANNOTATIONTYPE = 6

AUTH = True

AUTO_GENERATE_ID = True

HIDDEN = False

LABEL = 'Figure'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 0

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)

PHONCONTAINER = False

PRIMARYELEMENT = True

PRINTABLE = True

REQUIRED_ATTRIBS = None

REQUIRED_DATA = None

SETONLY = False

SPEAKABLE = False

SUBSET = None

TEXTCONTAINER = False

TEXTDELIMITER = '\n\n'

WREFABLE = False

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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return twotuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

Parameters Classes (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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

`AllowInlineAnnotation.annotations()` `AbstractElement.select()`

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

`AbstractElement.select()`

**Raises**

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

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`

**caption** ()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns  int

dep deepvalidation()
Perform deep validation of this element.

Raises  DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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

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

hasannotationlayer (annotationtype=None, set=False)
   Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
   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 correc-
     tion, and CorrectionHandling.EITHER if you don’t care.
   Returns bool

hastext (cls='current', strict=True, correctionhandling=1, hidden=False)
   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 en-
     countered. 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:

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

Returns `dict`

`layers(annnotationtype=None, set=False)`
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

`parseCommonArguments(doc, **kwargs)`
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

classmethod `parseXML(node, doc, **kwargs)`
Internal class method used for turning an XML element into an instance of the Class.

1.9. Structure Annotation Types 175
Parameters

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

Returns An instance of the current Class.

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

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 outputed, 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.
- hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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`, 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 el-
  ement. 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)

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative**.(to) –

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

**resolveoffsets**(begin, end, retaintokenisation=**True**, strictend=**True**, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:

```python
```

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()`.

setprocessor (processor)

Sets the processor for this element, taking care of adding an annotator in the declarations.

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.

**speechSpeaker()**
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

**speechSrc()**
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, normalizeSpaces=False, hidden=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 outputed, useful when chaining calls to *text()*.

• **strict** *(bool)* – Set this iif you are strictly interested in the text explicitly associated with the element, without recursing into children.

• **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.

• **normalizeSpaces** *(bool)* – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)

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.
- **hidden (bool)** – Include hidden elements, defaults to False.

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 (attribs=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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for text()

1.9.8 folia.main.Gap

class folia.main.Gap(doc, *args, **kwargs)
    Bases: folia.main.AbstractElement

Gap element, represents skipped portions of the text.

Usually contains Content and possibly also a Description element

Method Summary

__init__(doc, *args, **kwargs) Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.
addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.
addtoindex([norecurse]) 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.
ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
annotator2processor([annotator, ...]) Converts annotator information to processor information (FoLiA v2).
append(child, *args, **kwargs)
checkdeclaration() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.
content()
context(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right
copy([newdoc, idsuffix]) Make a deep copy of this element and all its children.
copychildren([newdoc, idsuffix]) Generator creating a deep copy of the children of this element.

Continued on next page
### Table 18 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>count</strong>(Class[, set, recursive, ignore, node])</td>
<td>Like <code>AbstractElement.select()</code> , but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><strong>description</strong>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><strong>findreplaceables</strong>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>getindex</strong>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong>([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong>(retaintokenisation)</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasphon</strong>(cls, strict, correctionhandling, ...)</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext</strong>(cls, strict, correctionhandling, ...)</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection</strong>()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><strong>insert</strong>(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td><strong>items</strong>(founditems)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><strong>json</strong>(attribs, recurse, ignorelist)</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>leftcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong>(Class, scope, reverse)</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong>(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong>(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><strong>parsexml</strong>(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong>(cls, previousdelimiter, strict, ...)</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>phoncontent</strong>(cls, correctionhandling, hidden)</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend</strong>()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>precedes</strong>(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><strong>previous</strong>(Class, scope)</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng</strong>(includechildren, extraattrs, ...)</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
</tbody>
</table>

Continued on next page
Table 18 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, . . .])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, . . .])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetexttext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Content'>, <class 'folia.main.Description'>, <class 'folia.main.Feature'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Metric'>, <class 'folia.main.Part'>)

ANNOTATIONTYPE = 25

AUTH = True

AUTO_GENERATE_ID = False

HIDDEN = False

LABEL = '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=False, 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, None, or False) – 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. Raise a No-
   SuchAnnotation exception if not found.

   Parameters Classes (*) – The possible classes (AbstractElement or subclasses) to se-
   lect from. Not instances!

   Example:
   ```python
   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 (tuple of) classes (AbstractElement or subclasses).
   Not instances!

   Yields elements (instances derived from AbstractElement)

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
   Converts annotator information to processor information (FoLiA v2). Can be called with arguments to
   override defaults.

append (child, *args, **kwargs)

checkdeclaration ()
   Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
   declared, raises an exception if not so, or auto-declares the annotation type if need be.

ccontent ()

ccontext (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=False, recursive=True, ignore=True, node=None)
   Like AbstractElement.select (), but instead of returning the elements, it merely counts them.

   Returns int

depdeepvalidation ()
   Perform deep validation of this element.
Raises DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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:

```python
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=False, **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, hidden=False)
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 correc-
tion, and CorrectionHandling.EITHER if you don’t care.
Returns bool

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

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 en-
countered. 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:

```python
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 el-
ement. Set to True to constrain to a default list of structure elements (Sen-
tence,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`

**parsecommonarguments** *(doc, **kwargs)*
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)*
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 outputed, 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.
- `hidden` *(bool)* — Include hidden elements, defaults to `False`.

**Example:**

```python
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, hidden=False)*
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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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.

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

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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()

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations
**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, hidden=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 outputed, 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
- **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:
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, hidden=False)
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.
  * hidden (bool) – Include hidden elements, defaults to False.

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)
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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for text()

1.9.9 folia.main.Head

class folia.main.Head(doc, *args, **kwargs)
    Bases: folia.main.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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 19 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right.</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a text class by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found recursively under this element.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found recursively under this element.</td>
</tr>
</tbody>
</table>
Table 19 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (
    <class 'folia.main.AbstractAnnotationLayer'>, 
    <class 'folia.main.AbstractInlineAnnotation'>, 
    <class 'folia.main.AbstractLayer'>, 
    <class 'folia.main.Annotation'>, 
    <class 'folia.main.AnnotationSet'>, 
    <class 'folia.main.AnnotationType'>, 
    <class 'folia.main.Audio'>, 
    <class 'folia.main.AudioFile'>, 
    <class 'folia.main.BoundingBox'>, 
    <class 'folia.main.Character'>, 
    <class 'folia.main.CharacterSet'>, 
    <class 'folia.main.CharacterSequence'>, 
    <class 'folia.main.Column'>, 
    <class 'folia.main.ColumnSet'>, 
    <class 'folia.main.Element'>, 
    <class 'folia.main.ElementSet'>, 
    <class 'folia.main.Grammar'>, 
    <class 'folia.main.GrammarSet'>, 
    <class 'folia.main.GrammarUnit'>, 
    <class 'folia.main.IdentifiedObject'>, 
    <class 'folia.main.Line'>, 
    <class 'folia.main.LineSet'>, 
    <class 'folia.main.Paragraph'>, 
    <class 'folia.main.ParagraphSet'>, 
    <class 'folia.main.Phrase'>, 
    <class 'folia.main.PhraseSet'>, 
    <class 'folia.main.PhonologicalAnnotation'>, 
    <class 'folia.main.PhraseAnnotation'>, 
    <class 'folia.main.PhonologicalAnnotationSet'>, 
    <class 'folia.main.PhraseAnnotationSet'>, 
    <class 'folia.main.PhraseType'>, 
    <class 'folia.main.PhonologicalType'>, 
    <class 'folia.main.PhraseTypeSet'>, 
    <class 'folia.main.PhonologicalTypeSet'>, 
    <class 'folia.main.PhraseTypeUnit'>, 
    <class 'folia.main.PhraseTypeUnitSet'>, 
    <class 'folia.main.PhraseUnit'>, 
    <class 'folia.main.PhraseUnitSet'>, 
    <class 'folia.main.Position'>, 
    <class 'folia.main.PositionSet'>, 
    <class 'folia.main.Sentence'>, 
    <class 'folia.main.SentenceSet'>, 
    <class 'folia.main.SentenceType'>, 
    <class 'folia.main.SentenceTypeSet'>, 
    <class 'folia.main.SentenceUnit'>, 
    <class 'folia.main.SentenceUnitSet'>, 
    <class 'folia.main.Text'>, 
    <class 'folia.main.TextAnnotation'>, 
    <class 'folia.main.TextAnnotationSet'>, 
    <class 'folia.main.TextContent'>, 
    <class 'folia.main.TextContentSet'>, 
    <class 'folia.main.TextString'>, 
    <class 'folia.main.TextStringSet'>, 
    <class 'folia.main.Token'>, 
    <class 'folia.main.TokenSet'>, 
    <class 'folia.main.TextUnit'>, 
    <class 'folia.main.Word'>, 
    <class 'folia.main.WordSet'>, 
    <class 'folia.main.WordType'>, 
    <class 'folia.main.WordTypeSet'>, 
    <class 'folia.main.WordUnit'>, 
    <class 'folia.main.WordUnitSet'>
)

ANNOTATIONTYPE = 4

AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Head'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
WREFABLE = False
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=False, 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, None, or False) – 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 sually 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=False, returnelements=False)
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)

• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).

Not instances!
Yields  elements (instances derived from AbstractElement)

annotation(type, set=False)

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 False (default), all elements regardless of set will be returned.

Returns  An element (instance derived from AbstractElement)

Example:

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

See also:

AllowInlineAnnotation.annotations() AbstractElement.select()

Raises  NoSuchAnnotation if no such annotation exists

annotations(Class, set=False)

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 False (default), all elements regardless of set will be returned.

Yields  Elements (instances derived from AbstractElement)

Example:

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

See also:

AbstractElement.select()

Raises

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

annotator2processor(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append(child, *args, **kwargs)

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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

hasannotationlayer (annotationtype=None, set=False)
Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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:

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

Returns dict

layers(annootationtype=None, set=False)
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

parseCommonArguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)

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.

### precedes(other)

Returns a boolean indicating whether this element precedes the other element

### 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
derived 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 el-
  ement. 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)

### classmethod relaxng_backwards()

internal helper function for backward compatibility

### 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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.(to)** –

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

### resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

### 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=False, 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 False (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, AlternativeLayers, 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:
```python
```

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)*

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.
- **hidden** *(bool)* – Include hidden elements, defaults to *False*.

**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** *(attribs=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*
def __iter__(self):
    Iterate over all children of this element.

    Example:
    
    ```python
    for annotation in word:
        ...
    ```

def __len__(self):
    Returns the number of child elements under the current element.

def __str__(self):
    Alias for `text()`

1.9.10 folia.main.Linebreak

class folia.main.Linebreak(doc, *args, **kwargs):

    Line break element, signals a line break.

    This element acts both as a structure element as well as a text markup element.

    **Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>See <code>AbstractElement.__init__()</code>, text is passed as a string in *args.</td>
</tr>
<tr>
<td>accepts(Class, raiseexceptions, parentinstance)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a text class by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotatiolayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])**</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling,...])**</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>See AbstractElement.json()</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>parsexml(node, doc)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolve()</code></td>
<td></td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text)</code></td>
<td>Sets the text content of the markup element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attrs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTED_DATA</td>
<td>(&lt;class 'folia.main.AbstractAnnotationLayer'&gt;, &lt;class 'folia.main.Alternative'&gt;, ...)</td>
</tr>
<tr>
<td>ANNOTATIONTYPE</td>
<td>8</td>
</tr>
<tr>
<td>AUTH</td>
<td>True</td>
</tr>
<tr>
<td>AUTO_GENERATE_ID</td>
<td>True</td>
</tr>
<tr>
<td>HIDDEN</td>
<td>False</td>
</tr>
<tr>
<td>LABEL</td>
<td>'Linebreak'</td>
</tr>
<tr>
<td>OCCURRENCES</td>
<td>0</td>
</tr>
<tr>
<td>OCCURRENCES_PER_SET</td>
<td>0</td>
</tr>
<tr>
<td>OPTIONAL_ATTRIBS</td>
<td>(0, 1, 2, 4, 5, 8, 6, 7, 9, 11, 13)</td>
</tr>
<tr>
<td>PHONCONTAINER</td>
<td>False</td>
</tr>
<tr>
<td>PRIMARYELEMENT</td>
<td>False</td>
</tr>
<tr>
<td>PRINTABLE</td>
<td>True</td>
</tr>
<tr>
<td>REQUIRED_ATTRIBS</td>
<td>None</td>
</tr>
<tr>
<td>REQUIRED_DATA</td>
<td>None</td>
</tr>
<tr>
<td>SETONLY</td>
<td>False</td>
</tr>
<tr>
<td>SPEAKABLE</td>
<td>True</td>
</tr>
<tr>
<td>SUBSET</td>
<td>None</td>
</tr>
<tr>
<td>TEXTCONTAINER</td>
<td>True</td>
</tr>
<tr>
<td>TEXTDEIMITER</td>
<td>''</td>
</tr>
<tr>
<td>WREFABLE</td>
<td>False</td>
</tr>
<tr>
<td>XLINK</td>
<td>True</td>
</tr>
<tr>
<td>XMLTAG</td>
<td>'br'</td>
</tr>
</tbody>
</table>

### Method Details

#### `__init__` (doc, *args, **kwargs)

See `AbstractElement.__init__()`, text is passed as a string in `*args`. 

---

**Table 20 – continued from previous page**
__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=False, 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, None, or False) – 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=False, returnelements=False)
    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)
        • returnelements – Return the actual matches within the alternatives, will return two-
tuples where the first is an instance of Alternative and the second an instance of Class.
    Yields Alternative elements

ancestor(*Classes)
    Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-
SuchAnnotation exception if not found.
    Parameters Classes (*–) – The possible classes (AbstractElement or subclasses) to se-
    lect 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 (tuple of) classes (`AbstractElement` or subclasses). Not instances!

Yields elements (instances derived from `AbstractElement`)

Annotation (`type`, `set=False`)  
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 False (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:

`AbstractElement.select()`

Raises `NoSuchAnnotation` if no such annotation exists

Annotations (`Class`, `set=False`)  
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 False (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

• `AllowInlineAnnotation.annotations()`  
• `NoSuchAnnotation` if no such annotation exists

Annotator2Processor (`annotator=None, annotatortype=None, parentprocessor=None`)  
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.
append(child, *args, **kwargs)
    See AbstractElement.append()

checkdeclaration()
    Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch(function)
    Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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 correction handling given a textclass by looking in the underlying corrections where it is reused

classmethod findreplaceables(parent, set=False, **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

gettmetadelimiter(retaintokenisation=False)
    Return the text delimiter for this class.

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

hasannotation(Class, set=False)
    Returns an integer indicating whether such an annotation exists, and if so, how many.

    See AllowInlineAnnotation.annotations() for a description of the parameters.

hasannotationlayer(annotationtype=None, set=False)
    Does the specified annotation layer exist?

hasphon(cls='current', strict=True, correctionhandling=1, hidden=False)
    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, hidden=False)
    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()`

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

**parsecommonarguments**(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)
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 outputed, 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.

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

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)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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()

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

resolvecword(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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'cornoeto', True, [folio.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()

setprocessor (processor)

Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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
- **hidden bool** – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)

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.

• **hidden** *(bool)*—Include hidden elements, defaults to `False`.

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** *(attribs=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()

1.9.11 folia.main.List

class folia.main.List(doc, *args, **kwargs)

Bases: folia.main.AbstractStructureElement


Method Summary

__init__(doc, *args, **kwargs) Initialize self.

accepts(Class[, raiseexceptions, parentinstance])

add(child, *args, **kwargs)

addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.

addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.

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

alternatives([Class, set, returnelements]) Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

ancestor(*Classes) Find the most immediate ancestor of the specified type, multiple classes may be specified.

ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

annotation(type[, set]) Obtain a single annotation element.

annotations(Class[, set]) Obtain child elements (annotations) of the specified class.

annotator2processor([annotator, …]) Converts annotator information to processor information (FoLiA v2).

append(child, *args, **kwargs) See AbstractElement.append()

checkdeclaration() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right

copy([newdoc, idsuffix]) Make a deep copy of this element and all its children.

copychildren([newdoc, idsuffix]) Generator creating a deep copy of the children of this element.

correct(**kwargs) Apply a correction (TODO: documentation to be written still)
Table 21 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>count</strong> (Class[, set, recursive, ignore, node])</td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong> (function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description</strong>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong> (subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong> (cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables</strong> (parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id</strong> (cls)</td>
<td></td>
</tr>
<tr>
<td><strong>getindex</strong> (child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong> ([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong> ([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong> (Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasannotationlayer</strong> ([annotationtype, set])</td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><strong>hasphon</strong> ([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext</strong> ([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection</strong>()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><strong>insert</strong> (index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><strong>items</strong> ([founditems])</td>
<td></td>
</tr>
<tr>
<td><strong>json</strong> ([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layers</strong> ([annotationtype, set])</td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><strong>leftcontext</strong> (size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong> ([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong> ([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>paragraphs</strong> ([index])</td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong> (doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><strong>parserxml</strong> (node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong> ([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>phoncontent</strong> ([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 21 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>postappend()</strong></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>precedes(other)</strong></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><strong>previous([Class, scope])</strong></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng([includechildren, extraattrs, ...])</strong></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><strong>relaxng_backwards()</strong></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><strong>remove(child)</strong></td>
<td>Removes the child element</td>
</tr>
<tr>
<td>**replace(child, *args, <strong>kwargs)</strong></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><strong>resolveoffsets(begin, end[, ...])</strong></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><strong>resolveword(id)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>rightcontext(size[, placeholder, scope])</strong></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><strong>select(Class[, set, recursive, ignore, node])</strong></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><strong>sentences([index])</strong></td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td><strong>setdoc(newdoc)</strong></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><strong>setdocument(doc)</strong></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><strong>setparents()</strong></td>
<td>Correct all parent relations for elements within the scope.</td>
</tr>
<tr>
<td><strong>setprocessor(processor)</strong></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><strong>settext(text[, cls])</strong></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><strong>speech_speaker()</strong></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>speech_src()</strong></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>stricttext([cls])</strong></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><strong>text([cls, retaintokenisation, ...])</strong></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textcontent([cls, correctionhandling, hidden])</strong></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textvalidation([warnonly])</strong></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><strong>toktext([cls])</strong></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><strong>updatetext()</strong></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><strong>words([index])</strong></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><strong>xml([attrs, elements, skipchildren])</strong></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>xmlstring([pretty_print])</strong></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong><strong>iter</strong>()</strong></td>
<td>Iterate over all children of this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 21 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes


ANNOTATIONTYPE = 5

AUTH = True

AUTO_GENERATE_ID = True

HIDDEN = False

LABEL = 'List'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 0

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 5, 8, 6, 7, 9, 11, 13)

PHONCONTAINER = False

PRIMARYELEMENT = True

PRINTABLE = True

REQUIRED_ATTRIBS = None

REQUIRED_DATA = None

SETONLY = False

TEXTCONTAINER = False

TEXTDELIMITER = '

'

WREFABLE = False

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)

doc(child, *args, **kwargs)
classmethod addable (parent, set=False, 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 overriden
  by subclasses for more customised behaviour.

  Parameters
  • parent (AbstractElement) – The element that is being added to
  • set (str, None, or False) – 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=False, returnelements=False)
  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)
  • returnelements – Return the actual matches within the alternatives, will return two-
    tuples where the first is an instance of Alternative and the second an instance of Class.

  Yields Alternative elements

ancestor (*Classes)
  Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-
  SuchAnnotation exception if not found.

  Parameters Classes (*) – The possible classes (AbstractElement or subclasses) to se-
    lect 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 (tuple of) classes (AbstractElement or subclasses). Not instances!

  Yields elements (instances derived from AbstractElement)
annotation (type, set=False)
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 False (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:
AllowInlineAnnotation.annotations() AbstractElement.select()

Raises NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)
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 False (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
  • AllowInlineAnnotation.annotations()
  • NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to
override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()
checkdeclaration()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch (function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

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=False`, `**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=False`)  
Returns an integer indicating whether such as annotation exists, and if so, how many.

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

`hasannotationlayer` (`annotationtype=None`, `set=False`)  
Does the specified annotation layer exist?

`hasphon` (`cls='current'`, `strict=True`, `correctionhandling=1`, `hidden=False`)  
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`, `hidden=False`)  
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.
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:

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

**Returns** `dict`

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

**parsecommonarguments**(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**classmethod parsexml**(node, doc, **kwargs)
Internal class method used for turning an XML element into an instance of the Class.

---

1.9. Structure Annotation Types 229
Parameters

• node - XML Element (*)
• doc - Document (*)

Returns An instance of the current Class.

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

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

The phonetic content will be constructed from child-elements whereever possible, as they are more specifc. 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 outputed, 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.
• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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'`, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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.

**resolvecorrection**

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!
rightcontext \((\text{size, placeholder} = \text{None}, \text{scope} = \text{None})\)

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 \((\text{Class, set} = \text{False, recursive} = \text{True, ignore} = \text{True, node} = \text{None})\)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** \((\text{class})\) – The class to select; any python class (not instance) subclassed off `AbstractElement`
- **Set** \((\text{str})\) – The set to match against, only elements pertaining to this set will be returned. If set to False (default), all elements regardless of set will be returned.
- **recursive** \((\text{bool})\) – 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`, `AlternativeLayers`, `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** \((\ast)\) – Reserved for internal usage, used in recursion.

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

**Example:**

```python
   ...
```

sentences \((\text{index} = \text{None})\)

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

**Parameters**

- **index** \((\text{int or None})\) – If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning a generator of all

setdoc \((\text{newdoc})\)

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

setdocument \((\text{doc})\)

Associate a document with this element.

**Parameters**

- **doc** \((\text{Document})\) – A document

Each element must be associated with a FoLiA document.

setparents ()

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

setprocessor \((\text{processor})\)

Sets the processor for this element, taking care of adding an annotator in the declarations.

settext \((\text{text, cls} = \text{‘current’})\)

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, hidden=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)

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.

- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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:
```python
for annotation in word:
    ...
```

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for `text()`

# 1.9.12 folia.main.ListItem

class folia.main.ListItem(doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement


Method Summary

__init__ (doc, *args, **kwargs) Initialize self.
accepts (Class[, raiseexceptions, parentinstance])
add (child, *args, **kwargs)
addable (parent[, set, raiseexceptions])
addidsuffix (idsuffix[, recursive])
addtoindex ([norecurse])
alternatives ([Class, set, returnelements])
ancestor (*Classes)
ancestors ([Class])
annotation (type[, set])
annotations (Class[, set])
annotator2processor ([annotator, . . . ])
append (child, *args, **kwargs)
checkdeclaration ()
context (size[, placeholder, scope])
copy ([newdoc, idsuffix])

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copychildren</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td></td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasannotationlayer([annotationtype, set])</td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, . . .])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, . . .])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>items([founditems])</td>
<td></td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layers([annotationtype, set])</td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>paragraphs([index])</td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
Table 22 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
**Class Attributes**

```python
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
```

**Method Details**

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

``` python
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
```

``` python
add(child, *args, **kwargs)
```
**classmethod addable** *(parent, set=False, 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, None, or False)* – 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=False, returnelements=False)*
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)
  - `returnelements` – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

- **Yields** Alternative elements

**ancestor** *(∗Classes)*
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
  Not instances!

- **Yields** elements (instances derived from AbstractElement)
annotation(type, set=False)

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 False (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:

AllowInlineAnnotation.annotations() AbstractElement.select()

Raises NoSuchAnnotation if no such annotation exists

annotations(Class, set=False)

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 False (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

• AllowInlineAnnotation.annotations()

• NoSuchAnnotation if no such annotation exists

annotator2processor(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append(child, *args, **kwargs)

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

custom (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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

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:

```python
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.

1.9. Structure Annotation Types

---

```python
def sense = word.annotation(folia.Sense)
def synset = sense.feat('synset')```
**class method findreplaceables** *(parent, set=False, **kwargs)*

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

**generate_id** *(cls)*

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

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

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

**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=False)*

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

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

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

Does the specified annotation layer exist?

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

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, hidden=False)*

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.
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:

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

**Returns** `dict`

`layers(annotationtype=None, set=False)`  
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.

`parsecommonarguments(doc, **kwargs)`  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

`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, hidden=False)`

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 outputed, 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.

• `hidden (bool)` – Include hidden elements, defaults to `False`.

Example:

```python
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, hidden=False)`

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 \texttt{CorrectionHandling.EITHER} if you don’t care.

\textbf{Returns} The phonetic content \texttt{(PhonContent)}

\textbf{Raises} \texttt{NoSuchPhon} if there is no phonetic content for the element

\textbf{See also:}

\texttt{phon()} \texttt{textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes(other)}

Returns a boolean indicating whether this element precedes the other element

\texttt{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.

\textbf{Parameters}

- \texttt{Class (\texttt{*})} – The class to select; any python class subclassed off \texttt{‘AbstractElement’}, may also be a tuple of multiple classes. Set to \texttt{True} to constrain to the same class as that of the current instance, set to \texttt{None} to not constrain at all

- \texttt{scope (\texttt{*})} – A list of classes which are never crossed looking for a next element. Set to \texttt{True} to constrain to a default list of structure elements (\texttt{Sentence,Paragraph,Division,Event, ListItem,Caption}), set to \texttt{None} to not constrain at all.

\texttt{classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)}

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

\texttt{classmethod relaxng_backwards()}

internal helper function for backward compatibility

\texttt{remove(child)}

Removes the child element

\texttt{replace(child, *args, **kwargs)}

Appends a child element like \texttt{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 \texttt{append()}

\textbf{Keyword Arguments}

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

- \texttt{be an alternative.(to)} –

\textbf{See} \texttt{AbstractElement.append()} for more information and all parameters.

\texttt{resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')}\texttt{resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')}\texttt{resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')}

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

\texttt{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=False, 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 False (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`, `AlternativeLayers`, `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, ...
```

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()`

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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
• **hidden (bool)** – Include hidden elements, defaults to `False`.

**Example:**

```python
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, hidden=False)*

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.
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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:

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

__len__ ()
Returns the number of child elements under the current element.

__str__ ()
Alias for text()

1.9.13 folia.main.Note

class folia.main.Note (doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement

Element used for notes, such as footnotes or warnings or notice blocks.

Method Summary

__init__ (doc, *args, **kwargs) Initialize self.

accepts(Class[, raiseexceptions, parentinstance])

add(child, *args, **kwargs)

addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.

addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.

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

alternatives([Class, set, returnelements]) Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

ancestor(*Classes) Find the most immediate ancestor of the specified type, multiple classes may be specified.

ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

annotation(type[, set]) Obtain a single annotation element.

annotations(Class[, set]) Obtain child elements (annotations) of the specified class.

annotator2processor([annotator, . . . ]) Converts annotator information to processor information (FoLiA v2).

append(child, *args, **kwargs) See AbstractElement.append()

checkdeclaration() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right.

copy([newdoc, idsuffix]) Make a deep copy of this element and all its children.

Continued on next page
Table 23 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren</code> (newdoc, idsuffix)</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs(index)</code></td>
<td>Returns a generator of Paragraph elements found recursively under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
Table 23 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon(cls, previousdelimiter, strict, …)</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent(cls, correctionhandling, hidden)</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, …])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, …])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>setttext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, …])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 23 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...

ANNOTATIONTYPE = 27

AUTH = True

AUTO_GENERATE_ID = True

HIDDEN = False

LABEL = 'Note'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 0

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)

PHONCONTAINER = False

PRIMARYELEMENT = True

PRINTABLE = True

REQUIRED_ATTRIBS = None

REQUIRED_DATA = None

SETONLY = False

SPEAKABLE = True

SUBSET = None

TEXTCONTAINER = False

TEXTDELMITER = '\n\n'

WREFABLE = False

XLINK = False

XMLTAG = 'note'

Method Details

__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=False, 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, None, or False) – 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=False, returnelements=False)

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)
• returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)
annotation(type, set=False)
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 False (default), all elements regardless of set will be returned.

Returns An element (instance derived from `AbstractElement`)  

Example:

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

See also:

- `AllowInlineAnnotation.annotations()`  
- `AbstractElement.select()`

Raises `NoSuchAnnotation` if no such annotation exists

annotations(Class, set=False)
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 False (default), all elements regardless of set will be returned.

Yields Elements (instances derived from `AbstractElement`)  

Example:

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

See also:

- `AbstractElement.select()`

Raises

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

annotator2processor(annotator=None, annotator_type=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append(child, *args, **kwargs)
See `AbstractElement.append()`
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
declared, raises an exception if not so, or auto-declares the annotation type if need be.

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

Returns int

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=False, **kwargs)
   Internal method to find replaceable elements. Auxiliary function used by AbstractElement.
   replace(). Can be overriden 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

getmetadat a (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 TEXTDE LIMITER attribute but may return a customised one instead.

hasannotation (Class, set=False)
   Returns an integer indicating whether such as annotation exists, and if so, how many.

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

hasannotationlayer (annotationtype=None, set=False)
   Does the specified annotation layer exist?

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
   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 correc-
     tion, and CorrectionHandling.EITHER if you don’t care.

   Returns bool

hastext (cls='current', strict=True, correctionhandling=1, hidden=False)
   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 en-
     countered. 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

\texttt{incorrection()}

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

\texttt{insert(index, child, *args, **kwargs)}

\texttt{items(founditems=[])}

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

\texttt{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:

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

Returns dict

\texttt{layers(annnotationtype=\texttt{None}, set=\texttt{False})}

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

\texttt{leftcontext(size, placeholder=\texttt{None}, scope=\texttt{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

\texttt{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

\texttt{originaltext(cls=\texttt{'original'})}

Alias for retrieving the original uncorrect text.

A call to \texttt{text()} with \texttt{correctionhandling=CorrectionHandling.ORIGINAL}

\texttt{paragraphs(index=None)}

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

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

\texttt{parsecommonarguments(doc, **kwargs)}

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

\texttt{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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden** (bool) – Include hidden elements, defaults to False.

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, hidden=False)

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 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.
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.

### precedes (`other`)

Returns a boolean indicating whether this element precedes the other element

### 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`, 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.

### classmethod relaxng(`includechildren=True, extraattrs=None, extraelements=None, origclass=None`)

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

### classmethod relaxng_backwards()

Internal helper function for backward compatibility

### 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.

### resolveoffsets (`begin, end, retaintokenisation=True, strictend=True, cls='current'`)

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

### resolveword (`id`)

1.9. Structure Annotation Types
rightcontext \((size, \text{placeholder}={\text{None}}, \text{scope}={\text{None}})\)

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 \((\text{Class, set}={\text{False}}, \text{recursive}={\text{True}}, \text{ignore}={\text{True}}, \text{node}={\text{None}})\)

Select child elements of the specified class.

A further restriction can be made based on set.

Parameters

- **Class** \((\text{class})\) – The class to select; any python class (not instance) subclassed off \(\text{AbstractElement}\)
- **Set** \((\text{str})\) – The set to match against, only elements pertaining to this set will be returned. If set to False (default), all elements regardless of set will be returned.
- **recursive** \((\text{bool})\) – 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, AlternativeLayers, Suggestion, and \(\text{folia.Original}\). 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** \((\ast)\) – Reserved for internal usage, used in recursion.

Yields

Elements (instances derived from \(\text{AbstractElement}\))

Example:

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

sentences \((index={\text{None}})\)

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

Parameters

- **index** \((\text{int or None})\) – If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning a generator of all

setdoc \((\text{newdoc})\)

Set a different document. Usually no need to call this directly, invoked implicitly by \(\text{copy()}\)

setdocument \((\text{doc})\)

Associate a document with this element.

Parameters

- **doc** \((\text{Document})\) – 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 \(\text{copy()}\)

setprocessor \((\text{processor})\)

Sets the processor for this element, taking care of adding an annotator in the declarations

settext \((\text{text, cls='current'}})\)

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, hidden=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 outputed, 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
• hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)

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.
- **hidden** (bool) – Include hidden elements, defaults to False.

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** (int) – 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()

1.9.14 folia.main.Paragraph

```python
class folia.main.Paragraph(doc, *args, **kwargs)

Bases: folia.main.AbstractStructureElement
```

Paragraph element. A structure element. Represents a paragraph and holds all its sentences (and possibly other structure Whitespace and Quotes).

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext(cls)</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
Table 24 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settextr(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 24 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml_string([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

- ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>), <class 'folia.main.AbstractInlineAnnotation'>)
- ANNOTATIONTYPE = 3
- AUTH = True
- AUTO_GENERATE_ID = True
- HIDDEN = False
- LABEL = 'Paragraph'
- OCCURRENCES = 0
- OCCURRENCES_PER_SET = 0
- OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
- PHONCONTAINER = False
- PRIMARYELEMENT = True
- PRINTABLE = True
- REQUIRED_ATTRIBS = None
- REQUIRED_DATA = None
- SETONLY = False
- SPEAKABLE = True
- SUBSET = None
- TEXTCONTAINER = False
- TEXTDELIMITER = '\n\n'
- WREFABLE = False
- 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=False, 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, None, or False)* – 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=False, returnelements=False)*
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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

**Yields** Alternative elements

**ancestor** *(Classes)*
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

**Example:**

```python
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** **Classes** *(tuple of) – The class or (tuple of classes (AbstractElement or subclasses).
Not instances!

**Yields** elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

`AllowInlineAnnotation.annotations()`  `AbstractElement.select()`

**Raises**  NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

`AbstractElement.select()`

**Raises**

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

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int
depdepthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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  
Internal method to find replaceable elements. Auxiliary function used by AbstractElement. replace(). Can be overridden for more fine-grained control.

generate_id  
getindex (child, recursive=True, ignore=True)  
Get the index at which an element occurs, recursive by default!

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

hasannotationlayer (annotationtype=None, set=False)  
Does the specified annotation layer exist?

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

*incorrectation()*

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:

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

**Returns** dict

*layers(annocationtype=None, set=False)*

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

*parsecommonarguments(doc, **kwargs)*

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

*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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- `hidden` (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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 \texttt{CorrectionHandling.EITHER} if you don’t care.

\textbf{Returns} The phonetic content (\texttt{PhonContent})

\textbf{Raises} NoSuchPhon if there is no phonetic content for the element

\textbf{See also:}

\texttt{phon()} \texttt{textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes(other)}

Returns a boolean indicating whether this element precedes the other element

\texttt{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.

\textbf{Parameters}

- \texttt{Class (*)} – The class to select; any python class subclassed off ‘AbstractElement’, may also be a tuple of multiple classes. Set to \texttt{True} to constrain to the same class as that of the current instance, set to \texttt{None} to not constrain at all

- \texttt{scope (*)} – A list of classes which are never crossed looking for a next element. Set to \texttt{True} to constrain to a default list of structure elements (Sentence,Paragraph,Division,Event, ListItem,Caption), set to \texttt{None} to not constrain at all.

\texttt{classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)}

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

\texttt{classmethod relaxng_backwards()}

internal helper function for backward compatibility

\texttt{remove(child)}

Removes the child element

\texttt{replace(child, *args, **kwargs)}

Appends a child element like \texttt{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()

\textbf{Keyword Arguments}

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

- \texttt{be an alternative. (to)} –

See \texttt{AbstractElement.append()} for more information and all parameters.

\texttt{resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')}

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

\texttt{resolveword(id)}
rightcontext \( (size, \text{placeholder}=\text{None}, \text{scope}=\text{None}) \)
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 \( (\text{Class}, \text{set}=\text{False}, \text{recursive}=\text{True}, \text{ignore}=\text{True}, \text{node}=\text{None}) \)
Select child elements of the specified class.

A further restriction can be made based on set.

Parameters

- **Class** \((\text{class})\) – The class to select; any python class (not instance) subclassed off
AbstractElement
- **Set** \((\text{str})\) – The set to match against, only elements pertaining to this set will be returned.
If set to False (default), all elements regardless of set will be returned.
- **recursive** \((\text{bool})\) – 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, AlternativeLayers, 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** \((\ast)\) – Reserved for internal usage, used in recursion.

Yields Elements (instances derived from AbstractElement)

Example:

```python
    ..
```

sentences \((\text{index}=\text{None})\)
Returns a generator of Sentence elements found (recursively) under this element

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

setdoc \((\text{newdoc})\)
Set a different document. Usually no need to call this directly, invoked implicitly by \texttt{copy()}

setdocument \((\text{doc})\)
Associate a document with this element.

Parameters **doc** \((\text{Document})\) – 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 \texttt{copy()}

setprocessor \((\text{processor})\)
Sets the processor for this element, taking care of adding an annotator in the declarations

settext \((\text{text}, \text{cls}='\text{current}')\)
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, hidden=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 outputed, 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

• **hidden**(bool) – Include hidden elements, defaults to `False`.

**Example:**

```python
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, hidden=False)*

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.
- **hidden (bool)** – Include hidden elements, defaults to `False`.

**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`
Iterate over all children of this element.

Example:

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

Returns the number of child elements under the current element.

Alias for `text()`

### 1.9.15 `folia.main.Part`

```python
class folia.main.Part (doc, *args, **kwargs)
```

**Bases:** `folia.main.AbstractStructureElement`

Generic structure element used to mark a part inside another block.

Do **not** use this for morphology, use `Morpheme` instead.

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code></td>
<td>Checks whether a new element can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code></td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>addidsuffix</code></td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addtoindex</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>alternatives</code></td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><code>ancestor</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code></td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><code>annotations</code></td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><code>annotator2processor</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code></td>
<td>See <code>AbstractElement.append()</code></td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td></td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasannotationlayer([annotationtype, set])</td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>Items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layers([annotationtype, set])</td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>paragraphs([index])</td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 25 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...

ANNOTATIONTYPE = 38

AUTH = True

AUTO_GENERATE_ID = True

HIDDEN = False

LABEL = 'Part'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 0

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)

PHONCONTAINER = False

PRIMARYELEMENT = True

PRINTABLE = True

REQUIRED_ATTRIBS = None

REQUIRED_DATA = None

SETONLY = False

SPEAKABLE = True

SUBSET = None

TEXTCONTAINER = False

TEXTDELSIMITER = ''

WREFABLE = False

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=False, 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, None, or False) – 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=False, returnelements=False)

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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```py
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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*

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 False (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:

- `AllowInlineAnnotation.annotations()`  
- `AbstractElement.select()`

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (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**

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

**annotator2processor** *(annotator=None, annotatorType=None, parentProcessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int
depdeepvalidation()
Perform deep validation of this element.

Raises DeepValidationError
depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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=False, **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=False)

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

See AllowInlineAnnotation.annotations() for a description of the parameters.

hasannotationlayer (annotationtype=None, set=False)

Does the specified annotation layer exist?

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

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, hidden=False)

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:

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

**Returns** `dict`

`layers(annotationtype=None, set=False)`  
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.

`parsecommonarguments(doc, **kwargs)`  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

`classmethod parsexml(node, doc, **kwargs)`  
Internal class method used for turning an XML element into an instance of the Class.

1.9. Structure Annotation Types
Parameters

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

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

```python
phon(cls='current', previousdelimiter='", strict=False, correctionhandling=1, hidden=False)
```

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

The phonetic content will be constructed from child-elements where ever 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 outputed, 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.
- **hidden** (`bool`) – Include hidden elements, defaults to `False`.

**Example:**

```python
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()`

```python
phoncontent(cls='current', correctionhandling=1, hidden=False)
```

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 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.
you want the content prior to correction, and `CorrectionHandling.EITHER` if you
do’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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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** (* falsely selects; 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** (* falsely – A list of classes which are never crossed looking for a next el-
  ement. 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)

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.**(to)

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

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:**

```python
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()`

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to **False**.

**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, hidden=False)*

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.
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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`
**__init__**
Iterate over all children of this element.

Example:
```python
for annotation in word:
    ...
```

**__len__**
Returns the number of child elements under the current element.

**__str__**
Alias for `text()`

### 1.9.16 folia.main.Quote

**class folia.main.Quote**(doc, *args, **kwargs)

Bases: folia.main.AbstractStructureElement

Quote: a structure element. For quotes/citations. May hold Word, Sentence or Paragraph data.

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code></td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td><code>add</code></td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td><code>addable</code></td>
<td>(parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td><code>addidsuffix</code></td>
<td>(idsuffix[, recursive])</td>
</tr>
<tr>
<td><code>addtoindex</code></td>
<td>([norecurse])</td>
</tr>
<tr>
<td><code>alternatives</code></td>
<td>([Class, set, returnelements])</td>
</tr>
<tr>
<td><code>ancestor</code></td>
<td>(*Classes)</td>
</tr>
<tr>
<td><code>ancestors</code></td>
<td>([Class])</td>
</tr>
<tr>
<td><code>annotation</code></td>
<td>(type[, set])</td>
</tr>
<tr>
<td><code>annotations</code></td>
<td>(Class[, set])</td>
</tr>
<tr>
<td><code>annotator2processor</code></td>
<td>([annotator, ...])</td>
</tr>
<tr>
<td><code>append</code></td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>()</td>
</tr>
<tr>
<td><code>context</code></td>
<td>(size[, placeholder, scope])</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>([newdoc, idsuffix])</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren(newdoc, idsuffix)</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hasText([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found recursively under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
Table 26 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True with text()</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>
Table 26 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring()</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTED_DATA</td>
<td>(&lt;class 'folia.main.AbstractAnnotationLayer'&gt;), (&lt;class 'folia.main.AbstractInlineAnnotation'&gt;),&lt;class 'folia.main.TextContent'&gt;, &lt;class 'folia.main.Utterance'&gt;, &lt;class 'folia.main.Whitespace'&gt;, &lt;class 'folia.main.Word'&gt;</td>
</tr>
<tr>
<td>ANNOTATIONTYPE</td>
<td>26</td>
</tr>
<tr>
<td>AUTH</td>
<td>True</td>
</tr>
<tr>
<td>AUTO_GENERATE_ID</td>
<td>True</td>
</tr>
<tr>
<td>HIDDEN</td>
<td>False</td>
</tr>
<tr>
<td>LABEL</td>
<td>'Quote'</td>
</tr>
<tr>
<td>OCCURRENCES</td>
<td>0</td>
</tr>
<tr>
<td>OCCURRENCES_PER_SET</td>
<td>0</td>
</tr>
<tr>
<td>OPTIONAL_ATTRIBS</td>
<td>(0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)</td>
</tr>
<tr>
<td>PHONCONTAINER</td>
<td>False</td>
</tr>
<tr>
<td>PRIMARYELEMENT</td>
<td>True</td>
</tr>
<tr>
<td>PRINTABLE</td>
<td>True</td>
</tr>
<tr>
<td>REQUIRED_ATTRIBS</td>
<td>None</td>
</tr>
<tr>
<td>REQUIRED_DATA</td>
<td>None</td>
</tr>
<tr>
<td>SETONLY</td>
<td>False</td>
</tr>
<tr>
<td>SPEAKABLE</td>
<td>True</td>
</tr>
<tr>
<td>SUBSET</td>
<td>None</td>
</tr>
<tr>
<td>TEXTCONTAINER</td>
<td>False</td>
</tr>
<tr>
<td>TEXTDELIMITER</td>
<td>'\n\n'</td>
</tr>
<tr>
<td>WREFABLE</td>
<td>False</td>
</tr>
<tr>
<td>XLINK</td>
<td>False</td>
</tr>
<tr>
<td>XMLTAG</td>
<td>'quote'</td>
</tr>
</tbody>
</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=False, 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, None, or False) – 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=False, returnelements=False)
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)
• returnelements – Return the actual matches within the alternatives, will return two-
tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)  
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (*) – The possible classes (AbstractElement or subclasses) to se-
lect 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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

- `AllowInlineAnnotation.annotations()`
- `AbstractElement.select()`

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

**annotations** *(Class, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

- `AbstractElement.select()`

**Raises**

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

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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

hasannotationlayer(annotationtype=None, set=False)
Does the specified annotation layer exist?

hasphon(cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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:
```python
import json
json.dumps(word.json())
```

Returns: `dict`

`layers(annotattiontype=None, set=False)`
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

`parsecommonarguments(doc, **kwargs)`
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

`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, hidden=False)`

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- `hidden (bool)` - Include hidden elements, defaults to False.

Example:

```python
class word:
    def phon(self):
        pass
```

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 (cls='current', correctionhandling=1, hidden=False)`

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 phonetic content. You can set this to CorrectionHandling.ORIGINAL if
you want the content prior to correction, and \texttt{CorrectionHandling.EITHER} if you
don’t care.

\textbf{Returns}  The phonetic content (\texttt{PhonContent})

\textbf{Raises}  \texttt{NoSuchPhon} if there is no phonetic content for the element

\textbf{See also:}
\texttt{phon()} \texttt{textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes(\texttt{other})}
Returns a boolean indicating whether this element precedes the other element

\texttt{previous(\texttt{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.

\textbf{Parameters}

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

\texttt{classmethod relaxng(\texttt{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)

\texttt{classmethod relaxng_backwards()}
internal helper function for backward compatibility

\texttt{remove(\texttt{child})}
Removes the child element

\texttt{replace(\texttt{child, *args, **kwargs})}
Appends a child element like \texttt{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 \texttt{append()}

\textbf{Keyword Arguments}

\begin{itemize}
  \item \texttt{alternative (\texttt{bool})} – If set to True, the \texttt{replaced} element will be made into an alter-
native. Simply use \texttt{AbstractElement.append()} if you want the added element
  \item \texttt{be an alternative. (to)} –
\end{itemize}

See \texttt{AbstractElement.append()} for more information and all parameters.

\texttt{resolveoffsets(\texttt{begin, end, retaintokenisation=True, strictend=True, cls='current'})}
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

\texttt{resolveword(\texttt{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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)

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.
- **hidden** (bool) – Include hidden elements, defaults to False.

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:

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

__len__()  
Returns the number of child elements under the current element.

__str__ ()  
Alias for `text()`

1.9.17 folia.main.Reference

class folia.main.Reference(doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement
A structural element that denotes a reference, internal or external. Examples are references to footnotes, bibliographies, hyperlinks.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
</tbody>
</table>

Continued on next page

1.9. Structure Annotation Types 305
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursively by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext(cls)</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsedxml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>
### Table 27 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolve()</code></td>
<td></td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 27 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.Alternative'>, ...
ANNOTATIONTYPE = 28
AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Reference'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' '
WREFABLE = False
XLINK = True
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=False, 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, None, or False)* – 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=False, returnelements=False)*

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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

**Yields** Alternative elements

**ancestor** *(*Classes)*

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).

Not instances!
Yields elements (instances derived from AbstractElement)

**annotation** *(type, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

- AllowInlineAnnotation.annotations() AbstractElement.select()

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

- AbstractElement.select()

**Raises**

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int
depdeepvalidation()
Perform deep validation of this element.

Raises DeepValidationError
depthfirstsearch(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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:

```python
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=False, **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
getmetatdata (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=False)
Returns an integer indicating whether such as annotation exists, and if so, how many.
See AllowInlineAnnotation.annotations() for a description of the parameters.
hasanntnotationlayer (annotationtype=None, set=False)
Does the specified annotation layer exist?
hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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

is_correction()

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:

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

Returns dict

layers(annotationtype=None, set=False)

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

left_context(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

original_text(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

parse_common_arguments(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

classmethod parsexml(node, doc, **kwargs)

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

1.9. Structure Annotation Types 313
Parameters

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

Returns An instance of the current Class.

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

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 outputed, 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.
- hidden (bool) - Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.

**precedes** *(other)*

Returns a boolean indicating whether this element precedes the other element

**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`, 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 el-
  ement. 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)

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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 alter-
  native. Simply use `AbstractElement.append()` if you want the added element

- **be an alternative.** *(to)* –

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

**resolve()**

**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**resolveword** *(id)*
rightcontext \( (size, \text{placeholder}=\text{None}, \text{scope}=\text{None}) \)

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 \( (\text{Class}, \text{set}=\text{False}, \text{recursive}=\text{True}, \text{ignore}=\text{True}, \text{node}=\text{None}) \)

Select child elements of the specified class.

A further restriction can be made based on set.

Parameters

- **Class** \( \text{(class)} \) – The class to select; any python class (not instance) subclassed off
  AbstractElement

- **Set** \( \text{(str)} \) – The set to match against, only elements pertaining to this set will be returned.
  If set to False (default), all elements regardless of set will be returned.

- **recursive** \( \text{(bool)} \) – 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, AlternativeLayers, Suggestion, and foila. 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** \( \ast \) – Reserved for internal usage, used in recursion.

Yields

Elements (instances derived from AbstractElement)

Example:

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

sentences \( \text{(index=\text{None})} \)

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

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

setdoc \( \text{(newdoc)} \)

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

setdocument \( \text{(doc)} \)

Associate a document with this element.

Parameters **doc** \( \text{(Document)} \) – A document

Each element must be associated with a FoLiA document.

setparents ()

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

setprocessor \( \text{(processor)} \)

Sets the processor for this element, taking care of adding an annotator in the declarations

settext \( \text{(text, cls='current')} \)

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**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, hidden=False)

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.
- **hidden** (`bool`) – Include hidden elements, defaults to `False`.

**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`

**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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for `text()`

### 1.9.18 folia.main.Row

class folia.main.Row(doc, *args, **kwargs)

Bases: folia.main.AbstractStructureElement

A row in a Table

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren(newdoc, idsuffix)</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_ids(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext(cls)</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>tokttext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 28 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

- `ACCEPTED_DATA` = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
- `ANNOTATIONTYPE` = None
- `AUTH` = True
- `AUTO_GENERATE_ID` = True
- `HIDDEN` = False
- `LABEL` = 'Table Row'
- `OCCURRENCES` = 0
- `OCCURRENCES_PER_SET` = 0
- `OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
- `PHONCONTAINER` = False
- `PRIMARYELEMENT` = True
- `PRINTABLE` = True
- `REQUIRED_ATTRIBS` = None
- `REQUIRED_DATA` = None
- `SETONLY` = False
- `SPEAKABLE` = True
- `SUBSET` = None
- `TEXTCONTAINER` = False
- `TEXTDELIMITER` = '\n'
- `WREFABLE` = False
- `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=False, 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, None, or False)* – 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=False, returnelements=False)*
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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

**Yields** *Alternative elements*

**ancestor** *(\*Classes)*
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No- SuchAnnotation exception if not found.

**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 (tuple of) classes *(AbstractElement or subclasses)*.
- Not instances!

**Yields** *elements (instances derived from AbstractElement)*
**annotation**(type, set=False)
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 False (default), all elements regardless of set will be returned.

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

Example:

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

**See also:**
- AllowInlineAnnotation.annotations()
- AbstractElement.select()

**Raises** NoSuchAnnotation if no such annotation exists

**annotations**(Class, set=False)
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 False (default), all elements regardless of set will be returned.

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

Example:

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

**See also:**
- AbstractElement.select()

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

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append**(child, *args, **kwargs)
See AbstractElement.append()
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int

dependency ()
Perform deep validation of this element.

Raises DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **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!

**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=False)

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

See AllowInlineAnnotation.annotations() for a description of the parameters.

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

Does the specified annotation layer exist?

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

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, hidden=False)

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
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:

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

**Returns** dict

`layers(annocationtype=None, set=False)` 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** (*)&nbsp; – 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** (*)&nbsp; – 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.

`parsecommonarguments(doc, **kwargs)` Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

`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, **hidden**=False)

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 outputed, 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.
- **hidden** (**bool**) – Include hidden elements, defaults to `False`

Example:

```python
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, **hidden**=False)

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.

**precedes(others)**

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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
- `to` –

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

**resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')**

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**resolveword(id)**
rightcontext \((\text{size}, \text{placeholder} = \text{None}, \text{scope} = \text{None})\)

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 \((\text{Class}, \text{set} = \text{False}, \text{recursive} = \text{True}, \text{ignore} = \text{True}, \text{node} = \text{None})\)

Select child elements of the specified class.

A further restriction can be made based on set.

**Parameters**

- **Class** \((\text{class})\) — The class to select; any python class (not instance) subclassed off `AbstractElement`

- **Set** \((\text{str})\) — The set to match against, only elements pertaining to this set will be returned. If set to False (default), all elements regardless of set will be returned.

- **recursive** \((\text{bool})\) — 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`, `AlternativeLayers`, `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** \((\ast)\) — Reserved for internal usage, used in recursion.

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

Example:

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

sentences \((\text{index} = \text{None})\)

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

**Parameters**

- **index** \((\text{int or None})\) — If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning a generator of all elements.

setdoc \((\text{newdoc})\)

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

setdocument \((\text{doc})\)

Associate a document with this element.

**Parameters**

- **doc** \((\text{Document})\) — 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()`

setprocessor \((\text{processor})\)

Sets the processor for this element, taking care of adding an annotator in the declarations.

settext \((\text{text, cls='current'})\)

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**Example:**

```python
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.

1.9. Structure Annotation Types 331
**textcontent** (cls='current', correctionhandling=1, hidden=False)

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.

- **hidden** (bool) – Include hidden elements, defaults to `False`.

**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()

1.9.19 folia.main.Sentence

class folia.main.Sentence(doc, *args, **kwargs)
Bases: folia.main.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

__init__(doc, *args, **kwargs)  
Example.

accepts(Class[, raiseexceptions, parentinstance])

add(child, *args, **kwargs)

addable(parent[, set, raiseexceptions])  
Tests whether a new element of this class can be added to the parent.

addidsuffix(idsuffix[, recursive])  
Appends a suffix to this element’s ID, and optionally to all child IDs as well.

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

alternatives([Class, set, returnelements])  
Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

ancestor(*Classes)  
Find the most immediate ancestor of the specified type, multiple classes may be specified.

ancestors([Class])  
Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

annotation(type[, set])  
Obtain a single annotation element.

annotations(Class[, set])  
Obtain child elements (annotations) of the specified class.

annotator2processor([annotator, ...])  
Converts annotator information to processor information (FoLiA v2).

append(child, *args, **kwargs)  
See AbstractElement.append()

checkdeclaration()  
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context(size[, placeholder, scope])  
Returns this word in context, {size} words to the left, the current word, and {size} words to the right.
### Table 29 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>corrections()</code></td>
<td>Are there corrections in this sentence?</td>
</tr>
<tr>
<td><code>correctwords(originalwords, newwords, **kwargs)</code></td>
<td>Generic correction method for words.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>deleteword(word, **kwargs)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>division()</code></td>
<td>Obtain the division this sentence is a part of (None otherwise).</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Inserts a word as a correction after an existing word.</td>
</tr>
<tr>
<td><code>insertword(newword, prevword, **kwargs)</code></td>
<td>Inserts a word as a correction before an existing word.</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
</tbody>
</table>
mergewords(newword, **kwargs) 

next([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.

originaltext([cls]) Alias for retrieving the original uncorrect text.

paragraph() Obtain the paragraph this sentence is a part of (None otherwise).

paragraphs([index]) Returns a generator of Paragraph elements found (recursively) under this element.

parsecommonarguments(doc, **kwargs) Internal function to parse common FoLiA attributes and sets up the instance accordingly.

parsexml(node, doc, **kwargs) Internal class method used for turning an XML element into an instance of the Class.

phon([cls, previousdelimiter, strict, ...]) Get the phonetic representation associated with this element (of the specified class)

phoncontent([cls, correctionhandling, hidden]) Get the phonetic content explicitly associated with this element (of the specified class).

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

precedes(other) Returns a boolean indicating whether this element precedes the other element.

previous([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.

relaxng([includechildren, extraattrs, ...]) Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

relaxng_backwards() Internal helper function for backward compatibility

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.

resolveoffsets(begin, end[, ...]) Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).

resolveword(id)

rightcontext(size[, placeholder, scope]) Returns the right context for an element, as a list.

select(Class[, set, recursive, ignore, node]) Select child elements of the specified class.

sentences([index]) Returns a generator of Sentence elements found (recursively) under this element.

setdoc(newdoc) Set a different document.

setdocument(doc) Associate a document with this element.

setparents() Correct all parent relations for elements within the scop.

setprocessor(processor) Sets the processor for this element, taking care of adding an annotator in the declarations.

settext(text[, cls]) Set the text for this element.

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

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

Continued on next page
Table 29 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>splitword(originalword, *newwords, **kwargs)</code></td>
<td>TODO: Write documentation</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmistring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

`ACCEPTED_DATA` = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ... , <class 'folia.main.String'>, <class 'folia.main.TextContent'>, <class 'folia.main.Whitespace'>, <class 'folia.main.Word'>)

`ANNOTATIONTYPE` = 9

`AUTH` = True

`AUTO_GENERATE_ID` = True

`HIDDEN` = False

`LABEL` = 'Sentence'

`OCCURRENCES` = 0

`OCCURRENCES_PER_SET` = 0

`OPTIONAL_ATTRIBS` = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)

`PHONCONTAINER` = False

`PRIMARYELEMENT` = True

`PRINTABLE` = True

`REQUIRED_ATTRIBS` = None

`REQUIRED_DATA` = None

`SETONLY` = False

`SPEAKABLE` = True

`SUBSET` = None

`TEXTCONTAINER` = False

`TEXTDELMITER` = ' '  

Chapter 1. Reading FoLiA
Method Details

__init__(doc, *args, **kwargs)

Example:

```python
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:

```python
paragraph.append(sentence)
```

See also:

AbstractElement.__init__()
Parameters

- **parent** (*AbstractElement*) – The element that is being added to
- **set** (*str, None, or False*) – 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=False, returnelements=False**)  
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)
- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields: Alternative elements

**ancestor** (**Classes**)  
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (*AbstractElement* or subclasses). Not instances!

Yields: elements (instances derived from *AbstractElement*)

**annotation** (**type, set=False**)  
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 False (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:**

`AllowInlineAnnotation.annotations()`  `AbstractElement.select()`

**Raises** NoSuchAnnotation if no such annotation exists

**annotations** *(Class, set=False)*

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 False (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**

• `AllowInlineAnnotation.annotations()`  
  
  **NoSuchAnnotation** if no such annotation exists

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`

**checkdeclaration** *

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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
depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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.feat('synset')
**Returns** str or list

`findcorrectionhandling(cls)`
Find the proper correction handling given a text class by looking in the underlying corrections where it is reused.

**classmethod findreplaceables(parent, set=False, **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=False)`
Returns an integer indicating whether such an annotation exists, and if so, how many.

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

`hasannotationlayer(annnotiontype=None, set=False)`
Does the specified annotation layer exist?

`hasphon(cls='current', strict=True, correctionhandling=1, hidden=False)`
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, hidden=False)`
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`.  

1.9. Structure Annotation Types
• **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** *(attribs=None, recurse=True, ignorelist=False)*

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

  **Example:**

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

  **Returns** dict

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

  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’th element (starting at 0) instead of returning the generator of all

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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

`textcontent()`

**phoncontent**( `cls='current', correctionhandling=1, hidden=False` )

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.

**precedes**( `other` )

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

Internal helper function for backward compatibility

```python
remove(child)
```

Removes the child element

```python
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.

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(id)
```

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

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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.

1.9. Structure Annotation Types
• **node** (+) – Reserved for internal usage, used in recursion.

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

Example:

```python
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original, ...
```

**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()*

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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.
- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.
- **hidden (bool)** – Include hidden elements, defaults to False.

**Returns** The phonetic content (TextContent)

**Raises** NoSuchText if there is no text content for the element
See also:

\text{text()} \ phoncontent() \ phon()

\text{textvalidation}(\text{warnonly}=\text{None})
Run text validation on this element. Checks whether any text redundancy is consistent and whether offsets are valid.

- **Parameters** \text{warnonly} (\text{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** \text{bool}

\text{toktext}(\text{cls}=\text{current})
Alias for \text{text()} with \text{retaintokenisation=True}

\text{updatetext}()
Recompute textual value based on the text content of the children. Only supported on elements that are a \text{TEXTCONTAINER}

\text{words}(\text{index}=\text{None})
Returns a generator of Word elements found (recursively) under this element.

- **Parameters** \text{index} (+) – If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning the list of all

\text{xml}(\text{attribs}=\text{None}, \text{elements}=\text{None}, \text{skipchildren}=\text{False})
Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

- **Returns** an lxml.etree.Element

See also:

\text{AbstractElement.xmlstring()} - for direct string output

\text{xmlstring}(\text{pretty_print}=\text{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** \text{str}

\text{__iter__}()
Iterate over all children of this element.

Example:

\begin{verbatim}
for annotation in word:
    ...
\end{verbatim}

\text{__len__}()
Returns the number of child elements under the current element.

\text{__str__}()
Alias for \text{text()}

1.9.20 \texttt{folia.main.Table}

\texttt{class folia.main.Table}(\texttt{doc, *args, **kwargs})
Bases: \texttt{folia.main.AbstractStructureElement}
A table consisting of Row elements that in turn consist of Cell elements

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code>(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code>(child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addaddable</code>(parent[, set, raiseexceptions])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>addidsuffix</code>(idsuffix[, recursive])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><code>ancestors</code>(parent[, set, raiseexceptions])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><code>annotation</code>(type[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><code>annotation2processor</code>([annotator, …])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code>()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy</code>([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code>(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description</code>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables</code>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 30 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found recursively under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 30 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>words(index)</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

### Class Attributes

- **ACCEPTED_DATA** = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
- **ANNOTATIONTYPE** = 36
- **AUTH** = True
- **AUTO_GENERATE_ID** = True
- **HIDDEN** = False
- **LABEL** = 'Table'
- **OCCURRENCES** = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHON_CONTAINER = False
PRIMARY_ELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXT_CONTAINER = False
TEXT_DELIMITER = '\n\n'
WREFABLE = False
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=False, 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, None, or False) – 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\(\text{\textit{norecurse=}}[]\)\)
Make sure this element (and all subelements), are properly added to the index.
Mostly for internal use.

alternatives\(\text{\textit{Class=}}\text{\textit{None, set=}}\text{\textit{False, returnelements=}}\text{\textit{False}}\)\)
Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.

Parameters
- Class\(\text{\textit{class}}\) – 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\(\text{\textit{str}}\) – The set you want to retrieve (defaults to None, which selects irregardless of set)
- returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor\(\text{\textit{*Classes}}\)\)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

Parameters Classes\(\text{\textit{\textit{}}}\) – The possible classes (AbstractElement or subclasses) to select from. Not instances!

Example:

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

ancestors\(\text{\textit{Class=none}}\)\)
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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)

annotation\(\text{\textit{type, set=}}\text{\textit{False}}\)\)
Obtain a single annotation element.
A further restriction can be made based on set.

Parameters
- Class\(\text{\textit{class}}\) – The class to select; any python class (not instance) subclassed off AbstractElement
- Set\(\text{\textit{str}}\) – The set to match against, only elements pertaining to this set will be returned. If set to False (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:

AllowInlineAnnotation.annotations() AbstractElement.select()
Raises NoSuchAnnotation if no such annotation exists

annations (Class, set=False)
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 False (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
- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declarbes the annotation type if need be.

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=False, 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

depthfirstsearch(function)
   Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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 correction handling given a text class by looking in the underlying corrections where it is reused

classmethod findreplaceables(parent, set=False, **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=False)
   Returns an integer indicating whether such an annotation exists, and if so, how many.
   
   See AllowInlineAnnotation.annotations`() for a description of the parameters.

hasannotationlayer(annotationtype=None, set=False)
   Does the specified annotation layer exist?
hasphon(\textit{cls}=\textit{current}, \textit{strict}=True, \textit{correctionhandling}=1, hidden=False)

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

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

\textbf{Parameters}

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

\textbf{Returns} bool

hastext(\textit{cls}=\textit{current}, \textit{strict}=True, \textit{correctionhandling}=1, hidden=False)

Does this element have text (of the specified class)

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

\textbf{Parameters}

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

\textbf{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(\textit{index}, \textit{child}, *\textit{args}, **\textit{kwargs})

\textbf{Items} (\textit{founditems=[]})

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

json(\textit{attribs=None}, \textit{recurse=True}, \textit{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())
```

\textbf{Returns} dict

layers(\textit{annotationtype=None}, \textit{set=False})

Returns a list of annotation layers found directly under this element, does not include alternative layers
leftcontext \((\text{size}, \text{placeholder}={None}, \text{scope}={None})\)

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 \((\text{Class}={True}, \text{scope}={True}, \text{reverse}={False})\)

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.

\textbf{Parameters}

- \textbf{Class} (*) – The class to select; any python class subclassed off 'AbstractElement', may
  also be a tuple of multiple classes. Set to \text{True} to constrain to the same class as that of
  the current instance, set to \text{None} to not constrain at all

- \textbf{scope} (*) – A list of classes which are never crossed looking for a next el-
  ement. Set to \text{True} to constrain to a default list of structure elements (Sentence,Paragraph,Division,Event, ListItem,Caption), set to \text{None} to not constrain at all.

originaltext \((\text{cls}='\text{original}')\)

Alias for retrieving the original uncorrect text.

A call to \text{text()} with \text{correctionhandling=CorrectionHandling.ORIGINAL}

paragraphs \((\text{index}={None})\)

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

\textbf{Parameters} \textbf{index} \((\text{int or None})\) – If set to an integer, will retrieve and return the n’th
  element (starting at 0) instead of returning the generator of all

parsecommonarguments \((\text{doc}, **\text{kwargs})\)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

classmethod parsexml \((\text{node}, \text{doc}, **\text{kwargs})\)

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

\textbf{Parameters}

- \textbf{node} – XML Element (*)–

- \textbf{doc} – Document (*)–

\textbf{Returns} An instance of the current Class.

phon \((\text{cls}='\text{current}', \text{previousdelimiter}=' ', \text{strict}={False}, \text{correctionhandling}={1}, \text{hidden}={False})\)

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

The phonetic content will be constructed from child-elements whereever possible, as they are more spe-
ific. 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.

\textbf{Parameters}

- \textbf{cls} (str) – The class of the phonetic content to obtain, defaults to \text{current}.

- \textbf{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 \text{False}.

- \textbf{previousdelimiter} (str) – Can be set to a delimiter that was last outputed, useful
  when chaining calls to phon(). Defaults to an empty string.

- \textbf{strict} (bool) – Set this if you are strictly interested in the phonetic content explicitly
  associated with the element, without recursing into children. Defaults to \text{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.

• **hidden** (bool) – Include hidden elements, defaults to False.

Example:

```python
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
- `textcontent()`

```python
phoncontent(cls='current', correctionhandling=1, hidden=False)
```

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:

```python
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.

**precedes** *(other)*

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

Internal helper function for backward compatibility.

```python
remove(child)
```

Removes the child element.

```python
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 to be an alternative.

• **be an alternative.** (to)

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

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(id)
```

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.

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

**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()*

**setprocessor** (*processor*)

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.
- **hidden (bool)** – Include hidden elements, defaults to False.

**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** (*int*) – 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)
Serialises the FoLiA element and all its contents to XML.

**Arguments** are mostly for internal use.

**Returns** *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:

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

### __len__ ()
Returns the number of child elements under the current element.

### __str__ ()
Alias for `text()`

## 1.9.21 folia.main.Term

### class folia.main.Term (doc, *args, **kwargs)

**Bases:** folia.main.AbstractStructureElement
A term, often used in context of *Entry*

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td></td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives([Class, set, returnelements])</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 31 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 31 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolveoffsets</code> (begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code> (id)</td>
<td></td>
</tr>
<tr>
<td><code>rightcontext</code> (size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code> (Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences</code> ([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>setdoc</code> (newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code> (doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents</code> ()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor</code> (processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext</code> ([text[, cls]])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker</code> ()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src</code> ()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext</code> ([cls])</td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text</code> ([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent</code> ([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation</code> ([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext</code> ([cls])</td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext</code> ()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words</code> ([index])</td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml</code> ([attribs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring</code> ([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__</code> ()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__</code> ()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__</code> ()</td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

```python
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
ANNOTATIONTYPE = 41
AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Term'
OCCURRENCES = 0
```

1.9. Structure Annotation Types 365
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = '\n\n'
WREFABLE = False
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=False, raiseexceptions=True)
Tests whether a new element of this class can be added to the parent.

Parameters

• parent (AbstractElement) – The element that is being added to
• set (str, None, or False) – 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=False, returnelements=False)
   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)
   • returnelements – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

   Yields Alternative elements

ancestor (*Classes)
   Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

   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 (tuple of) classes (AbstractElement or subclasses). Not instances!

   Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
   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 False (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:
   AllowInlineAnnotation.annotations() AbstractElement.select()
Raises  NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)
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 False (default), all elements regardless of set will be returned.

Yields  Elements (instances derived from AbstractElement)

Example:

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

See also:
AbstractElement.select()

Raises
  • AllowInlineAnnotation.annotations()
  • NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False*, *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

**depthfirstsearch** *(function)*
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

**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:

```python
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=False, **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.

**hasannotation** (*Class*, *set=False*)
Returns an integer indicating whether such an annotation exists, and if so, how many.

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

**hasannotationlayer** *(annotationtype=None, set=False)*
Does the specified annotation layer exist?
hasphon  
\(\text{cls='current', strict=True, correctionhandling=1, hidden=False}\)

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  
\(\text{cls='current', strict=True, correctionhandling=1, hidden=False}\)

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

incorrect()  
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(attributes=None, recurse=True, ignorelist=False)

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

Example:

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

**Returns** dict

layers(annotationtype=None, set=False)

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

parsecommonarguments (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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
- `textcontent()`

```python
phoncontent(cls='current', correctionhandling=1, hidden=False)
```

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()`

```python
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.

```python
precedes(other)
```

Returns a boolean indicating whether this element precedes the other element

```python
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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

internal helper function for backward compatibility

```python
remove(child)
```

Removes the child element

```python
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.

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(id)
```

```python
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

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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.

1.9. Structure Annotation Types 373
- **node** (*) – Reserved for internal usage, used in recursion.

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

Example:

```python
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original, →
  ...
```

**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()`

**setprocessor (processor)**

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.
- **hidden (bool)** – Include hidden elements, defaults to False.

Returns The phonetic content (TextContent)

Raises NoSuchText if there is no text content for the element

1.9. Structure Annotation Types
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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for `text()`

1.9.22 folia.main.TableHead

class folia.main.TableHead(`doc, *args, **kwargs`)  
Bases: folia.main.AbstractStructureElement
Encapsulated the header of a table, contains *Cell* elements

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add</td>
<td>Adds a new element to the parent.</td>
</tr>
<tr>
<td>addable</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>alternatives</td>
<td>Generates a list of alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td>ancestor</td>
<td>finds the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td>annotations</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling</td>
<td>Finds the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id</td>
<td></td>
</tr>
</tbody>
</table>

**Continued on next page**
Table 32 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer([annotationtype, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 32 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolveoffsets</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code></td>
<td>ID</td>
</tr>
<tr>
<td><code>rightcontext</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><code>setdoc</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext</code></td>
<td>Alias for <code>text()</code> with strict=True</td>
</tr>
<tr>
<td><code>text</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext</code></td>
<td>Alias for <code>text()</code> with retaintokenisation=True</td>
</tr>
<tr>
<td><code>updatetext</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

`ACCEPTED_DATA` = (class 'folia.main.AbstractAnnotationLayer'), (class 'folia.main.AbstractInlineAnnotation'), (class 'folia.main.AbstractStructuralAnnotation'), (class 'folia.main.Adamcik'), (class 'folia.main.AbstractMorphologicalAnnotation'), (class 'folia.main.AbstractLexicalUnit'), (class 'folia.main.FoliaElement'), (class 'folia.main.Concept'), (class 'folia.main.ConceptLayer'), (class 'folia.main.AnnotationLayer'), (class 'folia.main.Metric'), (class 'folia.main.Part'), (class 'folia.main.Relation'), (class 'folia.main.Row')

`ANNOTATIONTYPE` = None

`AUTH` = True

`AUTO_GENERATE_ID` = True

`HIDDEN` = False

`LABEL` = 'Table Header'

`OCCURRENCES` = 0

1.9. Structure Annotation Types
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 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 = '

'
WREFABLE = False
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)

classmethod addable(parent, set=False, 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, None, or False) – 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=False, returnelements=False)  
    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)
    • **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

    Yields Alternative elements

ancestor (*Classes)  
    Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

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

    Example:
    ```python
    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 (tuple of) classes (AbstractElement or subclasses). Not instances!

    Yields elements (instances derived from AbstractElement)

annotation (type, set=False)  
    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 False (default), all elements regardless of set will be returned.

    Returns An element (instance derived from AbstractElement)

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

See also: AllowInlineAnnotation.annotations() AbstractElement.select()
Raises NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)
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 False (default), all elements regardless of set will be returned.

Yields Elements (instances derived from AbstractElement)

Example:

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

See also:

AbstractElement.select()

Raises

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)*  
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns  int

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

**deepvalidation** ()  
Perform deep validation of this element.

**deepvalidation**  
Perform deep validation of this element.

**description** ()  
Obtain the description associated with the element.

**description**  
Obtain the description associated with the element.

**feat** *(subset)*  
Obtain the feature class value of the specific subset.

**feat**  
Obtain the feature class value of the specific subset.

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

Example:

```python
class = word.annotation(folia.Sense)
synset = class.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

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

**classmethod findreplaceables** *(parent, set=False, **kwargs)*  
Internal method to find replaceable elements. Auxiliary function used by AbstractElement.replace(). Can be overridden for more fine-grained control.

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

**generate_id** *(cls)*

**generate_id**

**getindex** *(child, recursive=True, ignore=True)*  
Get the index at which an element occurs, recursive by default!

**getindex**  
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

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

**gettextdelimiter** *(retaintokenisation=False)*  
Return the text delimiter for this class.

**gettextdelimiter**  
Return the text delimiter for this class.

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

**hasannotation** *(Class, set=False)*  
Returns an integer indicating whether such as annotation exists, and if so, how many.

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

See AllowInlineAnnotation.annotations() for a description of the parameters.

**hasannotationlayer** *(annotationtype=None, set=False)*  
Does the specified annotation layer exist?
hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)

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, hidden=False)

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:

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

Returns dict

layers (annotationtype=None, set=False)

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

parsecommonarguments (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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

- `textcontent()`

**phoncontent**(cls=`current`, correctionhandling=1, hidden=False)

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

Internal helper function for backward compatibility

```python
remove(child)
```

Removes the child element

```python
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.

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls=’current’)
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
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

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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.

1.9. Structure Annotation Types
• **node** (*node*) – Reserved for internal usage, used in recursion.

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

```python
...
```

### `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()`

### `setprocessor` *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

### `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, hidden=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 outputed, 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
- hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)

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.
- hidden (bool) – Include hidden elements, defaults to False.

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(attribs=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:

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

.. __len__()

Returns the number of child elements under the current element.

.. __str__()

Alias for .. text()
A full text. This is a high-level element (not to be confused with TextContent!). This element may contain `Division`, `Paragraph`, `Sentence`, etc.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code> (Class, raiseexceptions, parentinstance)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code> (child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addidsuffix</code> (idsuffix, recursive)</td>
<td>Makes sure this element (and all subelements) are properly added to the index.</td>
</tr>
<tr>
<td><code>addtoindex</code> (norecurse)</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><code>ancestor</code> (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code> (Class)</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code> (type, set)</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><code>annotations</code> (Class, set)</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><code>annotator2processor</code> ([annotator, …])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code> (child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code> ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code> (size, placeholder, scope)</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy</code> (newdoc, idsuffix)</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code> (newdoc, idsuffix)</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code> (**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code> (Class, set, recursive, ignore, node)</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code> ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code> (function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description</code> ()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code> (subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code> (cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables</code> (parent[. set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generate_id(cls)</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers([annotationtype, set])</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs([index])</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 33 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolveoffsets(begin, end, ...)</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

- `ACCEPTED_DATA` = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.AbstractInlineAnnotation'>, ...
- `ANNOTATIONTYPE` = None
- `AUTH` = True
- `AUTO_GENERATE_ID` = True
- `HIDDEN` = False
- `LABEL` = 'Text Body'
- `OCCURRENCES` = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 2, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYPELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELMITER = '\n\n\n'
WREFABLE = False
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)
Tests whether a new element of this class can be added to the parent.

Parameters

• parent (AbstractElement) – The element that is being added to
• set (str, None, or False) – 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 \( (\text{norecurse}=[]) \)

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

Mostly for internal use.

alternatives \( (\text{Class=None, set=False, returnelements=False}) \)

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

Parameters

- **Class** \( (\text{class}) \) – 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** \( (\text{str}) \) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor \( (*\text{Classes}) \)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

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

ancestors \( (\text{Class=None}) \)

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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)

annotation \( (\text{type, set=False}) \)

Obtain a single annotation element.

A further restriction can be made based on set.

Parameters

- **Class** \( (\text{class}) \) – The class to select; any python class (not instance) subclassed off AbstractElement

- **Set** \( (\text{str}) \) – The set to match against, only elements pertaining to this set will be returned. If set to False (default), all elements regardless of set will be returned.

Returns An element (instance derived from AbstractElement)

Example:

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

See also:

AllowInlineAnnotation.annotations() AbstractElement.select()
Raises NoSuchAnnotation if no such annotation exists

**annotations** (*Class, set=False*)

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 False (default), all elements regardless of set will be returned.

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

Example:

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

See also:

AbstractElement.select()

Raises

- AllowInlineAnnotation.annotations()
- NoSuchAnnotation if no such annotation exists

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

**depthfirstsearch** *(function)*
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

```python
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=False, **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=False)*
Returns an integer indicating whether such as annotation exists, and if so, how many.

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

**hasannotationlayer** *(annotationtype=None, set=False)*
Does the specified annotation layer exist?

1.9. Structure Annotation Types 397
hasphon \( \text{cls='current', strict=True, correctionhandling=I, hidden=False} \)

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

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

**Parameters**

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

**Returns** bool

hastext \( \text{cls='current', strict=True, correctionhandling=I, hidden=False} \)

Does this element have text (of the specified class)

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

**Parameters**

- \texttt{cls (str)} – The class of the text content to obtain, defaults to \texttt{current}.
- \texttt{strict (bool)} – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to \texttt{True}.
- \texttt{correctionhandling} – Specifies what text to check for when corrections are encountered. The default is \texttt{CorrectionHandling.CURRENT}, which will retrieve the corrected/current text. You can set this to \texttt{CorrectionHandling.ORIGINAL} if you want the text prior to correction, and \texttt{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 \( \text{index, child, *args, **kwargs} \)

items \( \text{founditems=[]} \)

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

json \( \text{attribs=None, recurse=True, ignorelist=False} \)

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

**Example:**

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

**Returns** dict

layers \( \text{annotationtype=None, set=False} \)

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

parsecommonarguments(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

• **hidden** (bool) – Include hidden elements, defaults to False.

Example:

```python
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
- `textcontent()`

`phoncontent(cls='current', correctionhandling=1, hidden=False)`

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.

`precedes(other)`

Returns a boolean indicating whether this element precedes the other element

`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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

internal helper function for backward compatibility

```python
remove(child)
```

Removes the child element

```python
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.

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(id)
```

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

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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:

```python
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()*

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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 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.
- **normalize_spaces** *(bool)* – Return the text with multiple spaces, linebreaks, tabs normalized to single spaces
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**Example:**

```python
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, hidden=False)*

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.
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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(attribs=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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

1.9.24 folia.main.Whitespace

class folia.main.Whitespace (doc, *args, **kwargs)
Bases: folia.main.AbstractStructureElement
Whitespace element, signals a vertical whitespace

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><strong>accepts</strong>(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td><strong>add</strong>(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td><strong>addable</strong>(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><strong>addidsuffix</strong>(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><strong>addtoindex</strong>[norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><strong>alternatives</strong>(Class, set, returnelements))</td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><strong>ancestor</strong>(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><strong>ancestors</strong>(Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><strong>annotation</strong>(type[, set])</td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><strong>annotations</strong>(Class[, set])</td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><strong>annotator2processor</strong>([annotator, …])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><strong>append</strong>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><strong>checkdeclaration</strong>()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><strong>context</strong>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><strong>copy</strong>(newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren</strong>(newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>correct</strong>(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count</strong>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><strong>description</strong>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong>(cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><strong>findreplaceables</strong>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 34 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getindex</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasannotationlayer</code></td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td><code>hasphon</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>(index, child, *args, **kwargs)</td>
</tr>
<tr>
<td><code>items</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layers</code></td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td><code>leftcontext</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext</code></td>
<td>(cls) Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>paragraphs</code></td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>parsecommonarguments</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon</code></td>
<td>(cls, previousdelimiter, strict, ...) Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent</code></td>
<td>(cls, strict, correctionhandling, hidden) Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>postappend</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element ( lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>(child, *args, **kwargs) Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>
Table 34 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>sentences([index])</code></td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>words([index])</code></td>
<td>Returns a generator of Word elements found (recursively) under this element.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractAnnotationLayer'>, <class 'folia.main.Alternative'>, ...

ANNOTATIONTYPE = 7
AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Whitespace'
OCCURRENCES = 0
```
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ''
WREFABLE = False
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=False, raiseexceptions=True)
Tests whether a new element of this class can be added to the parent.

Parameters

- **parent** (AbstractElement) – The element that is being added to
- **set** (str, None, or False) – 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=False, returnelements=False)

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)

- **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of Alternative and the second an instance of Class.

Yields Alternative elements

ancestor (*Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses). Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)

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 False (default), all elements regardless of set will be returned.

Returns An element (instance derived from AbstractElement)

Example:

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

See also:

AllowInlineAnnotation.annotations() AbstractElement.select()
Raises NoSuchAnnotation if no such annotation exists

annotations (Class, set=False)
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 False (default), all elements regardless of set will be returned.

Yields Elements (instances derived from AbstractElement)

Example:

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

See also:

AbstractElement.select()

Raises

• AllowInlineAnnotation.annotations()

• NoSuchAnnotation if no such annotation exists

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

countext (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=False, 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 DeepValidation`Error`

**depthfirstsearch**(function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

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

Returns str or list

**findcorrectionhandling**(cls)

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

**classmethod findreplaceables**(parent, set=False, **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=False)

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

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

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

Does the specified annotation layer exist?
hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)
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

incorrect ()
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:

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

Returns dict

layers (annotationtype=None, set=False)
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=\text{True}, scope=\text{True}, reverse=\text{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

- \textbf{Class} (\text{str}) – The class to select; any python class subclassed off \textit{AbstractElement}, may also be a tuple of multiple classes. Set to \text{True} to constrain to the same class as that of the current instance, set to \text{None} to not constrain at all

- \textbf{scope} (\text{str}) – A list of classes which are never crossed looking for a next element. Set to \text{True} to constrain to a default list of structure elements (Sentence,Paragraph,Division,Event, ListItem, Caption), set to \text{None} to not constrain at all.

originaltext (\text{cls=\text{original}})

Alias for retrieving the original uncorrect text.

A call to \texttt{text()} with \texttt{correctionhandling=CorrectionHandling.ORIGINAL}

paragraphs (index=None)

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

Parameters \textbf{index} (\text{int or \text{None}}) – If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning the generator of all

parsecommonarguments (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

classmethod parsexml (node, doc, **kwargs)

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

Parameters

- \textbf{node} – XML Element (\text{xml}) –

- \textbf{doc} – Document (\text{document}) –

Returns An instance of the current Class.

phon (\text{cls=\text{current}}, previousdelimiter=”, strict=False, correctionhandling=1, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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

- \textbf{cls} (\text{str}) – The class of the phonetic content to obtain, defaults to \text{current}.

- \textbf{retaintokenisation} (\text{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 \text{False}.

- \textbf{previousdelimiter} (\text{str}) – Can be set to a delimiter that was last outputed, useful when chaining calls to \texttt{phon()}. Defaults to an empty string.

- \textbf{strict} (\text{bool}) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to \text{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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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
- `textcontent()`

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

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element.

**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’, 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.

```python
classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

internal helper function for backward compatibility

```python
remove(child)
```

Removes the child element

```python
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.

```python
resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(id)
```

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.

```python
select(Class, set=False, 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 False (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, AlternativeLayers, 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.

1.9. Structure Annotation Types 415
• node (*) – Reserved for internal usage, used in recursion.

Yields Elements (instances derived from AbstractElement)

Example:

```python
...
```

**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()*

**setprocessor (processor)**

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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

- **hidden (bool)** – Include hidden elements, defaults to False.

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, hidden=False)

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.

- **hidden (bool)** – Include hidden elements, defaults to False.

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:

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

`__len__ ()`
Returns the number of child elements under the current element.

`__str__ ()`
Alias for `text()`

## 1.9.25 folia.main.Word

class folia.main.Word(doc, *args, **kwargs)
main.AllowCorrections

Word (aka token) element. Holds a word/token and all its related token annotations.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Constructor for words.</td>
</tr>
<tr>
<td><code>accepts</code></td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code></td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addable</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>alternativelayers</code></td>
<td>Generator over alternative layers, either all or only of a specific annotation type, and possibly restrained also by folia set.</td>
</tr>
<tr>
<td><code>alternatives</code></td>
<td>Generator over alternatives, either all or only of a specific annotation type, and possibly restrained also by set.</td>
</tr>
<tr>
<td><code>ancestor</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code></td>
<td>Obtain a single annotation element.</td>
</tr>
<tr>
<td><code>annotations</code></td>
<td>Obtain child elements (annotations) of the specified class.</td>
</tr>
<tr>
<td><code>annotator2processor</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code></td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 35 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>division()</td>
<td>Obtain the deepest division this word is a part of, otherwise return None</td>
</tr>
<tr>
<td>domain([set])</td>
<td>Shortcut: returns the FoLiA class of the domain annotation (will return only one if there are multiple!)</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set!])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>findspans(type[, set, alternatives, ...])</td>
<td>Yields span annotation elements of the specified type that include this word.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td></td>
</tr>
<tr>
<td>getcorrection([set, cls])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getcorrections([set, cls])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasannotationlayer([annotationtype, set])</td>
<td>Does the specified annotation layer exist?</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layers([annotationtype, set])</td>
<td>Returns a list of annotation layers found directly under this element, does not include alternative layers</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>lemma([set])</td>
<td>Shortcut: returns the FoLiA class of the lemma annotation (will return only one if there are multiple!)</td>
</tr>
<tr>
<td>morpheme(index[, set])</td>
<td>Returns a specific morpheme, the n’th morpheme (given the particular set if specified).</td>
</tr>
<tr>
<td>morphemes([set])</td>
<td>Generator yielding all morphemes (in a particular set if specified).</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>paragraph()</td>
<td>Obtain the paragraph this word is a part of, otherwise return None</td>
</tr>
<tr>
<td>paragraphs([index])</td>
<td>Returns a generator of Paragraph elements found (recursively) under this element.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parsecommonarguments</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>phoneme(index[, set])</td>
<td>Returns a specific phoneme, the n’th morpheme (given the particular set if specified).</td>
</tr>
<tr>
<td>phonemes([set])</td>
<td>Generator yielding all phonemes (in a particular set if specified).</td>
</tr>
<tr>
<td>post([set])</td>
<td>Shortcut: returns the FoLiA class of the PoS annotation (will return only one if there are multiple!)</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[,...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>sense([set])</td>
<td>Shortcut: returns the FoLiA class of the sense annotation (will return only one if there are multiple!)</td>
</tr>
<tr>
<td>sentence()</td>
<td>Obtain the sentence this word is a part of, otherwise return None</td>
</tr>
<tr>
<td>sentences([index])</td>
<td>Returns a generator of Sentence elements found (recursively) under this element</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
</tbody>
</table>
Table 35 – continued from previous page

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

Class Attributes

```

ANNOTATIONTYPE = 1

AUTH = True
AUTO_GENERATE_ID = True
HIDDEN = False
LABEL = 'Word/Token'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 3, 5, 8, 6, 7, 9, 10, 11, 13)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = ' ' 
```
WREFABLE = True
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:

```python
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:

```python
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=False, 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, None, or False) – 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.

**alternativelayers**(type, set=False, returnelements=False)

Generator over alternative layers, either all or only of a specific annotation type, and possibly restrained also by `folia` set.

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) – The set you want to retrieve (defaults to `None`, which selects irregardless of set)

• **returnelements** – Return the actual matches within the alternatives, will return two-tuples

Yields AlternativeLayers elements

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

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)

• **returnelements** – Return the actual matches within the alternatives, will return two-tuples where the first is an instance of `Alternative` and the second an instance of Class.

Yields `Alternative` elements

**ancestor**(Classes)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (*AbstractElement* or subclasses).
  Not instances!

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

annotation (**type**, **set=False**)
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 False (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:
* AllowInlineAnnotation.annotations()*
* *AbstractElement.select()*

Raises *NoSuchAnnotation* if no such annotation exists

annotations (**Class**, **set=False**)
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 False (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
- *AllowInlineAnnotation.annotations()*
- *NoSuchAnnotation* if no such annotation exists
annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)  
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)  
See AbstractElement.append()

checkdeclaration ()  
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

count (Class, set=False, recursive=True, ignore=True, node=None)  
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int

checkdeclaration ()  
Perform deep validation of this element.

Raises DeepValidationError

depthfirstsearch (function)  
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False)  
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:
```python
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=False, **kwargs*)
Internal method to find replaceable elements. Auxiliary function used by `AbstractElement.replace()`.

**findspans**(*type*, *set=False, alternatives=False, returnlayers=False*)
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/None/False*) – Constrain by set. Set to False to return regardless of set.
- **alternatives** – Find alternatives only (i.e. if set, this returns only alternatives and not the authoritative annotations, if unset, it does the opposite and returns only the authoritative annotations and no alternatives)
- **returnlayers** – Return the layers in additions to the actual span annotation elements

Example:
```python
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`)
If returnlayers is set, it returns a tuple `(AbstractSpanAnnotation, AbstractAnnotationLayer)`

**generate_id**(*cls*)

**getcorrection**(*set=False, cls=None*)

**getcorrections**(*set=False, 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

1.9. Structure Annotation Types 427
**gettextdelimiter** *(retaintokenisation=False)*

Return the text delimiter for this class.

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

**hasannotation** *(Class, set=False)*

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

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

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

Does the specified annotation layer exist?

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

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, hidden=False)*

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:

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

Returns  dict

layers (annotationtype=None, set=False)
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=False)
Shortcut: returns the FoLiA class of the lemma annotation (will return only one if there are multiple!)

morpheme (index, set=False)
Returns a specific morpheme, the n’th morpheme (given the particular set if specified).

morphemes (set=False)
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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

classmethod parsexml (node, doc, **kwargs)
Internal class method used for turning an XML element into an instance of the Class.

1.9. Structure Annotation Types 429
Parameters

- **node** - XML Element (*
- **doc** - Document (*

Returns An instance of the current Class.

### phon (`cls='current', previousdelimiter=", strict=False, correctionhandling=1, hidden=False`)

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 outputed, 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.
- **hidden** (`bool`) – Include hidden elements, defaults to `False`.

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, hidden=False`)

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 phonetic 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=False)
Returns a specific phoneme, the n’th morpheme (given the particular set if specified).

phonemes (set=False)
Generator yielding all phonemes (in a particular set if specified). For retrieving one specific morpheme by index, use morpheme() instead

pos (set=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards ()
internal helper function for backward compatibility

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.

resolveoffsets (begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

sense (set=False)
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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
correctionhandling
```

**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, hidden=False)*

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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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 (\texttt{index=None})

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

\textbf{Parameters} \texttt{index (\ast )} – If set to an integer, will retrieve and return the n’th element (starting at 0) instead of returning the list of all

\textbf{xml (\texttt{attribs=None, elements=None, skipchildren=False})}

Serialises the FoLiA element and all its contents to XML.

Arguments are mostly for internal use.

\textbf{Returns} an lxml.etree.Element

\textbf{See also:}

\texttt{AbstractElement.xmlstring()} - for direct string output

\textbf{xmlstring (\texttt{pretty_print=False})}

Serialises this FoLiA element and all its contents to XML.

\textbf{Returns} a string with XML representation for this element and all its children

\textbf{Return type} str

\textbf{\_\_iter\_\_()}\n
Iterate over all children of this element.

\textbf{Example:}

\begin{verbatim}
\texttt{for annotation in word:}
  ...
\end{verbatim}

\textbf{\_\_len\_\_()}

Returns the number of child elements under the current element.

\textbf{\_\_str\_\_()}

Alias for \texttt{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.

\section*{1.10 Common attributes}

The FoLiA paradigm features \textit{sets} and \textit{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 provenance, and more. A full list is provided below:

- \texttt{element.id (str)} - The unique identifier of the element
- \texttt{element.set (str)} - The set the element pertains to.
- \texttt{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 \texttt{class} is already a reserved keyword in python, the library consistently uses \texttt{cls} everywhere.
• `element.processor (str)` - The ID of the processor who last added/modified this element. The processor is an instance of `Processor` and is part of the provenance data. It contains information regarding who or what performed the annotation, such as (not exhaustive):
  - `element.processor.id (str)` - the ID of the processor, has to be unique
  - `element.processor.name (str)` - the name of the processor, e.g. the name of a certain software tool or human annotator, needs not be unique
  - `element.processor.type` - the type of processor (e.g. `folia.ProcessorType.MANUAL`, `folia.ProcessorType.AUTO`)

• `element.annotator (str)` - The name or ID of the annotator who last added/modified this element, this is a less extensive mechanism used only if processor is not used.

• `element.annotatortype` - Only if processor is not used: the type of annotator, can be either `folia.AnnotatorType.MANUAL` or `folia.AnnotatorType.AUTO`

• `element.confidence (float)` - A confidence value expressing the confidence the annotator has in this annotation.

• `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 of 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`.

### 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:

```python
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)
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.

### 1.11.1 Inline Annotation Types

The following inline annotation elements are available in FoLiA, they are embedded under a structural element (not necessarily a token, despite the name).

<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DomainAnnotation</strong></td>
<td>Domain annotation: an extended token annotation element</td>
</tr>
<tr>
<td><strong>PosAnnotation</strong></td>
<td>Part-of-Speech annotation: a token annotation element</td>
</tr>
<tr>
<td><strong>LangAnnotation</strong></td>
<td>Language annotation: an extended token annotation element</td>
</tr>
<tr>
<td><strong>LemmaAnnotation</strong></td>
<td>Lemma annotation: a token annotation element</td>
</tr>
<tr>
<td><strong>SenseAnnotation</strong></td>
<td>Sense annotation: a token annotation element</td>
</tr>
<tr>
<td><strong>SubjectivityAnnotation</strong></td>
<td>Subjectivity annotation/Sentiment analysis: a token annotation element</td>
</tr>
</tbody>
</table>

#### folia.main.DomainAnnotation

**class** `folia.main.DomainAnnotation` *(doc, *args, **kwargs)*

Base: `folia.main.AbstractInlineAnnotation`

Domain annotation: an extended token annotation element

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>*(doc, *args, *<em>kwargs)</em></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code><em>(Class[, raiseexceptions, parentinstance]</em>)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code>(child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addidsuffix</code>(idsuffix[, recursive])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>addtoindex</code>([norecurse])</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code>((Class))</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotator2processor</code>([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsecxml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, . . . ])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>phoncontent</strong>([cls, correctionhandling, hidden])**</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend()</strong></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>precedes(other)</strong></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><strong>previous([Class, scope])</strong></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng([includechildren, extraattrs, …])</strong></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><strong>relaxng_backwards()</strong></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><strong>remove(child)</strong></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>**replace(child, *args, <strong>kwargs)</strong></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><strong>resolveoffsets</strong>(begin, end[, …])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><strong>resolveword</strong>(id)</td>
<td></td>
</tr>
<tr>
<td><strong>rightcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><strong>select</strong>(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><strong>setdoc</strong>(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><strong>setdocument</strong>(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><strong>setparents()</strong></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><strong>setprocessor</strong>(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><strong>settext</strong>(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><strong>speech_speaker()</strong></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>speech_src()</strong></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>stricttext</strong>(cls)</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td><strong>text</strong>(cls[, retaintokenisation, …])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>textcontent</strong>(cls, correctionhandling, hidden)</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textvalidation</strong>(warnonly)</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><strong>toktext</strong>(cls)</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td><strong>updatetext()</strong></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><strong>xml</strong>(attribs, elements, skipchildren)</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>xmlstring</strong>(pretty_print)</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong><strong>iter</strong>()</strong></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong><strong>len</strong>()</strong></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong><strong>str</strong>()</strong></td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Description'>, <class 'folia.main.Feature'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Metric'>)
ANNOTATIONTYPE = 12
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

  Example:

  ```python
  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 (tuple of) classes (AbstractElement or subclasses). Not instances!

  Yields elements (instances derived from AbstractElement)

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

    Returns  int
depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None
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
depthfirstsearch (function)
Find the proper correction handling given a textclass by looking in the underlying corrections where it is
reused
classmethod findreplaceables (parent, set=False, **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, hidden=False)
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, hidden=False)**

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:

```python
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`

**parsecommonarguments**(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)

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.

- **retain.tokenisation** *(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 outputed, 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.

- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```
word.phon()
```
Returns The phonetic content of the element (\texttt{unicode} instance in Python 2, \texttt{str} in Python 3)

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

See also:

\texttt{phoncontent()}: Retrieves the phonetic content as an element rather than a string
\texttt{text()} \texttt{textcontent()}

\texttt{phoncontent(\texttt{cls='current'}, \texttt{correctionhandling=1}, \texttt{hidden=False})}

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

Unlike \texttt{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

- \texttt{cls (str)} – The class of the phonetic content to obtain, defaults to \texttt{current}.

- \texttt{correctionhandling} – Specifies what content to retrieve when corrections are encountered. The default is \texttt{CorrectionHandling.CURRENT}, which will retrieve the corrected/current content. You can set this to \texttt{CorrectionHandling.ORIGINAL} if you want the content prior to correction, and \texttt{CorrectionHandling.EITHER} if you don’t care.

Returns The phonetic content (\texttt{PhonContent})

Raises NoSuchPhon if there is no phonetic content for the element

See also:

\texttt{phon()} \texttt{textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes(other)}

Returns a boolean indicating whether this element precedes the other element

\texttt{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

- \texttt{Class (+)} – The class to select; any python class subclassed off \texttt{‘AbstractElement’}, may also be a tuple of multiple classes. Set to \texttt{True} to constrain to the same class as that of the current instance, set to \texttt{None} to not constrain at all.

- \texttt{scope (+)} – A list of classes which are never crossed looking for a next element. Set to \texttt{True} to constrain to a default list of structure elements (Sentence,Paragraph,Division,Event, ListItem,Caption), set to \texttt{None} to not constrain at all.

\texttt{classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)}

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

\texttt{classmethod relaxng_backwards()}

internal helper function for backward compatibility
The documentation for the FoLiA Python library includes several methods for manipulating elements, such as `remove`, `replace`, `resolveoffsets`, and `rightcontext`. Here are brief descriptions of each:

- **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 to be an alternative.

  - **be an alternative.**(to)

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

- **resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')
  Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive.

- **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=False, 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 False (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`, `AlternativeLayers`, `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:

  ```python
    ...
  ```

- **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 ()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

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, hidden=False)

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.
- **hidden (bool)** – Include hidden elements, defaults to False.

**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 ()

folia.main.PosAnnotation

class folia.main.PosAnnotation (doc, *args, **kwargs)

Bases: folia.main.AbstractInlineAnnotation

Part-of-Speech annotation: a token annotation element

Method Summary

__init__ (doc, *args, **kwargs) Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions])
addidsuffix(idsuffix[, recursive])
addtoindex([norecurse])
ancestor(*Classes) Find the most immediate ancestor of the specified type, multiple classes may be specified.
ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
annotator2processor([annotator, ...]) Converts annotator information to processor information (FoLiA v2).
append(child, *args, **kwargs) See AbstractElement.append()
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>
Table 38 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, …])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolvesoffsets(begin, end[, …])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, …])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

1.11. Annotations
Class Attributes

ACCEPTED_DATA = (\<class 'folia.main.Comment'>, \<class 'folia.main.Description'>, \<class 'folia.main.Feature'>, \<class 'folia.main.ForeignData'>, \<class 'folia.main.HeadFeature'>, \<class 'folia.main.Metric'>)
ANNOTATIONTYPE = 10
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = False
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=False, 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, None, or False) – The set
  - **raiseexceptions**(bool) – Raise an exception if the element can’t be added?

Returns bool

Raises ValueError

**addsuffix**(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. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).

Not instances!

Yields elements (instances derived from AbstractElement)

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int
deevalidation()
Perform deep validation of this element.

Raises DeepValidationError
depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None
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:

```python
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=False, **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, hidden=False)
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, hidden=False)**

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:

```python
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`

**parsecommonarguments** *(doc, **kwargs)*
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)*
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 outputed, 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.

- **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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 el-
ement. 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)

classmethod relaxng_backwards ()
internal helper function for backward compatibility
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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'carneto', 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()*

**setprocessor**(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, useful when chaining calls to *text()*.
- **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
- **hidden** (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.
- **hidden** (bool) – Include hidden elements, defaults to False.

**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()

folia.main.LangAnnotation

class folia.main.LangAnnotation(doc, *args, **kwargs)

    Bases: folia.main.AbstractInlineAnnotation

Language annotation: an extended token annotation element

Method Summary

__init__(doc, *args, **kwargs) Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.
addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.
addtoindex([norecurse]) 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.
ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
annotator2processor([annotator, ...]) Converts annotator information to processor information (FoLiA v2).
append(child, *args, **kwargs)

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retainokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorerlist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoncontent</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Description'>, <class 'folia.main.Feature'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Metric'>)
ANNOTATIONTYPE = 34
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
TEXTDELMITER = None
WREFABLE = False
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)

declassmethod addable (parent, set=False, 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, None, or False)` – The `set`

• `raiseexceptions (bool)` – Raise an exception if the element can’t be added?

Returns `bool`

Raises `ValueError`

`addidsuffix (idsuffix, recursive=\text{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. Raise a No-SuchAnnotation exception if not found.

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

Yields `elements` (instances derived from `AbstractElement`) Examples:

```python
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 (tuple of) classes (`AbstractElement` or subclasses). Not instances!

Yields `elements` (instances derived from `AbstractElement`) Examples:

`annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)`

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append ()`

`checkdeclaration ()`

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

`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=\text{None}, idsuffix=\text{”} )`

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 \((newdoc=None, idsuffix=\)"
Generates a deep copy of the children of this element.
Invokes \texttt{copy()} on all children, parameters are the same.

count (Class, set=False, recursive=True, ignore=True, node=None)
Like \texttt{AbstractElement.select()}, but instead of returning the elements, it merely counts them.

\textbf{Returns} \hspace{1em} int

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

description ()
Obtain the description associated with the element.

\textbf{Returns} \hspace{1em} 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.

\textbf{Example}:

\begin{verbatim}
    sense = word.annotation(folia.Sense)
    synset = sense.feat('synset')
\end{verbatim}

\textbf{Returns} \hspace{1em} str or list

findcorrectionhandling (cls)
Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.

classmethod findreplaceables (parent, set=False, **kwargs)
Internal method to find replaceable elements. Auxiliary function used by \texttt{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!

\textbf{Returns} \hspace{1em} 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 \texttt{TEXTDELMITER} attribute but may return a customised one instead.

hasphon (cls='current', strict=True, correctionhandling=1, hidden=False)
Does this element have phonetic content (of the specified class)

By default, and unlike \texttt{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, hidden=False)*

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:

```python
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 UncorrectText.

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

**parsecommonarguments** *(doc, **kwargs)*

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)*

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 beignored, 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 outputed, 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.
- **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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
textcontent() text()

phoncontent (cls='current', correctionhandling=1, hidden=False) 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.

precedes (other) Returns a boolean indicating whether this element precedes the other element

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’, 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.

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)

classmethod relaxng_backwards () internal helper function for backward compatibility
**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
...
```

**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()`

`setprocessor(processor)`
Sets the processor for this element, taking care of adding an annotator in the declarations

`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, hidden=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 outputed, 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
- **`hidden(bool)`** – Include hidden elements, defaults to `False`.

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, hidden=False`)  
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.
- **`hidden(bool)`** – Include hidden elements, defaults to `False`.

**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`)  
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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

folia.main.LemmaAnnotation
class folia.main.LemmaAnnotation(doc, *args, **kwargs)
    Bases: folia.main.AbstractInlineAnnotation
    Lemma annotation: a token annotation element

Method Summary

__init__(doc, *args, **kwargs)
    Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions])
    Tests whether a new element of this class can be added to the parent.
addidsuffix(idsuffix[, recursive])
    Appends a suffix to this element’s ID, and optionally to all child IDs as well.
addtoindex([norecurse])
    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.
ancestors([Class])
    Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
annotator2processor([annotator, ...])
    Converts annotator information to processor information (FoLiA v2).
append(child, *args, **kwargs)
    See AbstractElement.append()

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right.</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoncontent</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, …])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, …])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, …])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Description'>, <class 'folia.main.Feature'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Metric'>)
ANNOTATIONTYPE = 11
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
LABEL = 'Lemma'
OCCURRENCES = 0
OCCURRENCES_PER_SET = 1
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 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
WREFABLE = False
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=False, 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, None, or False)` – 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. Raise a No-SuchAnnotation exception if not found.

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

**Example:**

```python
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 (tuple of) classes (AbstractElement or subclasses).

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

### annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

### checkdeclaration ()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

### 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=False, 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 DeepValidation{Error

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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 correction handling given a text class by looking in the underlying corrections where it is
reused

classmethod findreplaceables (parent, set=False, **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 TEXTDELMITER attribute but may return a customised one instead.

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

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:

```python
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

**parsecommonarguments** (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False*)

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 outputed, 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.
- **hidden** (*bool*) – Include hidden elements, defaults to False.

Example:

```python
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

textcontent() (cls='current', correctionhandling=1, hidden=False)
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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility
**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:**

```python
    ...
```

**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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.

• hidden (bool) – Include hidden elements, defaults to False.

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:

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

__len__( )

Returns the number of child elements under the current element.

__str__( )

Alias for text()
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>checkdeclaration()</strong></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><strong>context(size[, placeholder, scope])</strong></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><strong>copy([newdoc, idsuffix])</strong></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren([newdoc, idsuffix])</strong></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>count(Class[, set, recursive, ignore, node])</strong></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation()</strong></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch(function)</strong></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><strong>description()</strong></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat(subset)</strong></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling(cls)</strong></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables(parent[, set])</strong></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id(cls)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getindex(child[, recursive, ignore])</strong></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata([key])</strong></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter([retaintokenisation])</strong></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasphon([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection()</strong></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>**insert(index, child, *args, <strong>kwargs)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>items([founditems])</strong></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><strong>json([attribs, recurse, ignorelist])</strong></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>leftcontext(size[, placeholder, scope])</strong></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next([Class, scope, reverse])</strong></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext([cls])</strong></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>**parsecommonarguments(doc, <strong>kwargs)</strong></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>**parsexml(node, doc, <strong>kwargs)</strong></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon([cls, previousdelimiter, strict, ...])</strong></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([lattrs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmistring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Description'>, <class 'folia.main.Feature'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Metric'>, <class 'folia.main.SynsetFeature'>)
ANNOTATIONTYPE = 13
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = False
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)

classmethod addable (parent, set=False, 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 overriden by subclasses for more customised behaviour.

   Parameters
      • parent (AbstractElement) – The element that is being added to
• **set**(str, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

**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 (tuple of) classes (AbstractElement or subclasses). Not instances!

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

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append**(child, *args, **kwargs)
See AbstractElement.append()

**checkdeclaration**()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

**depthfirstsearch**(function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

```python
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=False, **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, hidden=False)

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  

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

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  

**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:

```python
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** (*str*) – 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** (*str*) – 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

**parsecommonarguments** (*doc, **kwargs*)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

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

Parameters

- **node** – XML Element (*str*)

- **doc** – Document (*str*)

Returns

An instance of the current Class.

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

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 outputed, 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.

- **hidden** (*bool*) – Include hidden elements, defaults to False.

Example:

```python
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():
phoncontent (cls='current', correctionhandling=1, hidden=False)
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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility
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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:
    ...

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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, hidden=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 outputed, 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

- **hidden**(bool) – Include hidden elements, defaults to False.

Example:
```python
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, hidden=False)

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.

- **hidden**(bool) – Include hidden elements, defaults to False.

**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)

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()
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>checkdeclaration()</strong></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><strong>context(size[, placeholder, scope])</strong></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><strong>copy([newdoc, idsuffix])</strong></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren([newdoc, idsuffix])</strong></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>count(Class[, set, recursive, ignore, node])</strong></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation()</strong></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch(function)</strong></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description()</strong></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat(subset)</strong></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling(cls)</strong></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables(parent[, set])</strong></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id(cls)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getindex(child[, recursive, ignore])</strong></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata([key])</strong></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><strong>gettextdelimiter([retaintokenisation])</strong></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasphon([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><strong>hastext([cls, strict, correctionhandling, ...])</strong></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><strong>incorrection()</strong></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td>**insert(index, child, *args, <strong>kwargs)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>items([founditems])</strong></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><strong>json([attribs, recurse, ignorelist])</strong></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>leftcontext(size[, placeholder, scope])</strong></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next([Class, scope, reverse])</strong></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext([cls])</strong></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>**parsecommonarguments(doc, <strong>kwargs)</strong></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>**parsexml(node, doc, <strong>kwargs)</strong></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon([cls, previousdelimiter, strict, ...])</strong></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>
Table 42 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, …])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, …])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True.</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, …])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True.</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attrs, elements, skipchildren])</td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text().</td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (\texttt{\textless class 'folia.main.Comment'\textgreater}, \texttt{\textless class 'folia.main.Description'\textgreater}, \texttt{\textless class 'folia.main.Feature'\textgreater}, \texttt{\textless class 'folia.main.ForeignData'\textgreater}, \texttt{\textless class 'folia.main.Metric'\textgreater})

ANNOTATIONTYPE = 20

AUTH = True

AUTO_GENERATE_ID = False

HIDDEN = False

LABEL = 'Subjectivity/Sentiment'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 1

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 5, 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

WREFABLE = False

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=False, 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, None, or False)` – The set

• `raiseexceptions(booll)` – 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. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (`AbstractElement` or subclasses).

Not instances!

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

`annotator2processor(annotator=None, annotatortype=None, parentprocessor=None)`

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`

`checkdeclaration()`

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

`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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

description ()
Obtain the description associated with the element.

Returns 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=False, **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, hidden=False)
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, hidden=False)

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:

```python
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

parsecommonarguments (**doc**, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, **hidden**=False)

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.
- **retain_tokenisation** (**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 outputed, 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.
- **hidden** (**bool**) – Include hidden elements, defaults to False.

Example:

```python
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():
phoncontent (cls='current', correctionhandling=1, hidden=False)
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()
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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
   Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:
      ...

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()`

`setprocessor` (*processor*)

Sets the processor for this element, taking care of adding an annotator in the declarations

`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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```py
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, hidden=False)*

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.

• **hidden** *(bool)* – Include hidden elements, defaults to False.

**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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

### 1.11.2 Text and phonetic annotation

The actual text of an element, or a phonetic textual representation, are also considered annotations themselves.

| TextContent | Text content element (t), holds text to be associated with whatever element the text content element is a child of. |
| PhonContent | Phonetic content element (ph), holds a phonetic representation to be associated with whatever element the phonetic content element is a child of. |

**folia.main.TextContent**

class folia.main.TextContent(doc, *args, **kwargs)
Bases: folia.main.AbstractContentAnnotation

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Example.</td>
</tr>
<tr>
<td>accepts (Class, raiseexceptions, parentinstance)</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element's ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex ([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors ([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotator2processor ([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append (child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context (size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td>copy ([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren ([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>count (Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch (function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td>description ()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat (subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling (cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>finddefaultreference ()</td>
<td>Find the default reference for text offsets: The parent of the current textcontent’s parent (counting only Structure Elements and Subtoken Annotation Elements)</td>
</tr>
<tr>
<td>findreplaceables (parent, set, **kwargs)</td>
<td>(Method for internal usage, see AbstractElement)</td>
</tr>
<tr>
<td>getindex (child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata ([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>getreference ([validate])</td>
<td>Returns and validates the Text Content’s reference.</td>
</tr>
<tr>
<td>gettextdelimiter ([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hasphon</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hasText</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>See AbstractElement.json()</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text)</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([normalize_spaces])</code></td>
<td>Obtain the text (unicode instance)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

```
ACCEPTED_DATA = (class 'folia.main.AbstractTextMarkup'>, class 'folia.main.Comment'>)
ANNOTATIONTYPE = 0
AUTH = True
AUTOGENERATE_ID = False
HIDDEN = False
LABEL = 'Text'
OCURRENCES = 0
OCURRENCES_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
TEXTEXPLIMITER = None
WREFABLE = False
XLINK = True
```
Method Details

__init__ (doc, *args, **kwargs)

Example:

```python
text = folia.TextContent(doc, 'test')
text = folia.TextContent(doc, 'test', cls='original')
```

__init__ (doc, *args, **kwargs)

Example:

```python
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (`AbstractElement` or subclasses). Not instances!

Yields elements (instances derived from `AbstractElement`)

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

**checkdeclaration** *

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

**depthfirstsearch** *(function)*

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

```python
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

getrerference(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, hidden=False)
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, hidden=False)
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`

**parsecommonarguments(doc, **kwargs)**

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

(Method for internal usage, see `AbstractElement`)

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

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`. 

516 Chapter 1. Reading FoLiA
• **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 outputed, 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.

• **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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

`textcontent()`

`phoncontent (cls='current', correctionhandling=1, hidden=False)`

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 phonetic 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.
**precedes** *(other)*

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

**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)

**classmethod relaxng_backwards()**

Internal helper function for backward compatibility

**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.

**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:

```python
```

`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()`

`setprocessor(processor)`
Sets the processor for this element, taking care of adding an annotator in the declarations

`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, hidden=False)  
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.
• hidden (bool) – Include hidden elements, defaults to False.

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:

```python
for annotation in word:
    ...
```
__len__ ()
Returns the number of child elements under the current element.

__str__ ()
Alias for text ()

**folia.main.PhonContent**

class folia.main.PhonContent (doc, *args, **kwargs)
Bases: folia.main.AbstractContentAnnotation

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**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Example.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td>finddefaultreference()</td>
<td>Find the default reference for text offsets: The parent of the current textcontent’s parent (counting only Structure Elements and Subtoken Annotation Elements).</td>
</tr>
<tr>
<td>findreplaceables(parent, set, **kwargs)</td>
<td>(Method for internal usage, see AbstractElement)</td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursively by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td>getreference([validate])</td>
<td>Return and validate the Phonetic Content’s reference.</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)?</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)?</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flattened list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element.</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>(Method for internal usage, see AbstractElement)</td>
</tr>
<tr>
<td>phon()</td>
<td>Obtain the actual phonetic representation (unicode/str instance).</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 45 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setphon(phon)</code></td>
<td>Set the representation for the phonetic content (unicode instance), called whenever phon= is passed as a keyword argument to an element constructor</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>Serialises the FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

```
ACCEPTED_DATA = ({<class 'folia.main.Comment'>, <class 'folia.main.Description'>})
```
ANNOTATIONTYPE = 19
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
TEXTDELMITER = None
WREFABLE = False
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)
adchild, *args, **kwargs)

classmethod addable (parent, set=False, 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, None, or False*) – 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. *Raise a No-SuchAnnotation exception if not found.*

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

Example:

```python
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 (tuple of) classes (*AbstractElement* or subclasses).

Not instances!

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

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

**checkdeclaration** ()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

**depthfirstsearch**(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

```python
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)
Return and validate the Phonetic Content’s reference. Raises UnresolvableTextContent when invalid
**gettextdelimiter** *(retaintokenisation=False)*
Return the text delimiter for this class.

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

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

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

1.11. Annotations 527
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 el-
  ement. 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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

classtype parsexml (node, doc, **kwargs)
(Method for internal usage, see AbstractElement)

phon ()
Obtain the actual phonetic representation (unicode/str instance)

phoncontent (cls='current', correctionhandling=1, hidden=False)
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 en-
  countered. 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.

**precedes** *(other)*
Returns a boolean indicating whether this element precedes the other element

**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’, 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.

**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)

**classmethod relaxng_backwards** *
internal helper function for backward compatibility

**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=False, 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 False (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, AlternativeLayers, 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:

```python
  ...
```

**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

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.
• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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.

**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:**

```python
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.textcontent()`, or as element using `AbstractElement.phon()`. 
1.11.3 Span Annotation

FoLiA distinguishes inline annotation and span annotation, inline 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:

```python
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:

```python
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 use the AbstractAnnotationLayer.findspan method, available on all annotation layers:

```python
for span in annotationlayer.findspan(word1, word2):
    print("Class: ", span.cls)
    print("Text: ", span.text()) #same for every span here
```

1.11.4 Span Annotation Types

This section lists the available Span annotation elements, the layer that contains them is explicitly mentioned as well.

---

**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 OCRed text for instance) or with corrections of orthography or phonetics.
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:

<table>
<thead>
<tr>
<th>Annotation Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chunk</strong></td>
<td>Chunk element, span annotation element to be used in <em>ChunkingLayer</em></td>
</tr>
<tr>
<td><strong>CoreferenceChain</strong></td>
<td>Coreference chain.</td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td>Span annotation element to encode dependency relations</td>
</tr>
<tr>
<td><strong>Entity</strong></td>
<td>Entity element, for entities such as named entities, multi-word expressions, temporal entities.</td>
</tr>
<tr>
<td><strong>Observation</strong></td>
<td>Observation.</td>
</tr>
<tr>
<td><strong>Predicate</strong></td>
<td>Predicate, used within <em>SemanticRolesLayer</em>, takes <em>SemanticRole</em> annotations as children, but has its own annotation type and separate declaration</td>
</tr>
<tr>
<td><strong>Sentiment</strong></td>
<td>Sentiment.</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td>Statement.</td>
</tr>
<tr>
<td><strong>SyntacticUnit</strong></td>
<td>Syntactic Unit, span annotation element to be used in <em>SyntaxLayer</em></td>
</tr>
<tr>
<td><strong>SemanticRole</strong></td>
<td>Semantic Role</td>
</tr>
<tr>
<td><strong>TimeSegment</strong></td>
<td>A time segment</td>
</tr>
</tbody>
</table>

**fooia.main.Chunk**

class foia.main.Chunk(doc, *args, **kwargs)

Bases: foia.main.AbstractSpanAnnotation

Chunk element, span annotation element to be used in *ChunkingLayer*

**Method Summary**

- __init__(doc, *args, **kwargs) Initialize self.
- accepts(Class[, raiseexceptions, parentinstance])
- add(child, *args, **kwargs)
- addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.
- addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.
- addtoindex([norecurse]) 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.
- ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
- annotation(type[, set]) Will return a single annotation (even if there are multiple).
- annotations(Class[, set]) Obtain annotations.
- annotator2processor([annotator, ...]) Converts annotator information to processor information (FoLiA v2).

Continued on next page
### Table 47 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append(child, *args, **kwargs)</code></td>
<td>See <code>AbstractElement.append()</code></td>
</tr>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, <code>{size}</code> words to the left, the current word, and <code>{size}</code> words to the right.</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . . ])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to <code>True</code>), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of <code>all</code> items below this element (not limited to <code>AbstractElement</code>)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 47 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Replaces a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scope.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code>.</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code>.</td>
</tr>
</tbody>
</table>
Table 47 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>wrefs(index, recurse)</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ...
ANNOTATIONTYPE = 15
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = 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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

cHECKDeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

dEPTHFIRSTsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **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=False)
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, hidden=False)
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, hidden=False)
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)***

    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:

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

    Returns dict

    **layer()**
    Return the annotation layer this annotation pertains to

    **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

    **parsecommonarguments(doc, **kwargs)**
    Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

    **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**

```python
(cls='current', previousdelimiter=' ', strict=False, correctionhandling=1, hidden=False)
```

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 outputed, 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.
- **hidden** *(bool)* – Include hidden elements, defaults to *False*.

**Example:**

```python
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()**: Retrieves the phonetic content as an element rather than a string
- **textcontent()**: Retrieves the phonetic content as an element rather than a string

**phoncontent**

```python
(cls='current', correctionhandling=1, hidden=False)
```

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 \texttt{CorrectionHandling.EITHER} if you don’t care.

\textbf{Returns} \ The phonetic content (\texttt{PhonContent})

\textbf{Raises} \ \texttt{NoSuchPhon} if there is no phonetic content for the element

\textbf{See also:} \ \texttt{phon()} \texttt{textContent()} \texttt{text()}

\textbf{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.

\textbf{precedes (other) }

Returns a boolean indicating whether this element precedes the other element

\textbf{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.

\textbf{Parameters}

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

\textbf{classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None) }

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

\textbf{classmethod relaxng_backwards()}

internal helper function for backward compatibility

\textbf{remove (child) }

Removes the child element

\textbf{replace (child, *args, **kwargs) }

Appends a child element like \texttt{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 \texttt{append()}

\textbf{Keyword Arguments}

\begin{itemize}
  \item \texttt{alternative (bool)} – If set to True, the replaced element will be made into an alternative. Simply use \texttt{AbstractElement.append()} if you want the added element
  \item \texttt{be an alternative. (to)} –
\end{itemize}

See \texttt{AbstractElement.append()} for more information and all parameters.

\textbf{resolveoffsets (begin, end, retaintokenisation=True, strictend=True, cls=’current’) }

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

\textbf{resolveword (id) }
rightcontext \((size, \text{placeholder}=None, \text{scope}=None)\)

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 \((\text{Class}, \text{set}=False, \text{recursive}=True, \text{ignore}=True, \text{node}=None)\)

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 False (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`, `AlternativeLayers`, `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:

```python
```

setdoc \((\text{newdoc})\)

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

setdocument \((\text{doc})\)

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()`

setprocessor \((\text{processor})\)

Sets the processor for this element, taking care of adding an annotator in the declarations

setspan \((*\text{args})\)

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

**Parameters**

- ***\text{args}** – Instances of `Word`, `Morpheme` or `Phoneme`

settext \((\text{text}, \text{cls}='current')\)

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.

**sort (force=False)**

Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

**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, hidden=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 outputed, 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

• **hidden (bool)** – Include hidden elements, defaults to `False`.

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, hidden=False)
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.
- **hidden** (bool) – Include hidden elements, defaults to False.

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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

**Folia.main.CoreferenceChain**

```python
class Folia.main.CoreferenceChain(doc, *args, **kwargs)
Bases: Folia.main.AbstractSpanAnnotation
Coreference chain. Holds CoreferenceLink instances.
```

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>(idsuffix[, recursive])</td>
</tr>
<tr>
<td>addtoindex</td>
<td>([norecurse])</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes)</td>
</tr>
<tr>
<td>ancestors</td>
<td>([Class])</td>
</tr>
<tr>
<td>annotation</td>
<td>(type[, set])</td>
</tr>
<tr>
<td>annotations</td>
<td>(Class[, set])</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>([annotator, ...])</td>
</tr>
<tr>
<td>append</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>(size[, placeholder, scope])</td>
</tr>
<tr>
<td>copy</td>
<td>([newdoc, idsuffix])</td>
</tr>
<tr>
<td>copychildren</td>
<td>([newdoc, idsuffix])</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>correct(<strong>kwargs</strong>)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td></td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layer()</td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>precedes(\text{other})</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([\text{Class, scope}])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([\text{includechildren, extraattrs, ...}])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(\text{child})</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(\text{child}, *args, **kwargs)</td>
<td>Appends a child element like \text{append()}, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(\text{id})</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select([\text{Class[, set, recursive, ignore, node]}])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(\text{processor})</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>setspan(*args)</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td>settext(\text{text[, cls]})</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>sort([\text{force}])</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([\text{cls}])</td>
<td>Alias for \text{text()} with \text{strict=True}</td>
</tr>
<tr>
<td>text([\text{cls, retaintokenisation, ...}])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([\text{cls, correctionhandling, hidden}])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([\text{warnonly}])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([\text{cls}])</td>
<td>Alias for \text{text()} with \text{retaintokenisation=True}</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>wrefs([\text{index, recurse}])</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td>xml([\text{attribs, elements, skipchildren}])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td>xmlstring([\text{pretty_print}])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td>\texttt{<strong>iter</strong>}()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td>\texttt{<strong>len</strong>}()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 48 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

#### Class Attributes

- ACCEPTED_DATA = (&lt;class 'folia.main.AbstractInlineAnnotation'&gt;, &lt;class 'folia.main.Comment'&gt;, &lt;class 'folia.main.LinkReference'&gt;)
- ANNOTATIONTYPE = 31
- AUTH = True
- AUTO_GENERATE_ID = False
- HIDDEN = 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 = (&lt;class 'folia.main.CoreferenceLink'&gt;)
- SETONLY = False
- SPEAKABLE = True
- SUBSET = None
- TEXTCONTAINER = False
- TEXTDELMITER = None
- WREFABLE = False
- 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)**

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, None, or False*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can’t be added?

Returns  *bool*

Raises  *ValueError*

```python
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()`

```python
addtoindex (norecurse=None)
```

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

```python
ancestor (*Classes*)
```

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a *NoSuchAnnotation* exception if not found.

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

Example:

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

```python
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 (tuple of) classes (*AbstractElement* or subclasses). Not instances!

Yields  elements (instances derived from *AbstractElement*)

```python
annotation (type, set=False)
```

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

```python
annotations (Class, set=False)
```

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.

```python
annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
```

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

```python
append (child, *args, **kwargs)
```

See `AbstractElement.append()`
checkdeclaration()  
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

certainty (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=False, 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

depthfirstsearch (function)  
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **kwargs)
Internal method to find replaceable elements. Auxiliary function used by AbstractElement. replace(). Can be overriden 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=False)
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, hidden=False)
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, hidden=False)
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:

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

**Returns** dict

**layer()**

Return the annotation layer this annotation pertains to.

**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

**parsecommonarguments**(doc, **kwargs)**

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)**

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 outputed, useful when chaining calls to *phon()*.
- **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.
- **hidden** *(bool)* – Include hidden elements, defaults to *False*.

**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, hidden=False)

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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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 el-
  ement. 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, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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 alter-
  native. Simply use AbstractElement.append() if you want the added element

• **be an alternative.**(to)

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

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
```

**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()*

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** *(force=False)*

Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage
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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.
- `hidden` *(bool)* – Include hidden elements, defaults to False.

**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:

```python
for annotation in word:
    ...
```
__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

**folia.main.Dependency**

class folia.main.Dependency (doc, *args, **kwargs)
Bases: folia.main.AbstractSpanAnnotation

Span annotation element to encode dependency relations

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts (Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex ([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors ([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation (type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations (Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor ([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append (child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context (size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy ([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren ([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct (**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count (Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>dependent ()</td>
<td>Returns the dependent of the dependency relation.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>depthfirstsearch</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a text class by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><code>head()</code></td>
<td>Returns the head of the dependency relation.</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>Return the annotation layer this annotation pertains to.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previoudelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
</tbody>
</table>
### Table 49 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>relaxng</code>([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove</code>(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace</code>(child, *args, **kwargs)</td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets</code>(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code>(id)</td>
<td></td>
</tr>
<tr>
<td><code>rightcontext</code>(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code>(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc</code>(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code>(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents</code>()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor</code>(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan</code>(*args)</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext</code>(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort</code>([force])</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext</code>([cls])</td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text</code>([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent</code>([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation</code>([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext</code>([cls])</td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext</code>()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs</code>([index, recurse])</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml</code>([attribs, elements, skipchildren])</td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring</code>([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__</code>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__</code>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__</code>()</td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ...)
```
ANNOTATIONTYPE = 23
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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 'folia.main.DependencyDependent'>, <class 'folia.main.Headspan'>)
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = 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=False, 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, None, or False) – The set
• raiseexceptions (bool) – Raise an exception if the element can’t be added?

1.11. Annotations
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. Raise a No-SuchAnnotation exception if not found.

Parameters

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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 appended 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=False, 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*

**depthfirstsearch** (*function*)  
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

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

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

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

Returns  dict

layer()  
Return the annotation layer this annotation pertains to

leftcontext  
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  
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** (optional) – 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** (optional) – 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  
Alias for retrieving the original uncorrect text.

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

parsecommonarguments  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

Parameters

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

Returns  An instance of the current Class.

phon  
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 outputed, 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.

• **hidden (bool)** – Include hidden elements, defaults to False.

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

`textcontent()`

`phoncontent (cls='current', correctionhandling=1, hidden=False)`

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:**

```python
    ..
```

**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()`

**setprocessor** (*processor*)

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** (*force=False*)

Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

**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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

**Example:**

```python
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, hidden=False)
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.

- **hidden (bool)** – Include hidden elements, defaults to `False`.

**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()`
**Folia.main.Entity**

**class** Folia.main.Entity (doc, *args, **kwargs)  
**Bases:** Folia.main.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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts (Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex ([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors ([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation (type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations (Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor ([annotator, ...])</td>
<td>Converts annotator information to processor information (Folia v2).</td>
</tr>
<tr>
<td>append (child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context (size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy ([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren (newdoc, idsuffix)</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct (**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count (Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch (function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description ()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat (subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td></td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layer()</td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattribs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element ( lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
</tbody>
</table>
### Table 50 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs(index, recurse)&quot;</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])&quot;</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

#### Class Attributes

- `ANNOTATIONTYPE` = 16
- `AUTH` = True
- `AUTO_GENERATE_ID` = False

1.11. Annotations 575
HIDDEN = 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
TEXTDELMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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\(\) \((\star Classes)\)

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a NoSuchAnnotation exception if not found.

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

Example:

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

ancestors\(\) \((\text{Class=None})\)

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\(\star\) – The class or (tuple of) classes (AbstractElement or subclasses).

Not instances!

Yields elements (instances derived from AbstractElement)

annotation\(\) \((\text{type, set=False})\)

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

annotations\(\) \((\text{Class, set=False})\)

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

Parameters

\begin{itemize}
  \item \Class \– The Class you want to retrieve\(\star\)–
  \item \set \– The set you want to retrieve\(\star\)–
\end{itemize}

Yields elements

Raises NoSuchAnnotation if the specified annotation does not exist.

annotator2processor\(\) \((\text{annotator=None, annotatortype=None, parentprocessor=None})\)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append\(\) \((\text{child, *args, **kwargs})\)

See AbstractElement.append\(\)

checkdeclaration\(\)

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context\(\) \((\text{size, placeholder=None, scope=None})\)

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

copy\(\) \((\text{newdoc=None, idsuffix=\"\")\)

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

Parameters

\begin{itemize}
  \item \newdoc \text{(Document)} \– The document the copy should be associated with.
\end{itemize}
• **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=False, 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

**depthfirstsearch**(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

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

**Returns** str or list

**findcorrectionhandling**(cls)
Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused

**classmethod findreplaceables**(parent, set=False, **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 TEXTDELMITER attribute but may return a customised one instead.

hasannotation (Class, set=False)
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, hidden=False)
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, hidden=False)
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

layer()
Return the annotation layer this annotation pertains to

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 el-
  ement. 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

parsecommonarguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

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, hidden=False)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever possible, as they are more spe-
cific. 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 possi-
  ble. Defaults to False.
• **previousdelimiter** (*str*) – Can be set to a delimiter that was last outputed, 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.

• **hidden** (*bool*) – Include hidden elements, defaults to `False`.

Example:

```python
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:

```python
phoncontent(): Retrieves the phonetic content as an element rather than a string
text():
phoncontent (cls='current', correctionhandling=1, hidden=False)
```

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:

```python
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.

`precedes (other)`

Returns a boolean indicating whether this element precedes the other element
**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** (*str*) – 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** (*str*) – 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, origclass=None*)

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

**classmethod relaxng_backwards**()

Internal helper function for backward compatibility

**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.

**resolveoffsets** (*begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:

```python
...
```

**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()`

**setprocessor** (`processor`)
Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** (`force=False`)
Sort children (wrefs and child spans) in order of appearance. Returns `True` if sort is successful (or not needed), `False` if sort could not be performed at this stage

**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, hidden=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 outputed, 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
- `hidden (bool)` – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)`

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.
• **hidden** *(bool)* – Include hidden elements, defaults to False.

**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:

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

**__len__()**

Returns the number of child elements under the current element.

**__str__()**

Alias for text()
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accept *(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add *(child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addable *(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix *(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex *(norecurse)</td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td>ancestor *(Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors *(Clas)</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation *(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations *(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor *(annotator, …)</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append *(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context *(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td>copy *(newdoc, idsuffix)</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren *(newdoc, idsuffix)</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct *(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still).</td>
</tr>
<tr>
<td>count *(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch *(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td>description ()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat *(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling *(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td>findreplaceables *(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id *(cls)</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>getindex *(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata *[key)]</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td>gettextdelimiter *(retaintokenisation)</td>
<td>Return the text delimiter for this class.</td>
</tr>
</tbody>
</table>
Table 51 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling,...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling,...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict,...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs,...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 51 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmistring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

**Class Attributes**

```python
ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ...
ANNOTATIONTYPE = 46
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
LABEL = 'Observation'
OCURRENCES = 0
OCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11)
PHONCONTAINER = False
PRIMARYELEMENT = True
PRINTABLE = True
```
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELMITIER = None
WREFABLE = False
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=False, raiseexceptions=True)
Tests whether a new element of this class can be added to the parent.

Parameters
  • parent (AbstractElement) – The element that is being added to
  • set (str, None, or False) – 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. Raise a No-
SuchAnnotation exception if not found.

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

Example:

1.11. Annotations
paragraph = word.ancestor(folia.Paragraph)

Canons (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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

Annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

Annotations (Class, set=False)
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.

Annotator2Processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

Append (child, *args, **kwargs)
See AbstractElement.append()

CheckDeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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:
```python
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=False, **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=False)
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, hidden=False)
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, hidden=False)`

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:

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

**Returns** dict

`layer()`

Return the annotation layer this annotation pertains to

`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** (*str*) – 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** (*str*) – 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

**parsecommonarguments** (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

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

Parameters

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

Returns An instance of the current Class.

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

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 outputed, 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.
- **hidden** (*bool*) – Include hidden elements, defaults to False.

Example:

```python
word.phon()
```
Returns  The phonetic content of the element (\texttt{unicode} instance in Python 2, \texttt{str} in Python 3)

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

See also:

\texttt{phoncontent()}: Retrieves the phonetic content as an element rather than a string
\texttt{text()} \texttt{textcontent()}

\texttt{phoncontent(cls='current', correctionhandling=1, hidden=False)}

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

Unlike \texttt{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

- \texttt{cls (str)} – The class of the phonetic content to obtain, defaults to \texttt{current}.

- \texttt{correctionhandling} – Specifies what content to retrieve when corrections are encountered. The default is \texttt{CorrectionHandling.CURRENT}, which will retrieve the corrected/current content. You can set this to \texttt{CorrectionHandling.ORIGINAL} if you want the content prior to correction, and \texttt{CorrectionHandling.EITHER} if you don’t care.

Returns  The phonetic content (\texttt{PhonContent})

Raises  NoSuchPhon if there is no phonetic content for the element

See also:

\texttt{phon() textcontent()} \texttt{text()}

\texttt{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.

\texttt{precedes(other)}

Returns a boolean indicating whether this element precedes the other element

\texttt{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

- \texttt{Class (\texttt{\ast})} – The class to select; any python class subclassed off \texttt{‘AbstractElement’}, may also be a tuple of multiple classes. Set to \texttt{True} to constrain to the same class as that of the current instance, set to \texttt{None} to not constrain at all

- \texttt{scope (\texttt{\ast})} – A list of classes which are never crossed looking for a next element. Set to \texttt{True} to constrain to a default list of structure elements (Sentence, Paragraph, Division, Event, ListItem, Caption), set to \texttt{None} to not constrain at all

\texttt{classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, orig-class=None)}

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

\texttt{classmethod relaxng_backwards()}

internal helper function for backward compatibility
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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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.

sort (force=False)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

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, previoustiter=",", strict=False, correctionhandling=1, normalize_spaces=False, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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, hidden=False)*
```

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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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** boolean

```
**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()

class folia.main.Predicate(doc, *args, **kwargs)
    Bases: folia.main.AbstractSpanAnnotation

    Predicate, used within SemanticRolesLayer, takes SemanticRole annotations as children, but has its own annotation type and separate declaration

Method Summary

    __init__(doc, *args, **kwargs) Initialize self.
    accepts(Class[, raiseexceptions, parentinstance])
    add(child, *args, **kwargs)
    addable(parent[, set, raiseexceptions])
    addidsuffix(idsuffix[, recursive])
    addtoindex([norecurse])
    ancestor(*Classes)
    Continues on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ancestors((Class,))</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation(type[, set])</code></td>
<td>Will return a <code>single</code> annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations((Class[, set]))</code></td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor([annotator, ...])</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append(child, *args, **kwargs)</code></td>
<td>See <code>AbstractElement.append()</code></td>
</tr>
<tr>
<td><code>checkdeclaration()</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context(size[, placeholder, scope])</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><code>copy([newdoc, idsuffix])</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation((Class[, set]))</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon((cls, strict, correctionhandling, ...))</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext((cls, strict, correctionhandling, ...))</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 52 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer()</td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td>leftcontext()</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next()</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext()</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments()</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml()</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon()</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent()</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes()</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous()</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng()</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove()</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace()</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets()</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword()</td>
<td></td>
</tr>
<tr>
<td>rightcontext()</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select()</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc()</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument()</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor()</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>setspan()</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td>settext()</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>sort()</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext()</td>
<td>Alias for text() with strict=True</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 52 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmistring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

#### Class Attributes

- **ACCEPTED_DATA** = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, <class 'folia.main.Entity'>, <class 'folia.main.Epithet'>, <class 'folia.main.Role'>, <class 'folia.main.SemanticRole'>, <class 'folia.main.WordReference'>)
- **ANNOTATIONTYPE** = 45
- **AUTH** = True
- **AUTO_GENERATE_ID** = False
- **HIDDEN** = False
- **LABEL** = 'Predicate'
- **OCCURRENCES** = 0
- **OCCURRENCES_PER_SET** = 0
- **OPTIONAL_ATTRIBS** = (0, 1, 2, 4, 5, 6, 7, 8, 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
- **WREFABLE** = False
- **XLINK** = False

1.11. Annotations
XMLTAG = 'predicate'

**Method Details**

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

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

```python
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
```

```python
def add(child, *args, **kwargs)
```

```python
classmethod addable(parent, set=False, 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, None, or False*) – The set
- **raiseexceptions** (*bool*) – Raise an exception if the element can’t be added?

**Returns** *bool*

**Raises** *ValueError*

```python
def 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()`

```python
def addtoindex(norecurse=None)
```

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

```python
def ancestor(*Classes)
```

Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:

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

```python
def 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 (tuple of) classes (*AbstractElement* or subclasses).

Not instances!

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

```python
def annotation(type, set=False)
```

Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found
annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append(child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **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=False)
    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, hidden=False)
    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, hidden=False)
    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:

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

**Returns** `dict`

`layer()`

Return the annotation layer this annotation pertains to

`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`
parsecommonarguments \((doc, **kwargs)\)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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 \((cls='current', previousdelimiter=", strict=False, correctionhandling=1, hidden=False)\)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden** \((bool)\) – Include hidden elements, defaults to False.

Example:

```
word.phon()
```

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

Raises NoSuchPhon – if no phonetic conent is found at all.

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

phoncontent \((cls='current', correctionhandling=1, hidden=False)\)
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.

### precedes(*other*)

Returns a boolean indicating whether this element precedes the other element

### 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’, 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.

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

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

### classmethod relaxng_backwards()

internal helper function for backward compatibility

### 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.
**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'cornetto', True, [folia.Original, ...
```

**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()*

**setprocessor**(processor)

Sets the processor for this element, taking care of adding an annotator in the declarations.

**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.

sort\((force=False)\)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage.

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\((cls='current', retaintokenisation=False, previousdelimiter='', strict=False, correctionhandling=1, normalize_spaces=False, hidden=False)\)
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 outputed, 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
- **hidden** (bool) – Include hidden elements, defaults to False.
Example:

```python
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, hidden=False)

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.
- **hidden (bool)** – Include hidden elements, defaults to `False`.

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:

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

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

class folia.main.Sentiment(doc, *args, **kwargs)
    Bases: folia.main.AbstractSpanAnnotation

Sentiment. Takes span roles Headspan, Source and Target as children

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>Accepts (Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>(idsuffix[, recursive])</td>
</tr>
<tr>
<td>addtoindex</td>
<td>([norecurse])</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes)</td>
</tr>
<tr>
<td>ancestors</td>
<td>([Class])</td>
</tr>
<tr>
<td>annotation</td>
<td>(type[, set])</td>
</tr>
<tr>
<td>annotations</td>
<td>(Class[, set])</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>([annotator, ...])</td>
</tr>
<tr>
<td>append</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td>[Internal method (usually no need to call this) that checks whether the element's annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.]</td>
</tr>
<tr>
<td>context</td>
<td>(size[, placeholder, scope])</td>
</tr>
<tr>
<td>copy</td>
<td>([newdoc, idsuffix])</td>
</tr>
<tr>
<td>copychildren</td>
<td>([newdoc, idsuffix])</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>Return the annotation layer this annotation pertains to.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
</tbody>
</table>
Table 53 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precedes</strong></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><strong>previous</strong> ([class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng</strong> ([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><strong>relaxng_backwards()</strong></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><strong>remove</strong> (child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td><strong>replace</strong> (child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><strong>resolveoffsets</strong> (begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><strong>resolveword</strong> (id)</td>
<td></td>
</tr>
<tr>
<td><strong>rightcontext</strong> (size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><strong>select</strong> (class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><strong>setdoc</strong> (newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><strong>setdocument</strong> (doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><strong>setparents</strong> ()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><strong>setprocessor</strong> (processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><strong>setspan</strong> (*args)</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><strong>settext</strong> (text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><strong>sort</strong> ([force])</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><strong>speech_speaker</strong> ()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>speech_src</strong> ()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><strong>stricttext</strong> ([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td><strong>text</strong> ([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textcontent</strong> ([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>textvalidation</strong> ([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><strong>toktext</strong> ([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td><strong>updatetext</strong> ()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><strong>wrefs</strong> ([index, recurse])</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><strong>xml</strong> ([attribs, elements, skipchildren])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td><strong>xmlstring</strong> ([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 53 – continued from previous page

__str__() Alias for text()
Parameters

- **parent** *(AbstractElement)* – The element that is being added to
- **set** *(str, None, or False)* – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes *(AbstractElement* or subclasses). Not instances!

Yields elements (instances derived from *AbstractElement*)

annotation *(type, set=False)*

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

annotations *(Class, set=False)*

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.

annotator2processor *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See *AbstractElement.append()*
checkdeclaration()  
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

custom (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=False, 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

depthfirstsearch (function)  
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

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

Returns str or list

findcorrectionhandling (cls)  
Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused
classmethod findreplaceables (parent, set=False, **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

gettextrdelimiter (retaintokenisation=False)
Return the text delimiter for this class.

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

hasannotation (Class, set=False)
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, hidden=False)
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, hidden=False)
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:

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

Returns dict

layer()  
Return the annotation layer this annotation pertains to.

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

parsecommonarguments(doc, **kwargs)  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)  
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*.
- **reotaintokenisation** *(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 outputed, 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.
- **hidden** *(bool)* – Include hidden elements, defaults to *False*.

Example:

```python
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, hidden=False)*

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.

precedes(other)
Returns a boolean indicating whether this element precedes the other element

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’, 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 el-
  ement. 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, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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 alter-
  native. Simply use AbstractElement.append() if you want the added element

• be an alternative. (to)

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

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
for sense in text.select(folia.Sense, 'carnotto', 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 ()*

**setprocessor** (*processor*)
Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** (*force=False*)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

1.11. Annotations
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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)
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.
- **hidden (bool)** – Include hidden elements, defaults to `False`.

**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:**

  ```python
  for annotation in word:
      ...
  ```
__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for text()
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>depthfirstsearch</strong></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong></td>
<td>Find the proper correction handling given a text class by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables</strong></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getindex</strong></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasphon</strong></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext</strong></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection</strong></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><strong>insert</strong></td>
<td></td>
</tr>
<tr>
<td><strong>items</strong></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><strong>json</strong></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layer</strong></td>
<td>Return the annotation layer this annotation pertains to.</td>
</tr>
<tr>
<td><strong>leftcontext</strong></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><strong>parsexml</strong></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>phoncontent</strong></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend</strong></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>precedes</strong></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><strong>previous</strong></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relaxng</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!)</td>
</tr>
<tr>
<td>resolveword</td>
<td></td>
</tr>
<tr>
<td>rightcontext</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td>sort</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>speech_speaker</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent</td>
<td>Get the text content explicitly associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textvalidation</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>wrefs</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td>xml</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td>xmlstring</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong></td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>

Class Attributes

ANNOTATIONTYPE = 48
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = 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=False, 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, None, or False) – 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()

adddtoindex (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. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).

Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, recursive=True, ignore=True, node=None)
Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

Returns int

dep Validation

Perform deep validation of this element.

Raises DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

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

Returns str or list

findcorrectionhandling (cls)
Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused

classmethod findreplaceables (parent, set=False, **kwargs)
Internal method to find replaceable elements. Auxiliary function used by AbstractElement.replace(). Can be overriden 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 \( (Class, set=False) \)
Returns an integer indicating whether such as annotation exists, and if so, how many. See annotations() for a description of the parameters.

hasphon \( (cls='current', strict=True, correctionhandling=1, hidden=False) \)
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

- \( \text{cls} \ (str) \) – The class of the phonetic content to obtain, defaults to current.
- \( \text{strict} \ (bool) \) – Set this if you are strictly interested in the phonetic content explicitly associated with the element, without recursing into children. Defaults to True.
- \( \text{correctionhandling} \) – 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 \( \text{bool} \)

hastext \( (cls='current', strict=True, correctionhandling=1, hidden=False) \)
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

- \( \text{cls} \ (str) \) – The class of the text content to obtain, defaults to current.
- \( \text{strict} \ (bool) \) – Set this if you are strictly interested in the text explicitly associated with the element, without recursing into children. Defaults to True.
- \( \text{correctionhandling} \) – 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 \( \text{bool} \)

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

Returns  
dict

layer ()
Return the annotation layer this annotation pertains to

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 (str) – 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 (str) – A list of classes which are never crossed looking for a next el-
ement. Set to True to constrain to a default list of structure elements (Sen-
tence,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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

classmethod parsexml (node, doc, **kwargs)
Internal class method used for turning an XML element into an instance of the Class.

Parameters

• node – XML Element (str) –

• doc – Document (str) –

Returns  An instance of the current Class.

phon (cls='current', previousdelimiter='', strict=False, correctionhandling=1, hidden=False)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever possible, as they are more spe-
cific. 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 outputed, 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.

• **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:
```python
text = 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
- `textcontent()`

**phoncontent** *(cls='current', correctionHandling=1, hidden=False)*
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 phonetic 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.
**precedes** (*other*)

Returns a boolean indicating whether this element precedes the other element.

**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* (*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* (*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, origclass=None*)

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

**classmethod relaxng_backwards**()

Internal helper function for backward compatibility.

**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.

**resolveoffsets** (*begin, end, retaintokenisation=True, strictend=True, cls='current'*)

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, recursive=True, ignore=False, 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 False (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`, `AlternativeLayers`, `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`)

```python
...
```

- `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()`.

- `setprocessor(processor)`
  Sets the processor for this element, taking care of adding an annotator in the declarations.

- `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.

- `sort(force=False)`
  Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage.

- `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, hidden=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 outputed, 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

* hidden *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)*

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.
• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

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()`

---

**folia.main.SyntacticUnit**

**class** `folia.main.SyntacticUnit`(doc, *args, **kwargs)

**Bases:** `folia.main.AbstractSpanAnnotation`

Syntactic Unit, span annotation element to be used in SyntaxLayer
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code>(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code>(child, *args, **kwargs)</td>
<td>Add a new element to the parent.</td>
</tr>
<tr>
<td><code>addable</code>(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>addidsuffix</code>(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addtoindex</code>([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td><code>ancestor</code>(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code>([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code>(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations</code>(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor</code>(annotator, ...)</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code>()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><code>copy</code>([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code>(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description</code>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables</code>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id</code>(cls)</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>getindex</code>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code>([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code>([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 55 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hasannotation</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer</code></td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><code>leftcontext</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code></td>
<td>Associate a document with this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 55 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>setspan(*args)</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>sort([force])</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>wrefs([index, recurse])</td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td>xmistring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, <class 'folia.main.Relation'>, <class 'folia.main.SyntacticUnit'>, <class 'folia.main.WordReference'>)

ANNOTATIONTYPE = 14

AUTH = True

AUTO_GENERATE_ID = False

HIDDEN = False

LABEL = 'Syntactic Unit'

OCCURRENCES = 0

OCCURRENCES_PER_SET = 0

OPTIONAL_ATTRIBS = (0, 1, 2, 4, 5, 8, 6, 7, 9, 10, 11)

PHONCONTAINER = False

PRIMARYELEMENT = False

PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELMITER = None
WREFABLE = False
XLINK = False
XMLTAG = 'su'

Method Details

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

clsmethod accepts (Class, raiseexceptions=\texttt{True}, parentinstance=\texttt{None})

add (child, *args, **kwargs)

classmethod addable (parent, set=False, raiseexceptions=\texttt{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 OCURRENCES property, but may be overridden
by subclasses for more customised behaviour.

Parameters

• parent (\texttt{AbstractElement}) – The element that is being added to

• set (\texttt{str}, \texttt{None}, or \texttt{\texttt{False}}) – The set

• raiseexceptions (\texttt{bool}) – Raise an exception if the element can’t be added?

Returns bool

Raises \texttt{ValueError}

addidsuffix (idsuffix, recursive=\texttt{True})
Appends a suffix to this element’s ID, and optionally to all child IDs as well. There is sually no need
to call this directly, invoked implicitly by \texttt{copy()}

addtoindex (norecurse=\texttt{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. Raise a No-
SuchAnnotation exception if not found.

Parameters \texttt{Classes} (*) – The possible classes (\texttt{AbstractElement} or subclasses) to se-
lect from. Not instances!

Example:
```python
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 (tuple of) classes (*AbstractElement* or subclasses). Not instances!

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

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

**annotations** (*Class, set=False*)
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.

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append** (*child, *args, **kwargs*)
See *AbstractElement.append()*

**checkdeclaration()**
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, recursive=True, ignore=True, node=None*)
Like *AbstractElement.select()* but instead of returning the elements, it merely counts them.

---

1.11. Annotations
Returns int

deepvalidation()
Perform deep validation of this element.

Raises DeepValidationError

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function
returns None

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:

```python
sense = word.annotation(foliaSense)
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=False, **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=False)
Returns an integer indicating whether such as annotation exists, and if so, how many. See
annotations() for a description of the parameters.

hashphon (cls='current', strict=True, correctionhandling=1, hidden=False)
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, hidden=False)*

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:

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

**Returns** dict

**layer** *

Return the annotation layer this annotation pertains to

**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

parsecommonarguments (**doc, **kwargs**)  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False**)  
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 outputed, 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.

- **hidden** (**bool**) – Include hidden elements, defaults to False.

Example:

```python
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():
phoncontent(cls='current', correctionhandling=1, hidden=False)

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.

precedes(other)

Returns a boolean indicating whether this element precedes the other element

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 (tuple) – 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 (tuple) – 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, origclass=None)

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

classmethod relaxng_backwards()

internal helper function for backward compatibility
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.

class resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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,
```

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()`

`setprocessor` *(processor)*
Sets the processor for this element, taking care of adding an annotator in the declarations

`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.

`sort` *(force=False)*
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

`speak_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

`speak_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, hidden=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`.

1.11. Annotations 647
• **previousdelimiter** *(str)* – Can be set to a delimiter that was last outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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, hidden=False)*

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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for text()

folia.main.SemanticRole

class folia.main.SemanticRole(doc, *args, **kwargs)

Bases: folia.main.AbstractSpanAnnotation

Semantic Role

Method Summary

__init__(doc, *args, **kwargs) Initialize self.

accepts(Class[, raiseexceptions, parentinstance])

add(child, *args, **kwargs)

addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.

addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.

addtoindex([norecurse]) 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.

ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

Continued on next page
Table 56 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>annotation</code></td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations</code></td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code></td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>getindex</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer</code></td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>leftcontext()</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext()</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select([Class], set, recursive, ignore, node)</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with strict=True</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
</tbody>
</table>
Table 56 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, <class 'folia.main.InternalAnnotation'>, <class 'folia.main.Label'>, <class 'folia.main.LabelSet'>, <class 'folia.main.LexicalEntry'>, <class 'folia.main.Metric'>, <class 'folia.main.Relation'>, <class 'folia.main.WordReference'>)

ANNOTATIONTYPE = 32

AUTH = True

AUTO_GENERATE_ID = False

HIDDEN = 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

WREFABLE = False

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=False, 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, None, or False)` – 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. Raise a No-SuchAnnotation exception if not found.

**Parameters**

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

**Example:**
```python
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**

- `Classes` – The class or (tuple of) classes (`AbstractElement` or subclasses).
  Not instances!

**Yields**

elements (instances derived from `AbstractElement`)

`annotation (type, set=False)`
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

`annotations (Class, set=False)`
Obtain annotations. Very similar to `select()` but raises an error if the annotation was not found.

**Parameters**

---

1.11. Annotations
- **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.

  **annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*
  Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

  **append** *(child, *args, **kwargs)*
  See AbstractElement.append()

  **checkdeclaration** ()
  Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

  **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=False, 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

  **depthfirstsearch** *(function)*
  Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

  **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:

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

**Returns**  
str or list

**findcorrectionhandling** *(cls)*  
Find the proper correction handling given a text class by looking in the underlying corrections where it is reused

**classmethod findreplaceables** *(parent, set=False, **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!

**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=False)*  
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, hidden=False)*  
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, hidden=False)*  
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:

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

    Returns dict

**layer** ()
Return the annotation layer this annotation pertains to

**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

**parsecommonarguments** (*doc, **kwargs*)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.
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=0, hidden=False)

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.
• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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=0, hidden=False)

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.

**precedes(other)**

Returns a boolean indicating whether this element precedes the other element

**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`, 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.

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

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

**classmethod relaxng_backwards()**

Internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!
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=False, 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 False (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, AlternativeLayers, 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:

```python
    ... 
```

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()

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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.

**sort** *(force=False)*
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

**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, hidden=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 outputed, 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
- **hidden** *(bool)* – Include hidden elements, defaults to **False**.

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, hidden=False)
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.
- **hidden** (bool) – Include hidden elements, defaults to False.

**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()

folia.main.TimeSegment

class folia.main.TimeSegment (doc, *args, **kwargs)
    Bases: folia.main.AbstractSpanAnnotation
    A time segment

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, [size] words to the left, the current word, and [size] words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>correct</strong>(<strong>kwargs</strong>)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count</strong>(<strong>class</strong>, set[, recursive, ignore[, node]])</td>
<td>Like <strong>AbstractElement.select()</strong>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description</strong>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables</strong>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>generate_id</strong>(cls)</td>
<td></td>
</tr>
<tr>
<td><strong>getindex</strong>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong>(key)</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong>(retaintokenisation)</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong>(<strong>class</strong>, set)</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasphon</strong>(cls[, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><strong>hastext</strong>(cls[, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><strong>incorrection</strong>()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><strong>insert</strong>(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td><strong>items</strong>(founditems)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><strong>json</strong>(attribs, recurse[, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layer</strong>()</td>
<td>Return the annotation layer this annotation pertains to.</td>
</tr>
<tr>
<td><strong>leftcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong>(Class, scope[, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong>(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong>(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><strong>parsexml</strong>(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong>(cls[, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>phoncontent</strong>(cls[, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend</strong>()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[,...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select([Class[, set, recursive, ignore, node]])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
</tbody>
</table>
Table 57 – continued from previous page

**Class Attributes**

```
ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.ActorFeature'>, ...
ANNOTATIONTYPE = 24
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = 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
WREFABLE = False
XLINK = False
XMLTAG = 'timesegment'
```

**Method Details**

```
__str__()  # Alias for text()
```

```
__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=False, 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.
```

1.11. Annotations 665
Parameters

• **parent** (*AbstractElement*) – The element that is being added to

• **set** (*str, None, or False*) – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (*AbstractElement* or subclasses). Not instances!

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

**annotation** (*type, set=False*)

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

**annotations** (*Class, set=False*)

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.

**annotator2processor** (*annotator=None, annotatortype=None, parentprocessor=None*)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See *AbstractElement.append()*
checkdeclaration()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

custom (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 appended 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=False, 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

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.

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:

```python
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=False, **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=False)
   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, hidden=False)
   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 correc-
     tion, and CorrectionHandling.EITHER if you don’t care.

   Returns bool

hastext (cls='current', strict=True, correctionhandling=1, hidden=False)
   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 en-
     countered. 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:

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

Returns dict

layer()  
Return the annotation layer this annotation pertains to

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

doc.commonarguments(doc, **kwargs)
  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
  
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 outputed, 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.
- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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() (cls='current', correctionhandling=1, hidden=False)

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.

precedes(others)
Returns a boolean indicating whether this element precedes the other element

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 (object) – 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 (object) – A list of classes which are never crossed looking for a next el-
ement. 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, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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 al-
ternative. Simply use AbstractElement.append() if you want the added element
• be an alternative. (to)
See AbstractElement.append() for more information and all parameters.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc
are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
  ...
```

**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()`

**setprocessor**(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort**(force=False)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not
needed), False if sort could not be performed at this stage
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

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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)
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.
- `hidden` *(bool)* – Include hidden elements, defaults to `False`.

**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:**

```python
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:

<table>
<thead>
<tr>
<th>Annotation Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChunkingLayer</td>
<td>Annotation layer for Chunk span annotation elements</td>
</tr>
<tr>
<td>CoreferenceLayer</td>
<td>Syntax layer: Annotation layer for SyntacticUnit span annotation elements</td>
</tr>
<tr>
<td>DependenciesLayer</td>
<td>Dependencies layer: Annotation layer for Dependency span annotation elements.</td>
</tr>
<tr>
<td>EntitiesLayer</td>
<td>Entities layer: Annotation layer for Entity span annotation elements.</td>
</tr>
<tr>
<td>ObservationLayer</td>
<td>Observation layer: Annotation layer for Observation span annotation elements.</td>
</tr>
<tr>
<td>SentimentLayer</td>
<td>Sentiment layer: Annotation layer for Sentiment span annotation elements, used for sentiment analysis.</td>
</tr>
<tr>
<td>StatementLayer</td>
<td>Statement layer: Annotation layer for Statement span annotation elements, used for attribution annotation.</td>
</tr>
<tr>
<td>SyntaxLayer</td>
<td>Syntax layer: Annotation layer for SyntacticUnit span annotation elements</td>
</tr>
<tr>
<td>SemanticRolesLayer</td>
<td>Syntax layer: Annotation layer for SemanticRole span annotation elements</td>
</tr>
</tbody>
</table>

**folia.main.ChunkingLayer**

class folia.main.ChunkingLayer (doc, *args, **kwargs)  
Bases: folia.main_AbstractAnnotationLayer

Chunking Layer: Annotation layer for Chunk span annotation elements

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts (Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add (child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex [n recurse]</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor (*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ancestors</strong>(<a href="Class"></a>)</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><strong>annotation</strong>(type[, set])</td>
<td>Will return a <strong>single</strong> annotation (even if there are multiple).</td>
</tr>
<tr>
<td><strong>annotations</strong>(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><strong>annotator2processor</strong>([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><strong>append</strong>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><strong>checkdeclaration()</strong></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><strong>context</strong>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td><strong>copy</strong>([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren</strong>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>correct</strong>(<strong>kwargs</strong>)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count</strong>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation()</strong></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description()</strong></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><strong>findreplaceables</strong>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>findspan</strong>(*words)</td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><strong>generate_id</strong>(cls)</td>
<td></td>
</tr>
<tr>
<td><strong>getindex</strong>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong>([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong>([reintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong>(Class[, set])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><strong>hasphon</strong>([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><strong>hastext</strong>([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><strong>incorrection()</strong></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><strong>insert</strong>(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
</tbody>
</table>

Continued on next page
Table 59 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>json</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>leftcontext</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword</td>
<td></td>
</tr>
<tr>
<td>rightcontext</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>sort</td>
<td></td>
</tr>
<tr>
<td>speech_speaker</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext</td>
<td>Alias for text() with strict=True</td>
</tr>
<tr>
<td>text</td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 59 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retain_tokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attrs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

```
ANNOTATIONTYPE = 15
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELMITER = None
WREFABLE = False
XLINK = False
XMLTAG = 'chunking'
```

678 Chapter 1. Reading FoLiA
Method Details

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

```python
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
```
Tests whether a new element of this class can be added to the parent.

Parameters

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

Returns bool

Raises ValueError

```python
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()`

```python
addtoindex(norecurse=None)
```
Makes sure this element (and all subelements), are properly added to the index.

Mostly for internal use.

```python
ancestor(*Classes)
```
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

Parameters

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

Example:

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

```python
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

- **Classes** *(AbstractElement or subclasses)*
  Not instances!

Yields elements (instances derived from AbstractElement)

```python
annotation(type, set=False)
```
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

```python
annotations(Class, set=False)
```
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.

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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.

**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=False, 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

**depthfirstsearch**(function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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=False, **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=False)
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, hidden=False)
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, hidden=False)

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=[]) (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:

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

Returns dict

leftcontext (size=None, 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

parsecommonarguments (doc, **kwargs)
    Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

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, hidden=False)
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 spe-
cific. 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 possi-
ble. Defaults to False.
• previousdelimiter (str) – Can be set to a delimiter that was last outputed, 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 cor-
rections 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 correc-
tion, and CorrectionHandling.EITHER if you don’t care.
• hidden (bool) – Include hidden elements, defaults to False.

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 \(\text{cls}='\text{current}', \text{correctionhandling}=1, \text{hidden}=False\)

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

Unlike \text{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

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

Returns The phonetic content (\text{PhonContent})

Raises NoSuchPhon if there is no phonetic content for the element

See also: \text{phon()} \text{textcontent()} \text{text()}

\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.

\text{precedes}(\text{other})

Returns a boolean indicating whether this element precedes the other element

\text{previous}(\text{Class}=True, \text{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

- \text{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
- \text{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.

\text{classmethod relaxng}(\text{includechildren}=True, \text{extraattrs}=None, \text{extraelements}=None, \text{origclass}=None)

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

\text{classmethod relaxng_backwards}()

internal helper function for backward compatibility

\text{remove}(\text{child})

Removes the child element

\text{replace}(\text{child}, *\text{args}, **\text{kwargs})

Appends a child element like \text{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()
• **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 to be an alternative.

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

**resolveoffsets** (*begin, end, retaintokenisation=True, strictend=True, cls='current'*)

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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`, `AlternativeLayers`, `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:**

```python
   ...
```

**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()`
setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations.

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.

sort ()
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, hidden=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 outputed, 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.
- **hidden** (*bool*) – Include hidden elements, defaults to **False**.

Example:

```python
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, *hidden*=False)

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.
- **hidden** (*bool*) – Include hidden elements, defaults to **False**.

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

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

See also:

```python
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:

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

__len__ ()
Returns the number of child elements under the current element.

__str__ ()
Alias for `text()`

**folia.main.CoreferenceLayer**

class folia.main.CoreferenceLayer (doc, *args, **kwargs)
Bases: folia.main.AbstractAnnotationLayer

Syntax Layer: Annotation layer for `SyntacticUnit` span annotation elements

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong> (doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>addable</td>
<td>(parent[, set, raiseexceptions])</td>
</tr>
<tr>
<td>addidsuffix</td>
<td>(idsuffix[, recursive])</td>
</tr>
<tr>
<td>addtoindex</td>
<td>([norecurse])</td>
</tr>
<tr>
<td>ancestor</td>
<td>(*Classes)</td>
</tr>
<tr>
<td>ancestors</td>
<td>([Class])</td>
</tr>
<tr>
<td>annotation</td>
<td>(type[, set])</td>
</tr>
<tr>
<td>annotations</td>
<td>(Class[, set])</td>
</tr>
<tr>
<td>annotator2processor</td>
<td>([annotator, ...])</td>
</tr>
<tr>
<td>append</td>
<td>(child, *args, **kwargs)</td>
</tr>
<tr>
<td>checkdeclaration</td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>(size[, placeholder, scope])</td>
</tr>
<tr>
<td>copy</td>
<td>([newdoc, idsuffix])</td>
</tr>
<tr>
<td>copychildren</td>
<td>([newdoc, idsuffix])</td>
</tr>
</tbody>
</table>
Table 60 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan(*words)</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td><code>has_text([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecmonkeyarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsecxml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 60 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

`ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.CoreferenceChain'>, ...)`
ANNOTATIONTYPE = 31
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – The set
• raiseexceptions (bool) – Raise an exception if the element can’t be added?

Returns bool

1.11. Annotations
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. Raise a No-
    SuchAnnotation exception if not found.

        Parameters Classes (*) – The possible classes (AbstractElement or subclasses) to se-
        lect 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 (tuple of) classes (AbstractElement or subclasses).
        Not instances!

        Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
    Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if
    none was found

annotations (Class, set=False)
    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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
    Converts annotator information to processor information (FoLiA v2). Can be called with arguments to
    override defaults.

append (child, *args, **kwargs)
    See AbstractElement.append()

checkdeclaration ()
    Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
    declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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 DeepValidationException

**depthfirstsearch** *(function)*

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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=False, **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)*

Get the index at which an element occurs.

**Returns** int

**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=False)*

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, hidden=False)*

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, hidden=False)*

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:
    
    ```python
    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

parsecommonarguments (doc, **kwargs)
    Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
    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 outputed, 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.

• **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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
- `textcontent()`

```python
phoncontent(cls='current', correctionhandling=1, hidden=False)
```

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 phonetic 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()`

```python
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.
precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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.

resolveoffsets (begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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`, `AlternativeLayers`, `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`)

```python
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()`

**setprocessor**(processor)

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort**()

**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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)  
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.

• hidden (bool) – Include hidden elements, defaults to False.

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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

**Folia.main.DependenciesLayer**

class folia.main.DependenciesLayer (doc, *args, **kwargs)
Bases: folia.main.AbstractAnnotationLayer

Dependencies Layer: Annotation layer for Dependency span annotation elements. For dependency entities.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts</td>
<td>(Class[, raiseexceptions, parentinstance])</td>
</tr>
<tr>
<td>add</td>
<td>(child, *args, **kwargs)</td>
</tr>
</tbody>
</table>
### Table 61 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>addable</strong> (parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><strong>addidsuffix</strong> (idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><strong>addtoindex</strong> ([norecurse])</td>
<td>Makes sure this element (and all subelements) are properly added to the index.</td>
</tr>
<tr>
<td><strong>ancestor</strong> (<em>Classes</em>)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><strong>ancestors</strong> ([<em>Class]</em>)</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><strong>annotation</strong> (type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><strong>annotations</strong> (<em>Class</em>[ , set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><strong>annotator2processor</strong> ([annotator, …])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><strong>append</strong> (child, *args, **kwargs)</td>
<td>See <strong>AbstractElement.append()</strong></td>
</tr>
<tr>
<td><strong>checkdeclaration</strong> ()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><strong>context</strong> (size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><strong>copy</strong> ([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><strong>copychildren</strong> ([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><strong>correct</strong> (<strong>kwargs</strong>)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><strong>count</strong> (<em>Class</em>[ , set, recursive, ignore, node])</td>
<td>Like <strong>AbstractElement.select()</strong>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><strong>deepvalidation</strong> ()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><strong>depthfirstsearch</strong> (function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><strong>description</strong> ()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><strong>feat</strong> (subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><strong>findcorrectionhandling</strong> (cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><strong>findreplaceables</strong> (parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><strong>findspan</strong> (*words)</td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><strong>generate_id</strong> (cls)</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><strong>getmetadata</strong> ([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><strong>gettextdelimiter</strong> ([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>hasannotation</strong> (<em>Class</em>[ , set])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 61 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattribs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[.., ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select([Class[, set, recursive, ignore, node]])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 61 – continued from previous page

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

**Class Attributes**

- `ACCEPTED_DATA` = (`<class 'folia.main.Comment'>`, `<class 'folia.main.Correction'>`, `<class 'folia.main.Dependency'>`, `<class 'folia.main.Description'>`, `<class 'folia.main.ForeignData'>`)
- `ANNOTATIONTYPE` = 23
- `AUTH` = True
- `AUTO_GENERATE_ID` = False
- `HIDDEN` = False
- `OCCURRENCES` = 0
- `OCCURRENCES_PER_SET` = 0
- `OPTIONAL_ATTRIBS` = (0,)
- `PHONCONTAINER` = False
- `PRIMARYELEMENT` = False
- `PRINTABLE` = False
- `REQUIRED_ATTRIBS` = None
- `REQUIRED_DATA` = None
- `SETONLY` = True
- `SPEAKABLE` = False
- `SUBSET` = None
- `TEXTCONTAINER` = False
- `TEXTDELIMITER` = None
WREFABLE = False
XLINK = False
XMLTAG = 'dependencies'

Method Details

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

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

```python
classmethod accepts(Class, raiseexceptions=True, parentinstance=None)
```

```python
classmethod addable(parent, set=False, 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, None, or False) – The set
- **raiseexceptions** (bool) – Raise an exception if the element can’t be added?

Returns bool

Raises ValueError

```python
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()
```

```python
addtoindex(norecurse=[])
Makes sure this element (and all subelements), are properly added to the index.
Mostly for internal use.
```

```python
ancestor(*Classes)
Find the most immediate ancestor of the specified type, multiple classes may be specified. Raise a No-SuchAnnotation exception if not found.

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

Example:
```python
paragraph = word.ancestor(folia.Paragraph)
```

```python
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 (tuple of) classes (AbstractElement or subclasses).

Yields elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found.

**annotations** *(Class, set=False)*
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.

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

**append** *(child, *args, **kwargs)*
See `AbstractElement.append()`.

**checkdeclaration** ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

**depthfirstsearch** *(function)*
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.
**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:**

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

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correction handling given a textclass by looking in the underlying corrections where it is

**classmethod findreplaceables(parent, set=False, **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=False)**

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, hidden=False)**

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=True, hidden=False)*

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:

```python
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

**parsecommonarguments** (*doc**, **kwargs)  
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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**, **hidden**= **False**)  
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden** (**bool**) – Include hidden elements, defaults to False.

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
textcontent()

phoncontent (cls='current', correctionhandling=1, hidden=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards ()
internal helper function for backward compatibility

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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
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 scope. There is usually no need to call this directly, invoked implicitly by copy().

setprocessor(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations.

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.

sort()

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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

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, hidden=False)**

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.
- **hidden (bool)** – Include hidden elements, defaults to False.

**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:

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

__len__() Returns the number of child elements under the current element.

__str__() Alias for `text()`

### folia.main.EntitiesLayer

class folia.main.EntitiesLayer(doc, *args, **kwargs)

Bases: folia.main.AbstractAnnotationLayer

Entities Layer: Annotation layer for Entity span annotation elements. For named entities.

#### Method Summary

- `__init__`(doc, *args, **kwargs) Initialize self.
- `accepts`(Class[, raiseexceptions, parentinstance])
- `add`(child, *args, **kwargs)
- `addable`(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.
- `addidsuffix`(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.
- `addtoindex`([norecurse]) 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.
- `ancestors`([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
- `annotation`(type[, set]) Will return a single annotation (even if there are multiple).
- `annotations`(Class[, set]) Obtain annotations.
- `annotator2processor`([annotator, …]) Converts annotator information to processor information (FoLiA v2).
- `append`(child, *args, **kwargs) See `AbstractElement.append()`.
- `checkdeclaration`() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.
- `context`(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right.
- `copy`([newdoc, idsuffix]) Make a deep copy of this element and all its children.

Continued on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren(newdoc, idsuffix)</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan(*words)</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>Items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng(includechildren, extraattrs, ...)</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element.</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code>.</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code>.</td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attrs, elements, skipchildren])</code></td>
<td>See AbstractElement.xml().</td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.Entity'>, <class 'folia.main.ForeignData'>)
ANNOTATIONTYPE = 16
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

  **Example:**

  ```python
  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 (tuple of) classes *(AbstractElement or subclasses)*.

  Not instances!

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

**annotation** *(type, set=False)*

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

**annotations** *(Class, set=False)*

  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.

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

  Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

  See AbstractElement.append()

**checkdeclaration** *

  Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, recursive=True, ignore=True, node=None)
    Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

    Returns int

dependentvalidation ()
    Perform deep validation of this element.

    Raises DeepValidationWarning

depthfirstsearch (function)
    Generic depth first search algorithm using a callback function, continues as long as the callback function
    returns None

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=False, **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=False)
    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, hidden=False)
    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 correc-
        tion, and CorrectionHandling.EITHER if you don’t care.

    Returns bool

hastext(cls='current', strict=True, correctionhandling=1, hidden=False)
    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 en-
        countered. 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:

```python
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`

parsecommonarguments(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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 outputed, 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.

- **hidden** (bool) – Include hidden elements, defaults to `False`.

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
* `textcontent()`
* `text()`

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

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards**()

Internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

**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()*

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** *

**speechSpeaker** *

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, hidden=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 outputed, 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
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**Example:**

```python
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, hidden=False)*  
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.
• **hidden** (*bool*) – Include hidden elements, defaults to False.

**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:

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

__len__() Returns the number of child elements under the current element.

__str__() Alias for text()
__init__(doc, *args, **kwargs) Initialize self.

accepts(Class[, raiseexceptions, parentinstance])

add(child, *args, **kwargs) Tests whether a new element of this class can be added to the parent.

addable(parent[, set, raiseexceptions])

addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.

addtoindex([norecurse]) 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.

ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.

annotation(type[, set]) Will return a single annotation (even if there are multiple).

annotations(Class[, set]) Obtain annotations.

annotator2processor([annotator, ...]) Converts annotator information to processor information (FoLiA v2).

append(child, *args, **kwargs) See AbstractElement.append()

checkdeclaration() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

context(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right

copy([newdoc, idsuffix]) Make a deep copy of this element and all its children.

copychildren([newdoc, idsuffix]) Generator creating a deep copy of the children of this element.

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

count(Class[, set, recursive, ignore, node]) Like AbstractElement.select(), but instead of returning the elements, it merely counts them.

depthfirstsearch(function) Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

description() Obtain the description associated with the element.

feat(subset) Obtain the feature class value of the specific subset.

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

findreplaceables(parent[, set]) Internal method to find replaceable elements.

findspan(*words) Returns the span element which spans over the specified words or morphemes.

generate_id(cls)

getindex(child[, recursive, ignore]) Get the index at which an element occurs, recursive by default!

getmetadata([key]) Get the metadata that applies to this element, automatically inherited from parent elements

gettextdelimiter([retaintokenisation]) Return the text delimiter for this class.
Table 63 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasannotation([Class[, set]])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorporation()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattribs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select([Class[, set, recursive, ignore, node]])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 63 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext([text[, cls]])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```python
ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Observation'>)
ANNOTATIONTYPE = 46
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = 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)
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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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 (tuple of) classes (AbstractElement or subclasses).

Not instances!

Yields elements (instances derived from AbstractElement)

**annotation** *(type, set=False)*

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

**annotations** *(Class, set=False)*

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.

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See AbstractElement.append()

**checkdeclaration** *

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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 DeepValidationException

---

Chapter 1. Reading FoLiA
**depthfirstsearch** *(function)*
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**description()**
Obtain the description associated with the element.

*Raised* 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:
```python
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=False, **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=False)
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, hidden=False)
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, \*hidden=False})

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:

```python
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**

**parsecommonarguments** (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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=CorrectionHandling.CURRENT, hidden=False)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden** (bool) – Include hidden elements, defaults to False.

**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, hidden=False)`

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.

`precedes(other)`

Returns a boolean indicating whether this element precedes the other element.

`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’, 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.

`classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None)`

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

`classmethod relaxng_backwards()`

internal helper function for backward compatibility

`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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
    Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

    ```python
        ...
    ```

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 scoped. There is usually no need to call this directly, invoked implicitly by `copy()`.

**setprocessor**(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations.

**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.

**sort()**

**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, hidden=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 outputed, 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
- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)**

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.
- **hidden (bool)** – Include hidden elements, defaults to False.

**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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()
Table 64 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren</code>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan(*words)</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 64 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>spirit()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code>.</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code>.</td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Sentiment'>)
ANNOTATIONTYPE = 47
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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()

**adddtoindex**(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. Raise a NoSuchAnnotation exception if not found.

  **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 (tuple of) classes (AbstractElement or subclasses).

  Not instances!

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

**annotation**(type, set=False)

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

**annotations**(Class, set=False)

  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.

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

  Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

  See AbstractElement.append()

**checkdeclaration**()

  Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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 appended 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=False, 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** DeepValidationWarning

**depthfirstsearch** *(function)*

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:

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

**Returns** str or list

**findcorrectionhandling** *(cls)*

Find the proper correction handling given a text class by looking in the underlying corrections where it is reused

**classmethod findreplaceables** *(parent, set=False, **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 TEXTDELMITER attribute but may return a customised one instead.

hasannotation(Class, set=False)
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, hidden=False)
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, hidden=False)
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:

   ```python
   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

every (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 el-
     ement. 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

parsecommonarguments(doc, **kwargs)
   Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke
directly.

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, hidden=False)
   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 spe-
   cific. 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 outputed, 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.

- **hidden (bool)** – Include hidden elements, defaults to *False*.

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

`textcontent()`

`phoncontent (cls='current', correctionhandling=1, hidden=False)`

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 phonetic 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.

**precedes** *(other)*

Returns a boolean indicating whether this element precedes the other element

**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 (str)** – 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 (str)** – 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, origclass=None)*

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

**classmethod relaxng_backwards** *

Internal helper function for backward compatibility

**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.

**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:**

```python
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()`

**setprocessor**(processor)

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort**( )

**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, hidden=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

- **hidden** *(bool)* – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)*

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.
• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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()`

**folia.main.StatementLayer**

**class** `folia.main.StatementLayer`(doc, *args, **kwargs)

**Bases:** `folia.main.AbstractAnnotationLayer`

Statement Layer: Annotation layer for *Statement* span annotation elements, used for attribution annotation.

**Method Summary**
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code>(Class[, raiseexceptions, parentinstance])</td>
<td><code>add</code>(child[, *args, **kwargs])</td>
</tr>
<tr>
<td><code>addable</code>(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>addidsuffix</code>(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addtoindex</code>([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>ancestor</code>(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code>([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code>(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations</code>(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor</code>([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code>(child[, *args, **kwargs])</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code>()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code>(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy</code>([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code>(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description</code>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables</code>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan</code>(*words)</td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id</code>(cls)</td>
<td></td>
</tr>
<tr>
<td><code>getindex</code>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code>([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code>([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>hasannotation</code>([Class[, set]])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon</code>([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext</code>([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code>()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert</code>(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td><code>items</code>([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json</code>([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext</code>(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next</code>([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext</code>([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments</code>(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml</code>(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon</code>([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent</code>([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend</code>()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes</code>(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous</code>([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng</code>([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards</code>()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove</code>(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace</code>(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets</code>(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code>(id)</td>
<td></td>
</tr>
<tr>
<td><code>rightcontext</code>(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code>([Class[, set, recursive, ignore, node]])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc</code>(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code>(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents</code>()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
</tbody>
</table>
Table 65 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

```python
ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Statement'>)
ANNOTATIONTYPE = 48
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
```

1.11. Annotations
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=False, raiseexceptions=True)

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

Parameters

• parent (AbstractElement) – The element that is being added to
• set (str, None, or False) – 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. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (•) – The possible classes (AbstractElement or subclasses) to se-
lect 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 (tuple of) classes (*AbstractElement* or subclasses).
  Not instances!

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

**annotation**(type, set=True)

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

**annotations**(Class, set=True)

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.

**annotator2processor**(annotator=None, annotatortype=None, parentprocessor=None)

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See *AbstractElement.append()*

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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*
depthfirstsearch (function)
    Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:
    
    ```python
    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=False, **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=False)
    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, hidden=False)
    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, hidden=False)  
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:

```python
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`

**parsecommonarguments** (*doc, **kwargs*)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False*)
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.
- **hidden** (*bool*) – Include hidden elements, defaults to False.

Example:

```python
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
phoncontent():

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.

precedes(other) Returns a boolean indicating whether this element precedes the other element

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', 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.

classmethod relaxng(includechildren=True, extraattrs=None, extraelements=None, origclass=None) Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards() internal helper function for backward compatibility

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.

resolveoffsets(begn, end, retaintokenisation=True, strictend=True, cls='current')

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

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()`

`setprocessor(processor)`
Sets the processor for this element, taking care of adding an annotator in the declarations

`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.

`sort()`

`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, hidden=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 outputed, 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

- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)**

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.

- **hidden (bool)** – Include hidden elements, defaults to False.

**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:

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

__len__() Returns the number of child elements under the current element.

__str__() Alias for text()

folia.main.SyntaxLayer
class folia.main.SyntaxLayer(doc, *args, **kwargs)
Bases: folia.main.AbstractAnnotationLayer
Syntax Layer: Annotation layer for SyntacticUnit span annotation elements

Method Summary

__init__(doc, *args, **kwargs) Initialize self.
accepts(Class[, raiseexceptions, parentinstance])
add(child, *args, **kwargs)
addable(parent[, set, raiseexceptions]) Tests whether a new element of this class can be added to the parent.
addidsuffix(idsuffix[, recursive]) Appends a suffix to this element’s ID, and optionally to all child IDs as well.
addtoindex([norecurse]) 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.
ancestors([Class]) Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
annotation(type[, set]) Will return a single annotation (even if there are multiple).
annotations(Class[, set]) Obtain annotations.
annotator2processor([annotator, . . .]) Converts annotator information to processor information (FoLiA v2).
append(child, *args, **kwargs) See AbstractElement.append()
checkdeclaration() Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.
context(size[, placeholder, scope]) Returns this word in context, {size} words to the left, the current word, and {size} words to the right.
copy([newdoc, idsuffix]) Make a deep copy of this element and all its children.
### Table 66 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copychildren([newdoc, idsuffix])</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct(**kwargs)</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count(Class[, set, recursive, ignore, node])</code></td>
<td>Like <code>AbstractElement.select()</code>, but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation()</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch(function)</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description()</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat(subset)</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling(cls)</code></td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td><code>findreplaceables(parent[, set])</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan(*words)</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id(cls)</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex(child[, recursive, ignore])</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata([key])</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter([retaintokenisation])</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation(Class[, set])</code></td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, . . .])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td></td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, . . .])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select(Class[, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech Speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attrs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>, <class 'folia.main.SyntacticUnit'>)
ANNOTATIONTYPE = 14
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

**Example:**

```python
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 (tuple of) classes *(AbstractElement or subclasses)*. Not instances!

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

**annotation** *(type, set=False)*

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

**annotations** *(Class, set=False)*

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.

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See *AbstractElement.append()*

**checkdeclaration** *

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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

depthfirstsearch (function)
    Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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=False, **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=False)
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, hidden=False)
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, hidden=False)
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
incorrect()  
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:

```python
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

parsecommonarguments(doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)

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 outputed, useful when chaining calls to `phon()`.
- **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.
- **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:

```python
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
- `textcontent()`
- `text()`

`phoncontent` *(cls='current', correctionhandling=1, hidden=False)*

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 phonetic 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.

**precedes** *(other)*

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards** ()

internal helper function for backward compatibility

**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.

**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:**

```python
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()`

**setprocessor**(processor)

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort**()

**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, hidden=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 outputed, 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

• hidden (bool) – Include hidden elements, defaults to False.

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, hidden=False)
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.
• **hidden** *(bool)* – Include hidden elements, defaults to *False*.

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

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

See also:

text() phoncontent() phon()

textvalidation *(warnonly=\text{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'\text{\text{)}*  

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=\text{None}, elements=\text{None}, skipchildren=\text{False}\text{\text{)}*  

See AbstractElement.xml()

xmlstring *(pretty_print=\text{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:

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

__len__()

Returns the number of child elements under the current element.

__str__()

Alias for text()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code></td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>add</code></td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>addidsuffix</code></td>
<td>Adds a suffix to this element’s ID.</td>
</tr>
<tr>
<td><code>addtoindex</code></td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>ancestor</code></td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code></td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code></td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations</code></td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor</code></td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td><code>append</code></td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code></td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code></td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code></td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code></td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code></td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code></td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code></td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code></td>
<td>Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>findspan</code></td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td><code>generate_id</code></td>
<td></td>
</tr>
<tr>
<td><code>getindex</code></td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code></td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td><code>gettextdelimiter</code></td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td><strong>Continued on next page</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Table 67 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hasannotation([cls, set])</code></td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext([cls, strict, correctionhandling, ...])</code></td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection()</code></td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert(index, child, *args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>items([founditems])</code></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json([attribs, recurse, ignorelist])</code></td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>leftcontext(size[, placeholder, scope])</code></td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next([Class, scope, reverse])</code></td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>originaltext([cls])</code></td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments(doc, **kwargs)</code></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml(node, doc, **kwargs)</code></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon([cls, previousdelimiter, strict, ...])</code></td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent([cls, correctionhandling, hidden])</code></td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend()</code></td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes(other)</code></td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous([Class, scope])</code></td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><code>relaxng([includechildren, extraattrs, ...])</code></td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards()</code></td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove(child)</code></td>
<td>Removes the child element</td>
</tr>
<tr>
<td><code>replace(child, *args, **kwargs)</code></td>
<td>Appends a child element like <code>append()</code>, but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><code>resolveoffsets(begin, end[, ...])</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext(size[, placeholder, scope])</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select([Class, set, recursive, ignore, node])</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc(newdoc)</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument(doc)</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scope.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 67 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td></td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

Class Attributes

`ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>, <class 'folia.main.Predicate'>, <class 'folia.main.SemanticRole'>)`

`ANNOTATIONTYPE = 32`

`AUTH = True`

`AUTO_GENERATE_ID = False`

`HIDDEN = False`

`OCCURRENCES = 0`

`OCCURRENCES_PER_SET = 0`

`OPTIONAL_ATTRIBS = (0,)`

`PHONCONTAINER = False`

`PRIMARYELEMENT = False`

`PRINTABLE = False`

`REQUIRED_ATTRIBS = None`

`REQUIRED_DATA = None`

`SETONLY = True`

`SPEAKABLE = False`

`SUBSET = None`
TEXTCONTAINER = False
TEXTDELRIMITER = None
WREFABLE = False
XLINK = False
XMLTAG = 'semroles'

Method Details

__init__

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

__init__

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

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

add(child, *args, **kwargs)

classmethod addable (parent, set=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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=False, 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 DeepValidationWarning
**depthfirstsearch** *(function)*
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

**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:
```python
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=False, **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=False)
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, hidden=False)
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, hidden=False)

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:

```python
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

**parsecommonarguments** (doc, **kwargs)

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

**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, hidden=False)

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

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

• `hidden` (bool) – Include hidden elements, defaults to False.

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, hidden=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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', 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.

classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards()
internal helper function for backward compatibility

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.

resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')
  Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

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=False, 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 False (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, AlternativeLayers, 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:

```python
  ...
```

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()*

**setprocessor(processor)**
Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort ()**

**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, hidden=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 outputed, 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

• **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.

• **hidden (bool)** – Include hidden elements, defaults to False.

**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:

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

__len__()  
Returns the number of child elements under the current element.

__str__()  
Alias for text()

---

### foilia.main.TimingLayer

**class foilia.main.TimingLayer(doc, *args, **kwargs)**  
Bases: foilia.main.AbstractAnnotationLayer

Timing layer: Annotation layer for TimeSegment span annotation elements.

---

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td>ancestor(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copychildren**(newdoc, idsuffix)**</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct**(kwargs)**</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count**(Class[, set, recursive, ignore, node])**</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch**(function)**</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat**(subset)**</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling**(cls)**</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused</td>
</tr>
<tr>
<td>findreplaceables**(parent[, set])**</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>findspan**(parent[, set])**</td>
<td>Returns the span element which spans over the specified words or morphemes.</td>
</tr>
<tr>
<td>generate_id**(cls)**</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>getindex**(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata**(key)**</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td>gettextdelimiter**(retaintokenisation)**</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation**(Class[, set])**</td>
<td>Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon**(cls, strict, correctionhandling, . . .)**</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hasphon**(cls, previousdelimiter, strict, . . .)**</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert**(index, child, *args, <strong>kwargs)</strong></td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>items**(founditems)**</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>json**(attribs, recurse, ignorelist)**</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>leftcontext**(size[, placeholder, scope])**</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next**(Class, scope, reverse)**</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext**(cls)**</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments**(doc, <strong>kwargs)</strong></td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsesxml**(node, doc, <strong>kwargs)</strong></td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon**(cls, previousdelimiter, strict, . . .)**</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent**(cls, correctionhandling, hidden)**</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
</tbody>
</table>
Table 68 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element.</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattrs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string).</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility.</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element.</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select(Class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td>setparents()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td>setprocessor(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td>settext(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td>sort()</td>
<td></td>
</tr>
<tr>
<td>speech_speaker()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>speech_src()</td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td>stricttext([cls])</td>
<td>Alias for text() with strict=True.</td>
</tr>
<tr>
<td>text([cls, retaintokenisation, ...])</td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td>textcontent([cls, correctionhandling, hidden])</td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>textvalidation([warnonly])</td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td>toktext([cls])</td>
<td>Alias for text() with retaintokenisation=True.</td>
</tr>
<tr>
<td>updatetext()</td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td>xml([attribs, elements, skipchildren])</td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td>xmlstring([pretty_print])</td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><strong>iter</strong>()</td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><strong>len</strong>()</td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>Alias for text()</td>
</tr>
</tbody>
</table>
Class Attributes

ACCEPTED_DATA = (<class 'folia.main.Comment'>, <class 'folia.main.Correction'>, <class 'folia.main.Description'>, <class 'folia.main.ForeignData'>, <class 'folia.main.TimeSegment'>)
ANNOTATIONTYPE = 24
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = False
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = True
SPEAKABLE = False
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a NoSuchAnnotation exception if not found.

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

Example:

```python
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 (tuple of) classes (AbstractElement or subclasses).
Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

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 appended 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=False, 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 DeepValidationErro

depthfirstsearch (function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

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

Returns str or list

findcorrectionhandling (cls)
Find the proper correction handling given a textclass by looking in the underlying corrections where it is reused

classmethod findreplaceables (parent, set=False, **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 TEXTDELMITER attribute but may return a customised one instead.

hasannotation(Class, set=False)
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, hidden=False)
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, hidden=False)
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

parsecommonarguments(doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
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 outputed, 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.

- **hidden (bool)** – Include hidden elements, defaults to False.

Example:

```python
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()**: Retrieves the phonetic content as a string text()

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

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 phonetic 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.

**precedes**(other)
- Returns a boolean indicating whether this element precedes the other element

**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’, 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.

**classmethod relaxng**(includechildren=True, extraattrs=None, extraelements=None, oriclass=None)
- Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

**classmethod relaxng_backwards**()
- internal helper function for backward compatibility

**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.

**resolveoffsets**(begin, end, retaintokenisation=True, strictend=True, cls='current')
- Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

**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=False, 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 False (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, AlternativeLayers, 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:

```python
    ...
```

**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()`

**setprocessor**(processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort**()

**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, hidden=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 outputed, 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
- `hidden (bool)` – Include hidden elements, defaults to `False`.

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, hidden=False)`

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.
• **hidden** *(bool)* – Include hidden elements, defaults to False.

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

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

See also:

`text() phoncontent() phon()`

```plaintext
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

```plaintext
toktext(cls='current')
```

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

```plaintext
updatetext()
```

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

```plaintext
xml(attribs=None, elements=None, skipchildren=False)
```

See `AbstractElement.xml()`

```plaintext
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

```plaintext
__iter__()
```

Iterate over all children of this element.

Example:

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

```plaintext
__len__()
```

Returns the number of child elements under the current element.

```plaintext
__str__()
```

Alias for `text()`

Some span annotation elements take *span roles*, depending on their type:

<table>
<thead>
<tr>
<th>CoreferenceLink</th>
<th>Coreference link.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DependencyDependent</td>
<td>Span role element that marks the dependent in a dependency relation.</td>
</tr>
<tr>
<td>Headspan</td>
<td>The headspan role is used to mark the head of a span annotation.</td>
</tr>
</tbody>
</table>
**folia.main.CoreferenceLink**

**class** folia.main.CoreferenceLink(*doc, *args, **kwargs*)

**Bases:** folia.main.AbstractSpanRole

Coreference link. Used in `CoreferenceChain`

**Method Summary**

- `__init__(doc, *args, **kwargs)`: Initialize self.
- `accepts(Class[, raiseexceptions, parentinstance])`: Tests whether a new element of this class can be added to the parent.
- `add(child, *args, **kwargs)`: Adds a new element to the current element.
- `addable(parent[, set, raiseexceptions])`: Tests whether a new element of this class can be added to the parent.
- `addidsuffix(idsuffix[, recursive])`: Appends a suffix to this element’s ID, and optionally to all child IDs as well.
- `addtoIndex([norecurse])`: 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.
- `ancestors([Class])`: Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.
- `annotation(type[, set])`: Will return a single annotation (even if there are multiple).
- `annotations(Class[, set])`: Obtain annotations.
- `annotator2processor([annotator, ...])`: Converts annotator information to processor information (FoLiA v2).
- `append(child, *args, **kwargs)`: See `AbstractElement.append()`
- `checkdeclaration()`: Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.
- `context(size[, placeholder, scope])`: Returns this word in context, `{size}` words to the left, the current word, and `{size}` words to the right.
- `copy([newdoc, idsuffix])`: Make a deep copy of this element and all its children.
- `copychildren([newdoc, idsuffix])`: Generator creating a deep copy of the children of this element.
- `correct(**kwargs)`: Apply a correction (TODO: documentation to be written still)
- `count(Class[, set, recursive, ignore, node])`: Like `AbstractElement.select()`, but instead of returning the elements, it merely counts them.
- `deepvalidation()`: Perform deep validation of this element.
- `depthfirstsearch(function)`: Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.
- `description()`: Obtain the description associated with the element.
- `feat(subset)`: Obtain the feature class value of the specific subset.
- `findcorrectionhandling(cls)`: Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>findreplaceables</code></td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id</code></td>
<td>(cls)</td>
</tr>
<tr>
<td><code>getindex</code></td>
<td>(child[, recursive, ignore]) Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code></td>
<td>([key]) Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code></td>
<td>([retaintokenisation]) Return the text delimiter for this class.</td>
</tr>
<tr>
<td><code>hasannotation</code></td>
<td>(Class[, set]) Returns an integer indicating whether such an annotation exists, and if so, how many.</td>
</tr>
<tr>
<td><code>hasphon</code></td>
<td>([cls, strict, correctionhandling, ...]) Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td><code>hastext</code></td>
<td>([cls, strict, correctionhandling, ...]) Does this element have text (of the specified class)</td>
</tr>
<tr>
<td><code>incorrection</code></td>
<td>() Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>(index, child, *args, **kwargs) Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>items</code></td>
<td>([founditems]) Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td><code>json</code></td>
<td>([attribs, recurse, ignorelist]) Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><code>layer</code></td>
<td>() Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><code>leftcontext</code></td>
<td>(size[, placeholder, scope]) Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><code>next</code></td>
<td>([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.</td>
</tr>
<tr>
<td><code>originaltext</code></td>
<td>([cls]) Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><code>parsecommonarguments</code></td>
<td>(doc, **kwargs) Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><code>parsexml</code></td>
<td>(node, doc, **kwargs) Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><code>phon</code></td>
<td>([cls, previousdelimiter, strict, ...]) Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>phoncontent</code></td>
<td>([cls, correctionhandling, hidden]) Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>postappend</code></td>
<td>() This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><code>precedes</code></td>
<td>(other) Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><code>previous</code></td>
<td>([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.</td>
</tr>
<tr>
<td><code>relaxng</code></td>
<td>([includechildren, extraattribs, ...]) Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><code>relaxng_backwards</code></td>
<td>() Internal helper function for backward compatibility</td>
</tr>
<tr>
<td><code>remove</code></td>
<td>(child) Removes the child element</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>(child, *args, **kwargs) Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 70 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolveoffsets</code></td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><code>resolveword</code></td>
<td></td>
</tr>
<tr>
<td><code>rightcontext</code></td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><code>select</code></td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><code>setdoc</code></td>
<td>Set a different document.</td>
</tr>
<tr>
<td><code>setdocument</code></td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><code>setparents</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext</code></td>
<td>Alias for <code>text()</code> with strict=True</td>
</tr>
<tr>
<td><code>text</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext</code></td>
<td>Alias for <code>text()</code> with retaintokenisation=True</td>
</tr>
<tr>
<td><code>updatetext</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml</code></td>
<td>See AbstractElement.xml()</td>
</tr>
<tr>
<td><code>xmlstring</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

- ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ...)  
- ANNOTATIONTYPE = 31  
- AUTH = True  
- AUTO_GENERATE_ID = False  
- HIDDEN = False  
- LABEL = 'Coreference Link'  

1.11. Annotations
OCCURRENCES = 0
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELETIMITER = None
WREFABLE = False
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=False, 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, None, or False) – 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. Raise a No-
    SuchAnnotation exception if not found.

    Parameters Classes (*+) – The possible classes (AbstractElement or subclasses) to se-
    lect 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 (tuple of) classes (AbstractElement or subclasses).
    Not instances!

    Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
    Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if
    none was found

annotations (Class, set=False)
    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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
    Converts annotator information to processor information (FoLiA v2). Can be called with arguments to
    override defaults.

append (child, *args, **kwargs)
    See AbstractElement.append()

checkdeclaration()
    Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
    declared, raises an exception if not so, or auto-declares the annotation type if need be.

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(**kwargs)
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=False, 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

depthfirstsearch(function)
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

```python
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=False, **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=False)*

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, hidden=False)*

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, hidden=False)*

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:

```python
import json
json.dumps(word.json())
```
Returns `dict`

`layer()`
Return the annotation layer this annotation pertains to

`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

`parsecommonarguments (doc, **kwargs)`
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

`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, hidden=False)`
Get the phonetic representation associated with this element (of the specified class)

The phonetic content will be constructed from child-elements whereever 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 outputed, 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.

• **hidden**(bool) – Include hidden elements, defaults to `False`.

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

`textcontent()`

`phoncontent(cls='current', correctionhandling=1, hidden=False)`

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.

`precedes(other)`

Returns a boolean indicating whether this element precedes the other element

`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’, 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.

```python
classmethod relaxng(self, includechildren=True, extraattrs=None, extraelements=None, origclass=None)
```

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

```python
classmethod relaxng_backwards()
```

Internal helper function for backward compatibility

```python
remove(self, child)
```

Removes the child element

```python
replace(self, 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.

```python
resolveoffsets(self, begin, end, retaintokenisation=True, strictend=True, cls='current')
```

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

```python
resolveword(self, id)
```

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

```python
select(self, Class, set=False, 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 False (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, AlternativeLayers, Suggestion, and folia. Original. These elements and those contained within are never authoriative. 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:

```python
for sense in text.select(folia.Sense, 'cornoeto', True, 
                        [folia.Original, 
```

**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 sually no need to call this directly, invoked implicitly by `copy()`

**setprocessor** *(processor)*

Sets the processor for this element, taking care of adding an annotator in the declarations

**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.

**sort** *(force=False)*

Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

**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, hidden=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 outputed, 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
- **hidden**(bool) – Include hidden elements, defaults to False.

Example:

```python
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, hidden=False)

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.
- **hidden**(bool) – Include hidden elements, defaults to False.

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:

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

__len__()
Returns the number of child elements under the current element.

__str__()
Alias for text()

1.11. Annotations 813

tolia.main.DependencyDependent
class tolia.main.DependencyDependent(doc, *args, **kwargs)
Bases: tolia.main.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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>accepts</code>(Class[, raiseexceptions, parentinstance])</td>
<td></td>
</tr>
<tr>
<td><code>add</code>(child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td><code>addable</code>(parent[, set, raiseexceptions])</td>
<td></td>
</tr>
<tr>
<td><code>addidsuffix</code>(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td><code>adddtoindex</code>([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
<tr>
<td><code>ancestor</code>(*Classes)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td><code>ancestors</code>([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td><code>annotation</code>(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td><code>annotations</code>([Class[, set]])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td><code>annotator2processor</code>([annotator,...])</td>
<td></td>
</tr>
<tr>
<td><code>append</code>(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td><code>checkdeclaration</code>()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declarres the annotation type if need be.</td>
</tr>
<tr>
<td><code>context</code>(size[, placeholder, scope])</td>
<td></td>
</tr>
<tr>
<td><code>copy</code>([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td><code>copychildren</code>([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td><code>correct</code>(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td><code>count</code>(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td><code>deepvalidation</code>()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td><code>depthfirstsearch</code>(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None</td>
</tr>
<tr>
<td><code>description</code>()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td><code>feat</code>(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td><code>findcorrectionhandling</code>(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td><code>findreplaceables</code>(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td><code>generate_id</code>(cls)</td>
<td></td>
</tr>
<tr>
<td><code>getindex</code>(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td><code>getmetadata</code>([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements</td>
</tr>
<tr>
<td><code>gettextdelimiter</code>([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class)</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class)</td>
</tr>
<tr>
<td>incorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>items([founditems])</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement)</td>
</tr>
<tr>
<td>json([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td>layer()</td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td>leftcontext(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td>next([Class, scope, reverse])</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>originaltext([cls])</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td>parsecommonarguments(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td>parsexml(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td>phon([cls, previousdelimiter, strict, ...])</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td>phoncontent([cls, correctionhandling, hidden])</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td>postappend()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td>precedes(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td>previous([Class, scope])</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td>relaxng([includechildren, extraattribs, ...])</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td>relaxng_backwards()</td>
<td>Internal helper function for backward compatibility</td>
</tr>
<tr>
<td>remove(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td>replace(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td>resolveoffsets(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td>resolveword(id)</td>
<td></td>
</tr>
<tr>
<td>rightcontext(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td>select([Class, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td>setdoc(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td>setdocument(doc)</td>
<td>Associate a document with this element.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 71 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setparents()</code></td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><code>setprocessor(processor)</code></td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><code>setspan(*args)</code></td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><code>settext(text[, cls])</code></td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><code>sort([force])</code></td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><code>speech_speaker()</code></td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>strict=True</code></td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with <code>retaintokenisation=True</code></td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmistring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

- `ANNOTATIONTYPE` = None |
- `AUTH` = True |
- `AUTO_GENERATE_ID` = False |
- `HIDDEN` = False |
- `LABEL` = 'Dependent' |
- `OCCURRENCES` = 1 |
- `OCCURRENCES_PER_SET` = 0 |
- `OPTIONAL_ATTRIBS` = (0,)
- `PHONCONTAINER` = False |
- `PRIMARYELEMENT` = False |
- `PRINTABLE` = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELIMITER = None
WREFABLE = False
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=False, 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, None, or False) – The set

• raiseexceptions (bool) – Raise an exception if the element can’t be added?

Returns bool

Raises ValueError

callsuffix (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. Raise a No-
SuchAnnotation exception if not found.

Parameters Classes (+) – The possible classes (AbstractElement or subclasses) to se-
lect 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 (tuple of) classes (AbstractElement or subclasses).
    Not instances!

Yields elements (instances derived from AbstractElement)

annotation (type, set=False)
Will return a single annotation (even if there are multiple). Raises a NoSuchAnnotation exception if
none was found

annotations (Class, set=False)
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.

annotator2processor (annotator=None, annotatortype=None, parentprocessor=None)
Converts annotator information to processor information (FoLiA v2). Can be called with arguments to
override defaults.

append (child, *args, **kwargs)
See AbstractElement.append()

checkdeclaration ()
Internal method (usually no need to call this) that checks whether the element’s annotation type is properly
declared, raises an exception if not so, or auto-declares the annotation type if need be.

count (Class, set=False, 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

depthfirstsearch(function)  
Generic depth first search algorithm using a callback function, continues as long as the callback function returns None

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:

```python
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=False, **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 TEXTDELMITER attribute but may return a customised one instead.

hasannotation(Class, set=False)  
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, hidden=False)  
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, hidden=False*)

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 Nothing

**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*

**layer**()

Return the annotation layer this annotation pertains to

**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 Nothing if no next element is found. Non-authoritative elements are never returned.
Parameters

- **Class (mut)** – 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 (mut)** – 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`.

**parsecommonarguments**(doc, **kwargs)**

Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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

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

Parameters

- **node** – XML Element (mut)

- **doc** – Document (mut)

Returns

An instance of the current Class.

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

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 outputed, 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.

- **hidden**(bool) – Include hidden elements, defaults to False.

Example:

```python
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

phoncontent (cls='current', correctionhandling=1, hidden=False)
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.

precedes (other)
Returns a boolean indicating whether this element precedes the other element

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’, 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.

classmethod relaxng (includechildren=True, extraattrs=None, extraelements=None, origclass=None)
Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)

classmethod relaxng_backwards ()
internal helper function for backward compatibility
**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 to be an alternative.

**be an alternative.**(to) –
See `AbstractElement.append()` for more information and all parameters.

**resolveoffsets** *(begin, end, retaintokenisation=True, strictend=True, cls='current')*
Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive.

**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=False, 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 `False` (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`, `AlternativeLayers`, `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:

```python
...
```

**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()

setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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.

sort (force=False)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

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, hidden=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 outputed, 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

• **hidden** (*bool*) – Include hidden elements, defaults to `False`.

Example:

```python
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, hidden=False)*

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.

• **hidden** (*bool*) – Include hidden elements, defaults to `False`.

**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:

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

**__len__()**
Returns the number of child elements under the current element.

**__str__()**
Alias for text()

---

**folia.main.Headspan**

class folia.main.Headspan(doc, *args, **kwargs)
Bases: folia.main.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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(doc, *args, **kwargs)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>accepts(Class, raiseexceptions, parentinstance)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>add(child, *args, **kwargs)</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addable(parent[, set, raiseexceptions])</td>
<td>Tests whether a new element of this class can be added to the parent.</td>
</tr>
<tr>
<td>addidsuffix(idsuffix[, recursive])</td>
<td>Appends a suffix to this element’s ID, and optionally to all child IDs as well.</td>
</tr>
<tr>
<td>addtoindex([norecurse])</td>
<td>Makes sure this element (and all subelements), are properly added to the index.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor(<em>Classes</em>)</td>
<td>Find the most immediate ancestor of the specified type, multiple classes may be specified.</td>
</tr>
<tr>
<td>ancestors([Class])</td>
<td>Generator yielding all ancestors of this element, effectively back-tracing its path to the root element.</td>
</tr>
<tr>
<td>annotation(type[, set])</td>
<td>Will return a single annotation (even if there are multiple).</td>
</tr>
<tr>
<td>annotations(Class[, set])</td>
<td>Obtain annotations.</td>
</tr>
<tr>
<td>annotator2processor([annotator, ...])</td>
<td>Converts annotator information to processor information (FoLiA v2).</td>
</tr>
<tr>
<td>append(child, *args, **kwargs)</td>
<td>See AbstractElement.append()</td>
</tr>
<tr>
<td>checkdeclaration()</td>
<td>Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.</td>
</tr>
<tr>
<td>context(size[, placeholder, scope])</td>
<td>Returns this word in context, {size} words to the left, the current word, and {size} words to the right.</td>
</tr>
<tr>
<td>copy([newdoc, idsuffix])</td>
<td>Make a deep copy of this element and all its children.</td>
</tr>
<tr>
<td>copychildren([newdoc, idsuffix])</td>
<td>Generator creating a deep copy of the children of this element.</td>
</tr>
<tr>
<td>correct(**kwargs)</td>
<td>Apply a correction (TODO: documentation to be written still)</td>
</tr>
<tr>
<td>count(Class[, set, recursive, ignore, node])</td>
<td>Like AbstractElement.select(), but instead of returning the elements, it merely counts them.</td>
</tr>
<tr>
<td>deepvalidation()</td>
<td>Perform deep validation of this element.</td>
</tr>
<tr>
<td>depthfirstsearch(function)</td>
<td>Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.</td>
</tr>
<tr>
<td>description()</td>
<td>Obtain the description associated with the element.</td>
</tr>
<tr>
<td>feat(subset)</td>
<td>Obtain the feature class value of the specific subset.</td>
</tr>
<tr>
<td>findcorrectionhandling(cls)</td>
<td>Find the proper correctionhandling given a textclass by looking in the underlying corrections where it is reused.</td>
</tr>
<tr>
<td>findreplaceables(parent[, set])</td>
<td>Internal method to find replaceable elements.</td>
</tr>
<tr>
<td>generate_id(cls)</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getindex(child[, recursive, ignore])</td>
<td>Get the index at which an element occurs, recursive by default!</td>
</tr>
<tr>
<td>getmetadata([key])</td>
<td>Get the metadata that applies to this element, automatically inherited from parent elements.</td>
</tr>
<tr>
<td>gettextdelimiter([retaintokenisation])</td>
<td>Return the text delimiter for this class.</td>
</tr>
<tr>
<td>hasannotation(Class[, set])</td>
<td>Returns an integer indicating whether such as annotation exists, and if so, how many.</td>
</tr>
<tr>
<td>hasphon([cls, strict, correctionhandling, ...])</td>
<td>Does this element have phonetic content (of the specified class).</td>
</tr>
<tr>
<td>hastext([cls, strict, correctionhandling, ...])</td>
<td>Does this element have text (of the specified class).</td>
</tr>
<tr>
<td>inincorrection()</td>
<td>Is this element part of a correction? If it is, it returns the Correction element (evaluating to True), otherwise it returns None.</td>
</tr>
<tr>
<td>insert(index, child, *args, **kwargs)</td>
<td>Returns a depth-first flat list of all items below this element (not limited to AbstractElement).</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 72 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>json</strong>([attribs, recurse, ignorelist])</td>
<td>Serialises the FoLiA element and all its contents to a Python dictionary suitable for serialisation to JSON.</td>
</tr>
<tr>
<td><strong>layer</strong>()</td>
<td>Return the annotation layer this annotation pertains to</td>
</tr>
<tr>
<td><strong>leftcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the left context for an element, as a list.</td>
</tr>
<tr>
<td><strong>next</strong>(class, scope, reverse)</td>
<td>Returns the next element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>originaltext</strong>(cls)</td>
<td>Alias for retrieving the original uncorrect text.</td>
</tr>
<tr>
<td><strong>parsecommonarguments</strong>(doc, **kwargs)</td>
<td>Internal function to parse common FoLiA attributes and sets up the instance accordingly.</td>
</tr>
<tr>
<td><strong>parsetxml</strong>(node, doc, **kwargs)</td>
<td>Internal class method used for turning an XML element into an instance of the Class.</td>
</tr>
<tr>
<td><strong>phon</strong>(cls, previousdelimiter, strict, ...)</td>
<td>Get the phonetic representation associated with this element (of the specified class)</td>
</tr>
<tr>
<td><strong>phoncontent</strong>(cls, correctionhandling, hidden)</td>
<td>Get the phonetic content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><strong>postappend</strong>()</td>
<td>This method will be called after an element is added to another and does some checks.</td>
</tr>
<tr>
<td><strong>precedes</strong>(other)</td>
<td>Returns a boolean indicating whether this element precedes the other element</td>
</tr>
<tr>
<td><strong>previous</strong>(class, scope)</td>
<td>Returns the previous element, if it is of the specified type and if it does not cross the boundary of the defined scope.</td>
</tr>
<tr>
<td><strong>relaxng</strong>(includechildren, extraattrs, ...)</td>
<td>Returns a RelaxNG definition for this element (as an XML element (lxml.etree) rather than a string)</td>
</tr>
<tr>
<td><strong>relaxng_backwards</strong>()</td>
<td>internal helper function for backward compatibility</td>
</tr>
<tr>
<td><strong>remove</strong>(child)</td>
<td>Removes the child element</td>
</tr>
<tr>
<td><strong>replace</strong>(child, *args, **kwargs)</td>
<td>Appends a child element like append(), but replaces any existing child element of the same type and set.</td>
</tr>
<tr>
<td><strong>resolveoffsets</strong>(begin, end[, ...])</td>
<td>Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!).</td>
</tr>
<tr>
<td><strong>resolveword</strong>(id)</td>
<td></td>
</tr>
<tr>
<td><strong>rightcontext</strong>(size[, placeholder, scope])</td>
<td>Returns the right context for an element, as a list.</td>
</tr>
<tr>
<td><strong>select</strong>(class[, set, recursive, ignore, node])</td>
<td>Select child elements of the specified class.</td>
</tr>
<tr>
<td><strong>setdoc</strong>(newdoc)</td>
<td>Set a different document.</td>
</tr>
<tr>
<td><strong>setdocument</strong>(doc)</td>
<td>Associate a document with this element.</td>
</tr>
<tr>
<td><strong>setparents</strong>()</td>
<td>Correct all parent relations for elements within the scop.</td>
</tr>
<tr>
<td><strong>setprocessor</strong>(processor)</td>
<td>Sets the processor for this element, taking care of adding an annotator in the declarations</td>
</tr>
<tr>
<td><strong>setspan</strong>(args)</td>
<td>Sets the span of the span element anew, erases all data inside.</td>
</tr>
<tr>
<td><strong>settext</strong>(text[, cls])</td>
<td>Set the text for this element.</td>
</tr>
<tr>
<td><strong>sort</strong>(force)</td>
<td>Sort children (wrefs and child spans) in order of appearance.</td>
</tr>
<tr>
<td><strong>speech_speaker</strong>()</td>
<td>Retrieves the speaker of the audio or video file associated with the element.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>speech_src()</code></td>
<td>Retrieves the URL/filename of the audio or video file associated with the element.</td>
</tr>
<tr>
<td><code>stricttext([cls])</code></td>
<td>Alias for <code>text()</code> with strict=True</td>
</tr>
<tr>
<td><code>text([cls, retaintokenisation, ...])</code></td>
<td>Get the text associated with this element (of the specified class)</td>
</tr>
<tr>
<td><code>textcontent([cls, correctionhandling, hidden])</code></td>
<td>Get the text content explicitly associated with this element (of the specified class).</td>
</tr>
<tr>
<td><code>textvalidation([warnonly])</code></td>
<td>Run text validation on this element.</td>
</tr>
<tr>
<td><code>toktext([cls])</code></td>
<td>Alias for <code>text()</code> with retaintokenisation=True</td>
</tr>
<tr>
<td><code>updatetext()</code></td>
<td>Recompute textual value based on the text content of the children.</td>
</tr>
<tr>
<td><code>wrefs([index, recurse])</code></td>
<td>Returns a list of word references, these can be Words but also Morphemes or Phonemes.</td>
</tr>
<tr>
<td><code>xml([attribs, elements, skipchildren])</code></td>
<td>See <code>AbstractElement.xml()</code></td>
</tr>
<tr>
<td><code>xmlstring([pretty_print])</code></td>
<td>Serialises this FoLiA element and all its contents to XML.</td>
</tr>
<tr>
<td><code>__iter__()</code></td>
<td>Iterate over all children of this element.</td>
</tr>
<tr>
<td><code>__len__()</code></td>
<td>Returns the number of child elements under the current element.</td>
</tr>
<tr>
<td><code>__str__()</code></td>
<td>Alias for <code>text()</code></td>
</tr>
</tbody>
</table>

### Class Attributes

```
ACCEPTED_DATA = (<class 'folia.main.AbstractInlineAnnotation'>, <class 'folia.main.Comment'>, ...
ANNOTATIONTYPE = None
AUTH = True
AUTO_GENERATE_ID = False
HIDDEN = False
LABEL = 'Head'
OCCURRENCES = 1
OCCURRENCES_PER_SET = 0
OPTIONAL_ATTRIBS = (0,)
PHONCONTAINER = False
PRIMARYELEMENT = False
PRINTABLE = True
REQUIRED_ATTRIBS = None
REQUIRED_DATA = None
SETONLY = False
SPEAKABLE = True
SUBSET = None
TEXTCONTAINER = False
TEXTDELEIMITER = None
```

1.11. Annotations
WRETABLE = False
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=False, 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, None, or False) – 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. Raise a No-SuchAnnotation exception if not found.

    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 (tuple of) classes (AbstractElement or subclasses).
    Not instances!

    Yields elements (instances derived from AbstractElement)
**annotation** *(type, set=False)*

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

**annotations** *(Class, set=False)*

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.

**annotator2processor** *(annotator=None, annotatortype=None, parentprocessor=None)*

Converts annotator information to processor information (FoLiA v2). Can be called with arguments to override defaults.

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

See `AbstractElement.append()`.

**checkdeclaration**()

Internal method (usually no need to call this) that checks whether the element’s annotation type is properly declared, raises an exception if not so, or auto-declares the annotation type if need be.

**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=False, 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** DeepValidationException

**depthfirstsearch**(function)

Generic depth first search algorithm using a callback function, continues as long as the callback function returns None.
**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:

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

**Returns** str or list

**findcorrectionhandling(cls)**

Find the proper correction handling given a text class by looking in the underlying corrections where it is reused.

**classmethod findreplaceables(parent, set=False, **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=False)**

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, hidden=False)**

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, hidden=False)

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:

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

Returns dict

layer ()

Return the annotation layer this annotation pertains to

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

parsecommonarguments (doc, **kwargs)
Internal function to parse common FoLiA attributes and sets up the instance accordingly. Do not invoke directly.

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, hidden=False)
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 outputed, 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.
- hidden (bool) – Include hidden elements, defaults to False.

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 ()
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()`

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.

**precedes**(other)

Returns a boolean indicating whether this element precedes the other element

**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’, 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.

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

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

**classmethod relaxng_backwards()**

internal helper function for backward compatibility

**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()
• **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 to be an alternative.

• **be an alternative.**(to) – See *AbstractElement.append()* for more information and all parameters.

`resolveoffsets(begin, end, retaintokenisation=True, strictend=True, cls='current')`

Resolves supplied character offset information and returns tokens (non-token structures like linebreaks etc are ignored!). Note: offsets are zero-indexed and the end is non-inclusive!

`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=False, recursive=True, ignore=True, node=None)`

Select child elements of the specified class. A further restriction can be made based on set.

- **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 False (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, AlternativeLayers, 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:

```python
 ...
```

`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()*
setprocessor (processor)
Sets the processor for this element, taking care of adding an annotator in the declarations

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.

sort (force=False)
Sort children (wrefs and child spans) in order of appearance. Returns True if sort is successful (or not needed), False if sort could not be performed at this stage

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, hidden=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 outputed, 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

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

Example:
```python
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, hidden=False)*

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.

• **hidden** *(bool)* – Include hidden elements, defaults to `False`.

**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`

**updatecontext** *

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:

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

__len__()

Returns the number of child elements under the current element.

__str__()

Alias for text()
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:

```python
doc = folia.Document(id='example')
```

2.2 Declarations

Whenever you add a new type of annotation, no matter whether linguistic, structural or otherwise, you need to declare it. Now this FoLiA library is capable of automatically declaring annotations as you go along as long as there is no ambiguity, so you won’t always need to do this explicitly, but it is still important to understand what is going on as you will run into it eventually.

Declarations are made using the `Document.declare()` method. For example, do you want to use paragraphs in your document? Declare it. The simplest form of declaration looks as follows:

```python
doc.declare(folia.Paragraph)
```

In your declaration you can associate a set with the annotation type, we do this in the second parameter to `Document.declare()` (for various annotation types this is mandatory even because without this there can be no classes for the annotations). The set defines the vocabulary that is used, the declaration points to a URL where this set is hosted, but don’t worry about this too much yet:

```python
doc.declare(folia.PosAnnotation, 'http://somewhere/brown-tag-set')
```

At this point, you may also include information about who or what performed this type of annotation. For instance, your program or script. We call this provenance information, and each annotator is added through a processor, an instance of `Processor`, passed as argument to the `Document.declare()` method:
If you want to add another processor, simply call the declare method again, it will still result in only one declaration, but this will be tied to multiple processors:

```python
othertagger = doc.declare(folia.PosAnnotation, 'http://some/path/brown-tag-set', Processor(name="othertagger") )
```

As shown in the above example, the `Document.declare()` method will actually return the `Processor` instance, which is useful if you have multiple, as each processor will automatically (unless you specify it explicitly) get assigned an ID, which you can pass to individual annotations to associate your annotation with a particular processor. This will be illustrated later.

You’re not limited to just using one set, simply call declare with another set to add another declaration:

```python
doc.declare(folia.PosAnnotation, 'http://some/path/cgn-tag-set' )
```

To check if a particular annotation type and set is declared, use the `Document.declared()` method.

### 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:

```python
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 inline annotation. Using `append()` will be faster though.

### 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:

```python
#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')
```

(continues on next page)
# 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:

```python
# 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:

```python
Class(document, *children, **kwaargs)
```

**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`
- `processor`
- `annotator`
- `annotatortype`
- `confidence`
- `src`
- `speaker`
- `begintime`
- `endtime`

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 if often aesthetically unpleasing for our purposes. Example of this use case will be shown in the next section.

### 2.5 Provenance Information

We already introduced the concept of provenance in the section on *Declarations*, provenance data clarifies what the origin of an annotation is, i.e. who or what annotated it. If you declared an annotation type with a single processor, then it will automatically act as the default for annotations of that type (and set). If, however, you have multiple processors for a given annotation type and set (and it’s good practise to always assume this), you should make this explicit when adding the annotation, using the `processor` attribute:

```python
# First we declare the annotation type with a processor
posprocessor = doc.declare(folia.PosAnnotation, set='brown-tagset', processor=Processor(name="mypostagger"))

# Then we add an annotation to our word
word.add(folia.PosAnnotation, set='brown-tagset', cls='n', processor=posprocessor)
```

The processor attribute takes an instance of `Processor`, or the ID (not the name!) of an existing processor. If the processor has not been declared yet, the library will do that for you automatically.

You can iterate over the entire provenance chain of a document `doc` by iterating over `doc.provenance` (Provenance). To get a specific processor by ID: `doc.provenance[id]`.

Instead of explicitly assigning a processor with individual annotations, you can do so implicitly by associating a processor with the `Document`. It will then be automatically be used for any subsequent annotations you add. You can associate a processor immediately upon document instantiation:

```python
doc = folia.Document(file="/tmp/example.folia.xml", processor=Processor(name="myscript", version="0.1"))
```

Instead of using `Processor()`, you instantiate one using `Processor.create()` which will autodetect a lot of information regarding your processor for you, such as the system you’re running on, command that was executed, date & time, etc...:

```python
doc = folia.Document(file="/tmp/example.folia.xml", processor=Processor.create(name="myscript", version="0.1"))
```

You can also associate a processor after instantiation (useful in case you want to use different processors at different points in your script), but in that case you need to make sure to append it to the provenance chain yourself:

```python
doc.processor = Processor.create(name="myscript", version="0.1")
doc.provenance.append(doc.processor)
```

Unsetting a main processor is done using a simple `doc.processor = None`. 

```python
844 Chapter 2. Editing FoLiA
```
2.6 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:

```python
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:

```python
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:

```python
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
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='det'),
    folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.4'], cls='np'),
    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='n'),
]),
```

(continues on next page)
Note: The lower-level `AbstractElement.append()` method would have had the same effect in the above syntax tree sample.

### 2.7 Deleting annotations

Any element can be deleted by calling the `AbstractElement.remove()` method on its parent. Suppose we want to delete `word`:

```python
text = folia.SyntacticUnit(self.doc, self.doc['example.s.1.w.6'], cls='fin')
```  

```python
word.parent.remove(word)
```  

### 2.8 Copying annotations

A *deep copy* can be made of any element by calling its `AbstractElement.copy()` method:

```python
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:

```python
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:

```python
word2 = word.copy(doc, "copy")
```
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:

```python
from folia import fql
from pynlpl.formats import cql
```

### 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 University, 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”:

```python
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.

## 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
```

### 3.2.1 `folia.fql.Query`

```python
class folia.fql.Query(q, context=<folia.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:

```python
gquery = 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:

```python
gquery = 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:

```python
gquery = 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:

```python
gquery = 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:

```python
gquery = fql.Query('SELECT entity WHERE class = "person" FOR w WHERE text != "John"
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:

```python
gquery = 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 EDT, ADD, APPEND and PREPEND.

**Note:** Consult the FQL documentation at [https://github.com/proycon/foliadocserve/blob/master/README.rst](https://github.com/proycon/foliadocserve/blob/master/README.rst) for further documentation on the language.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(ql, context)</td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>parse</code>(ql, i)</td>
<td>Selects and if necessary adds the necessary processor or (nested) processors</td>
</tr>
<tr>
<td><code>setprocessor</code>(doc, processor, debug)</td>
<td></td>
</tr>
</tbody>
</table>

3.2. FoLiA Query Language (FQL) 849
Method Details

__init__(q, context=<folia.fql.Context object>)
Initialize self. See help(type(self)) for accurate signature.

__init__(q, context=<folia.fql.Context object>)
Initialize self. See help(type(self)) for accurate signature.

parse(q, i=0)

setprocessor (doc, processor, debug=False)
Selects and if necessary adds the necessary processor or (nested) processors

Selecting a word with a particular text is done as follows:

```python
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:

```python
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:

```python
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:

```python
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:

```python
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:

```python
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:

```python
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.

### 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:

```python
Reader
```

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)
```
4.1 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:

- TextMarkupGap
- TextMarkupString
- TextMarkupStyle
- TextMarkupCorrection
- TextMarkupError

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.feat() 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:

```python
pos = word.annotation(folia.PosAnnotation)
if pos.cls == "n":
    if pos.feat('number') == 'plural':
        print("We have a plural noun!")
    elif pos.feat('number') == 'singular':
        print("We have a singular noun!")
```

The AbstractElement.feat() 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:

```python
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:

- `SynsetFeature`
- `ActorFeature`
- `BegindatetimeFeature`
- `EnddatetimeFeature`

## 4.3 Alternatives

A key feature of FoLiA is its ability to make explicit alternative annotations, for inline 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:

```python
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:

```python
alt = sentence.add(folia.AlternativeLayers)
layers = 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:

- `Alternative`
- `AlternativeLayers`
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, inline 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`:

```python
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:

```python
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:

```python
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:

```plaintext
Correction
Current
ErrorDetection
New
Original
```

Continued on next page
4.5 Relations

Relations 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:

Relation
LinkReference

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:

Description
Metric
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:

```python
doc = folia.Document(file="/path/to/document.xml")
doc.metadata['language'] = "en"
```
CHAPTER 6

Indices and tables

- genindex
- modindex
- search
f

folia.main.1
Symbols

__init__() (folia.fql.Query method), 850
__init__() (folia.main.AbstractAnnotationLayer method), 59
__init__() (folia.main.AbstractElement method), 19
__init__() (folia.main.AbstractSpanAnnotation method), 46
__init__() (folia.main.AbstractStructureElement method), 32
__init__() (folia.main.AbstractTextMarkup method), 71
__init__() (folia.main.Cell method), 86
__init__() (folia.main.Chunk method), 537
__init__() (folia.main.ChunkingLayer method), 679
__init__() (folia.main.CoreferenceChain method), 550
__init__() (folia.main.CoreferenceLayer method), 691
__init__() (folia.main.CoreferenceLink method), 804
__init__() (folia.main.Definition method), 100
__init__() (folia.main.DependenciesLayer method), 704
__init__() (folia.main.Dependency method), 563
__init__() (folia.main.DependencyDependent method), 817
__init__() (folia.main.Division method), 114
__init__() (folia.main.Document method), 5, 7
__init__() (folia.main.DomainAnnotation method), 440
__init__() (folia.main.EntitiesLayer method), 716
__init__() (folia.main.Entity method), 576
__init__() (folia.main.Entry method), 128
__init__() (folia.main.Event method), 142
__init__() (folia.main.Example method), 156
__init__() (folia.main.Figure method), 170
__init__() (folia.main.Gap method), 184
__init__() (folia.main.Head method), 196
__init__() (folia.main.Headspan method), 830
__init__() (folia.main.LangAnnotation method), 464
__init__() (folia.main.LemmaAnnotation method), 476
__init__() (folia.main.Linebreak method), 210
__init__() (folia.main.List method), 224
__init__() (folia.main.ListItem method), 238
__init__() (folia.main.Note method), 252
__init__() (folia.main.Observation method), 589
__init__() (folia.main.ObservationLayer method), 729
__init__() (folia.main.Paragraph method), 266
__init__() (folia.main.Part method), 280
__init__() (folia.main.PhonContent method), 524
__init__() (folia.main.Predicate method), 602
__init__() (folia.main.Quote method), 294
__init__() (folia.main.Reference method), 308
__init__() (folia.main.Row method), 322
__init__() (folia.main.SemanticRole method), 653
__init__() (folia.main.SemanticRolesLayer method), 779
__init__() (folia.main.SenseAnnotation method), 488
__init__() (folia.main.Sentiment method), 337
__init__() (folia.main.SentimentLayer method), 741
__init__() (folia.main.Statement method), 627
__init__() (folia.main.StatementLayer method), 754
__init__() (folia.main.SubjectivityAnnotation method), 500
__init__() (folia.main.SyntacticUnit method), 640
__init__() (folia.main.SyntaxLayer method), 766
__init__() (folia.main.Table method), 352
__init__() (folia.main.TableHead method), 380
__init__() (folia.main.Term method), 366
__init__() (folia.main.Text method), 394
__init__() (folia.main.TextContent method), 513
__init__() (folia.main TimingLayer method), 791
__init__() (folia.mainWhiteSpace method), 408
__init__() (folia.main.Word method), 423
Index

**FoLiA Python Library Documentation, Release v2.2.2, FoLiA v2.2.1**

__len__() (folia.main.Head method), 207
__len__() (folia.main.Headspan method), 839
__len__() (folia.main.LangAnnotation method), 473
__len__() (folia.main.LemmaAnnotation method), 485
__len__() (folia.main.Linebreak method), 221
__len__() (folia.main.List method), 235
__len__() (folia.main.ListItem method), 249
__len__() (folia.main.Note method), 263
__len__() (folia.main.Observation method), 598
__len__() (folia.main.ObservationLayer method), 738
__len__() (folia.main.Paragraph method), 277
__len__() (folia.main.Part method), 291
__len__() (folia.main.PhonContent method), 532
__len__() (folia.main.PosAnnotation method), 461
__len__() (folia.main.Predicate method), 611
__len__() (folia.main.Quote method), 305
__len__() (folia.main.Reference method), 319
__len__() (folia.main.Row method), 333
__len__() (folia.main.SemanticRole method), 662
__len__() (folia.main.SemanticRolesLayer method), 788
__len__() (folia.main.Sentence method), 348
__len__() (folia.main.Sentiment method), 623
__len__() (folia.main.SentimentLayer method), 750
__len__() (folia.main.Statement method), 636
__len__() (folia.main.StatementLayer method), 763
__len__() (folia.main.SubjectivityAnnotation method), 509
__len__() (folia.main.SyntacticUnit method), 649
__len__() (folia.main.SyntaxLayer method), 775
__len__() (folia.main.Table method), 362
__len__() (folia.main.TableHead method), 390
__len__() (folia.main.Term method), 376
__len__() (folia.main.Text method), 404
__len__() (folia.main.TextContent method), 520
__len__() (folia.main.TimeSegment method), 674
__len__() (folia.main.TimingLayer method), 800
__len__() (folia.main.Whitespace method), 418
__len__() (folia.main.Word method), 435
__str__() (folia.main.AbstractAnnotationLayer method), 68
__str__() (folia.main.AbstractElement method), 28
__str__() (folia.main.AbstractSpanAnnotation method), 55
__str__() (folia.main.AbstractStructureElement method), 42
__str__() (folia.main.AbstractTextMarkup method), 79
__str__() (folia.main.Cell method), 97
__str__() (folia.main.Chunk method), 547
__str__() (folia.main.ChunkingLayer method), 688
__str__() (folia.main.CoreferenceChain method), 560
__str__() (folia.main.CoreferenceLayer method), 700
__str__() (folia.main.CoreferenceLink method), 813
__str__() (folia.main.Definition method), 111
__str__() (folia.main.DependenciesLayer method), 713
__str__() (folia.main.Dependency method), 572
__str__() (folia.main.DependencyDependent method), 826
__str__() (folia.main.Division method), 125
__str__() (folia.main.DonationAnnotation method), 449
__str__() (folia.main.EntitiesLayer method), 725
__str__() (folia.main.Entity method), 585
__str__() (folia.main.Entry method), 139
__str__() (folia.main.Event method), 153
__str__() (folia.main.Example method), 167
__str__() (folia.main.Figure method), 181
__str__() (folia.main.Gap method), 193
__str__() (folia.main.Head method), 207
__str__() (folia.main.Headspan method), 839
__str__() (folia.main.LangAnnotation method), 473
__str__() (folia.main.LemmaAnnotation method), 485
__str__() (folia.main.LivingLayer method), 738
__str__() (folia.main.Paragraph method), 277
__str__() (folia.main.Part method), 291
__str__() (folia.main.PosAnnotation method), 461
__str__() (folia.main.Predicate method), 611
__str__() (folia.main.Quote method), 305
__str__() (folia.main.Reference method), 319
__str__() (folia.main.Row method), 333
__str__() (folia.main.SemanticRole method), 662
__str__() (folia.main.SemanticRolesLayer method), 788
__str__() (folia.main.Sentence method), 348
__str__() (folia.main.Sentiment method), 623
__str__() (folia.main.SentimentLayer method), 750
__str__() (folia.main.Statement method), 636
__str__() (folia.main.StatementLayer method), 763
__str__() (folia.main.SubjectivityAnnotation method), 509
__str__() (folia.main.SyntacticUnit method), 649
__str__() (folia.main.SyntaxLayer method), 775
__str__() (folia.main.Table method), 362
__str__() (folia.main.TableHead method), 390
__str__() (folia.main.Term method), 376
__str__() (folia.main.Text method), 404
__str__() (folia.main.TextContent method), 520
__str__() (folia.main.TimeSegment method), 674
__str__() (folia.main.TimingLayer method), 800
__str__() (folia.main.Whitespace method), 418
__str__() (folia.main.Word method), 435
__str__() (folia.main.AbsentElement method), 28
__str__() (folia.main.AbsentSpanAnnotation method), 55
__str__() (folia.main.AbsentStructureElement method), 42
__str__() (folia.main.AbsentTextMarkup method), 79
__str__() (folia.main.Cell method), 97
__str__() (folia.main.Chunk method), 547
__str__() (folia.main.ChunkingLayer method), 688
__str__() (folia.main.CoreferenceChain method), 560
__str__() (folia.main.CoreferenceLayer method), 700
__str__() (folia.main.CoreferenceLink method), 813
__str__() (folia.main.Definition method), 111
__str__() (folia.main.DependenciesLayer method), 713
__str__() (folia.main.Dependency method), 572
__str__() (folia.main.DependencyDependent method), 826
__str__() (folia.main.Division method), 125
__str__() (folia.main.DonationAnnotation method), 449
__str__() (folia.main.EntitiesLayer method), 725
__str__() (folia.main.Entity method), 585
__str__() (folia.main.Entry method), 139
__str__() (folia.main.Event method), 153
__str__() (folia.main.Example method), 167
__str__() (folia.main.Figure method), 181
__str__() (folia.main.Gap method), 193
__str__() (folia.main.Head method), 207
__str__() (folia.main.Headspan method), 839
__str__() (folia.main.LangAnnotation method), 473
__str__() (folia.main.LemmaAnnotation method), 485
__str__() (folia.main.LivingLayer method), 738
__str__() (folia.main.Paragraph method), 277
__str__() (folia.main.Part method), 291
__str__() (folia.main.PosAnnotation method), 461
__str__() (folia.main.Predicate method), 611
__str__() (folia.main.Quote method), 305
__str__() (folia.main.Reference method), 319
__str__() (folia.main.Row method), 333
__str__() (folia.main.SemanticRole method), 662
__str__() (folia.main.SemanticRolesLayer method), 788
__str__() (folia.main.Sentence method), 348
__str__() (folia.main.Sentiment method), 623
__str__() (folia.main.SentimentLayer method), 750
__str__() (folia.main.Statement method), 636
__str__() (folia.main.StatementLayer method), 763
__str__() (folia.main.SubjectivityAnnotation method), 509
__str__() (folia.main.SyntacticUnit method), 649
__str__() (folia.main.SyntaxLayer method), 775
__str__() (folia.main.Table method), 362
__str__() (folia.main.TableHead method), 390

__str__() (folia.main.Term method), 376
__str__() (folia.main.Text method), 404
__str__() (folia.main.TextContent method), 521
__str__() (folia.main.TimeSegment method), 675
__str__() (folia.main.TimingLayer method), 800
__str__() (folia.main.Whitespace method), 418
__str__() (folia.main.Word method), 435

A
AbstractAnnotationLayer (class in folia.main), 55
AbstractElement (class in folia.main), 15
AbstractSpanAnnotation (class in folia.main), 42
AbstractStructureElement (class in folia.main), 28
AbstractTextMarkup (class in folia.main), 68
ACCEPTED_DATA (folia.main.AbstractAnnotationLayer attribute), 58
ACCEPTED_DATA (folia.main.AbstractElement attribute), 19
ACCEPTED_DATA (folia.main.AbstractSpanAnnotation attribute), 45
ACCEPTED_DATA (folia.main.AbstractStructureElement attribute), 31
ACCEPTED_DATA (folia.main.AbstractTextMarkup attribute), 70
ACCEPTED_DATA (folia.main.Cell attribute), 86
ACCEPTED_DATA (folia.main.Chunk attribute), 537
ACCEPTED_DATA (folia.main.ChunkingLayer attribute), 678
ACCEPTED_DATA (folia.main.CoreferenceChain attribute), 550
ACCEPTED_DATA (folia.main.CoreferenceLayer attribute), 690
ACCEPTED_DATA (folia.main.CoreferenceLink attribute), 803
ACCEPTED_DATA (folia.main.Definition attribute), 100
ACCEPTED_DATA (folia.main.DependenciesLayer attribute), 703
ACCEPTED_DATA (folia.main.Dependency attribute), 562
ACCEPTED_DATA (folia.main.DependencyDependent attribute), 816
ACCEPTED_DATA (folia.main.Division attribute), 114
ACCEPTED_DATA (folia.main.DomainAnnotation attribute), 440
ACCEPTED_DATA (folia.main.EntitiesLayer attribute), 716
ACCEPTED_DATA (folia.main.Entity attribute), 575
ACCEPTED_DATA (folia.main.Entry attribute), 128
ACCEPTED_DATA (folia.main.Event attribute), 142
ACCEPTED_DATA (folia.main.Example attribute), 156
ACCEPTED_DATA (folia.main.Figure attribute), 170
ACCEPTED_DATA (folia.main.Gap attribute), 183
ACCEPTED_DATA (folia.main.Head attribute), 196
ACCEPTED_DATA (folia.main.Headspan attribute), 829
ACCEPTED_DATA (folia.main.LangAnnotation attribute), 464
ACCEPTED_DATA (folia.main.LemmaAnnotation attribute), 476
ACCEPTED_DATA (folia.main.Linebreak attribute), 210
ACCEPTED_DATA (folia.main.List attribute), 224
ACCEPTED_DATA (folia.main.ListItem attribute), 238
ACCEPTED_DATA (folia.main.Note attribute), 252
ACCEPTED_DATA (folia.main.Observation attribute), 588
ACCEPTED_DATA (folia.main.ObservationLayer attribute), 728
ACCEPTED_DATA (folia.main.Paragraph attribute), 266
ACCEPTED_DATA (folia.main.Part attribute), 280
ACCEPTED_DATA (folia.main.PhonContent attribute), 523
ACCEPTED_DATA (folia.main.PosAnnotation attribute), 452
ACCEPTED_DATA (folia.main.Predicate attribute), 601
ACCEPTED_DATA (folia.main.Quote attribute), 294
ACCEPTED_DATA (folia.main.Reference attribute), 308
ACCEPTED_DATA (folia.main.Row attribute), 322
ACCEPTED_DATA (folia.main.SentimentLayer attribute), 652
ACCEPTED_DATA (folia.main.SemanticRole attribute), 778
ACCEPTED_DATA (folia.main.SenseAnnotation attribute), 488
ACCEPTED_DATA (folia.main.Sentence attribute), 336
ACCEPTED_DATA (folia.main.Sentiment attribute), 614
ACCEPTED_DATA (folia.main.SentimentLayer attribute), 741
ACCEPTED_DATA (folia.main.Statement attribute), 626
ACCEPTED_DATA (folia.main.StatementLayer attribute), 753
ACCEPTED_DATA (folia.main.SubjectivityAnnotation attribute), 500
ACCEPTED_DATA (folia.main.SyntacticUnit attribute), 639
ACCEPTED_DATA (folia.main.SyntaxLayer attribute), 766
ACCEPTED_DATA (folia.main.Table attribute), 351
ACCEPTED_DATA (folia.main.TableHead attribute), 379
ACCEPTED_DATA (folia.main.Term attribute), 365
ACCEPTED_DATA (folia.main.Text attribute), 393
ACCEPTED_DATA (folia.main.TextContent attribute), 512
ACCEPTED_DATA (folia.main.TimeSegment attribute), 665
ACCEPTED_DATA (folia.main.TimingLayer attribute), 791
ACCEPTED_DATA (folio.main.Whitespace attribute), 407
ACCEPTED_DATA (folio.main.Word attribute), 422
accepts() (folio.main.AbstractAnnotationLayer class method), 59
accepts() (folio.main.AbstractElement class method), 19
accepts() (folio.main.AbstractSpanAnnotation class method), 46
accepts() (folio.main.AbstractStructureElement class method), 32
accepts() (folio.main.AbstractTextMarkup class method), 71
accepts() (folio.main.Cell class method), 86
accepts() (folio.main.Chunk class method), 538
accepts() (folio.main.ChunkingLayer class method), 679
accepts() (folio.main.CoreferenceChain class method), 550
accepts() (folio.main.CoreferenceLayer class method), 691
accepts() (folio.main.CoreferenceLink class method), 804
accepts() (folio.main.Definition class method), 100
accepts() (folio.main.DependenciesLayer class method), 704
accepts() (folio.main.Dependency class method), 563
accepts() (folio.main.DependencyDependent class method), 817
accepts() (folio.main.Division class method), 114
accepts() (folio.main.DomainAnnotation class method), 440
accepts() (folio.main.EntitiesLayer class method), 716
accepts() (folio.main.Entity class method), 576
accepts() (folio.main.Entry class method), 128
accepts() (folio.main.Event class method), 142
accepts() (folio.main.Example class method), 156
accepts() (folio.main.Figure class method), 170
accepts() (folio.main.Gap class method), 184
accepts() (folio.main.Head class method), 196
accepts() (folio.main.Headspan class method), 830
accepts() (folio.main.LangAnnotation class method), 464
accepts() (folio.main.LemmaAnnotation class method), 476
accepts() (folio.main.Linebreak class method), 211
accepts() (folio.main.List class method), 224
accepts() (folio.main.ListItem class method), 238
accepts() (folio.main.Note class method), 252
accepts() (folio.main.Observation class method), 589
accepts() (folio.main.ObservationLayer class method), 729
accepts() (folio.main.Paragraph class method), 266
accepts() (folio.main.Part class method), 280
accepts() (folio.main.Phrase class method), 524
accepts() (folio.main.PhraseMarker class method), 452
accepts() (folio.main.Predicate class method), 602
accepts() (folio.main.PredicateAttribute class method), 294
accepts() (folio.main.Rectangle class method), 308
accepts() (folio.main.Row class method), 322
accepts() (folio.main.Sentence class method), 653
accepts() (folio.main.SentenceLayer class method), 779
accepts() (folio.main.Sense class method), 488
accepts() (folio.main.Sentence class method), 337
accepts() (folio.main.Sentence class method), 614
accepts() (folio.main.SentimentLayer class method), 741
accepts() (folio.main.Statement class method), 627
accepts() (folio.main.StatementLayer class method), 754
accepts() (folio.main.SubjectivityAnnotation class method), 500
accepts() (folio.main.SyntacticUnit class method), 640
accepts() (folio.main.SyntaxLayer class method), 766
accepts() (folio.main.Table class method), 352
accepts() (folio.main.TableHead class method), 380
accepts() (folio.main.Term class method), 366
accepts() (folio.main.Text class method), 394
accepts() (folio.main.TextContent class method), 513
accepts() (folio.main.TimeSegment class method), 665
accepts() (folio.main.TimingLayer class method), 791
accepts() (folio.main.Whitespace class method), 408
accepts() (folio.main.Word class method), 423
add() (folio.main.AbstractAnnotationLayer method), 408
add() (folio.main.AbstractElement method), 59
add() (folio.main.AbstractSpanAnnotation method), 46
add() (folio.main.AbstractStructureElement method), 32
add() (folio.main.AbstractTextMarkup method), 71
add() (folio.main.Cell method), 86
add() (folio.main.Chunk method), 538
add() (folio.main.ChunkingLayer method), 679
add() (folio.main.CoreferenceChain method), 550
add() (folio.main.CoreferenceLayer method), 691
add() (folio.main.CoreferenceLink method), 804
add() (folio.main.Definition method), 100
add() (folio.main.DependenciesLayer method), 704
add() (folio.main.Dependency method), 563
add() (folio.main.DependencyDependent method), 817
add() (folio.main.Duration class method), 114
add() (folia.main.Division method), 115
add() (folia.main.Document method), 8
add() (folia.main.DomainAnnotation method), 440
add() (folia.main.EntitiesLayer method), 716
add() (folia.main.Entity method), 576
add() (folia.main.Entry method), 129
add() (folia.main.Event method), 142
add() (folia.main.Example method), 156
add() (folia.main.Figure method), 170
add() (folia.main.Gap method), 184
add() (folia.main.Head method), 197
add() (folia.main.Headspan method), 830
add() (folia.main.LangAnnotation method), 464
add() (folia.main.Linebreak method), 211
add() (folia.main.List method), 224
add() (folia.main.ListItem method), 238
add() (folia.main.Note method), 252
add() (folia.main.Observation method), 589
add() (folia.main.ObservationLayer method), 716
add() (folia.main.Paragraph method), 266
add() (folia.main.Part method), 280
add() (folia.main.Phrase method), 524
add() (folia.main.Predicate method), 602
add() (folia.main.Predicate class method), 294
add() (folia.main.Reference method), 309
add() (folia.main.Row method), 322
add() (folia.main.Sentiment method), 614
add() (folia.main.SentimentLayer method), 716
add() (folia.main.Statement method), 716
add() (folia.main.StatementLayer method), 741
add() (folia.main.SubjectivityAnnotation method), 500
add() (folia.main.SyntaxicUnit method), 640
add() (folia.main.SyntaxLayer method), 766
add() (folia.main.Table method), 352
add() (folia.main.TableHead method), 380
add() (folia.main.Term method), 366
add() (folia.main.Text method), 394
add() (folia.main.TextContent method), 513
add() (folia.main.TimeSegment method), 665
add() (folia.main.TimingLayer method), 791
add() (folia.mainWhitespace method), 408
add() (folia.main.Word method), 423
addable() (folia.main.AbstractAnnotationLayer class method), 59
addable() (folia.main.AbstractElement class method), 20
addable() (folia.main.AbstractSpanAnnotation class method), 46
addable() (folia.main.AbstractStructureElement class method), 32
addable() (folia.main.AbstractTextMarkup class method), 71
addable() (folia.main.Cell class method), 86
addable() (folia.main.Chunk class method), 538
addable() (folia.main.ChunkingLayer class method), 679
addable() (folia.main.CoreferenceChain class method), 550
addable() (folia.main.CoreferenceLayer class method), 691
addable() (folia.main.CoreferenceLink class method), 804
addable() (folia.main.Definition class method), 100
addable() (folia.main.DependenciesLayer class method), 704
addable() (folia.main.Dependency class method), 563
addable() (folia.main.DependencyDependent class method), 817
addable() (folia.main.Division class method), 115
addable() (folia.main.DomainAnnotation class method), 440
addable() (folia.main.EntitiesLayer class method), 716
addable() (folia.main.Entity class method), 576
addable() (folia.main.Entity class method), 129
addable() (folia.main.Event class method), 142
addable() (folia.main.Example class method), 156
addable() (folia.main.Figure class method), 170
addable() (folia.main.Gap class method), 184
addable() (folia.main.Head class method), 197
addable() (folia.main.Headspan class method), 830
addable() (folia.main.LangAnnotation class method), 464
addable() (folia.main.LemmaAnnotation class method), 476
addable() (folia.main.Linebreak class method), 211
addable() (folia.main.List class method), 224
addable() (folia.main.ListItem class method), 238
addable() (folia.main.Note class method), 252
addable() (folia.main.Observation class method), 589
addable() (folia.main.ObservationLayer class method), 716
addable() (folia.main.Paragraph class method), 266
addable() (folia.main.Part class method), 280
addable() (folia.main.PosAnnotation method), 524
addable() (folia.main.Predicate method), 602
addable() (folia.main.Predicate class method), 294
addable() (folia.main.Reference class method), 309
addable() (folia.main.Row class method), 322
addable() (folia.main.SyntaxicUnit method), 640
addable() (folia.main.SyntaxLayer method), 766
addable() (folia.main.Table method), 352
addable() (folia.main.TableHead method), 380
addable() (folia.main.Term method), 366
addable() (folia.main.Text method), 394
addable() (folia.main.TextContent method), 513
addable() (folia.main.TimeSegment method), 665
addable() (folia.main.TimingLayer method), 791
addable() (folia.mainWhitespace method), 408
addable() (folia.main.Word method), 423
addable() (folia.main.AbstractAnnotationLayer class method), 59
addable() (folia.main.AbstractElement class method), 20
addable() (folia.main.AbstractSpanAnnotation class method), 46
Index 871

alternatives() (folia.main.Event method), 143
alternatives() (folia.main.Example method), 157
alternatives() (folia.main.Figure method), 171
alternatives() (folia.main.Head method), 197
alternatives() (folia.main.Linebreak method), 211
alternatives() (folia.main.List method), 225
alternatives() (folia.main.ListItem method), 239
alternatives() (folia.main.Note method), 253
alternatives() (folia.main.Paragraph method), 267
alternatives() (folia.main.Part method), 281
alternatives() (folia.main.Quote method), 295
alternatives() (folia.main.Reference method), 309
alternatives() (folia.main.Row method), 323
alternatives() (folia.main.Sentence method), 338
alternatives() (folia.main.Table method), 353
alternatives() (folia.main.TableHead method), 381
alternatives() (folia.main.Term method), 367
alternatives() (folia.main.Text method), 395
alternatives() (folia.main.Whitespace method), 409
alternatives() (folia.main.Word method), 424
ancestor() (folia.main.AbstractAnnotationLayer method), 59
ancestor() (folia.main.Annotations method), 20
ancestor() (folia.main.AbstractAnnotationLayer method), 46
ancestor() (folia.main.AbstractTextMarkup method), 53
ancestor() (folia.main.Cell method), 87
ancestor() (folia.main.Chunk method), 538
ancestor() (folia.main.ChunkingLayer method), 679
ancestor() (folia.main.CoreferenceChain method), 551
ancestor() (folia.main.CoreferenceLayer method), 692
ancestor() (folia.main.CoreferenceLink method), 805
ancestor() (folia.main.Definition method), 101
ancestor() (folia.main.DependenciesLayer method), 704
ancestor() (folia.main.Dependency method), 564
ancestor() (folia.main.DependencyDependent method), 817
ancestor() (folia.main.Division method), 115
ancestor() (folia.main.DominantAnnotation method), 441
ancestor() (folia.main.EntitiesLayer method), 717
ancestor() (folia.main.Entity method), 577
ancestor() (folia.main.Entry method), 129
ancestor() (folia.main.Event method), 143
ancestor() (folia.main.Example method), 157
ancestor() (folia.main.Figure method), 171
ancestors() (folia.main.Cell method), 87
ancestors() (folia.main.Chunk method), 538
ancestors() (folia.main.ChunkingLayer method), 679
ancestors() (folia.main.CoreferenceChain method), 551
ancestors() (folia.main.CoreferenceLayer method), 692
ancestors() (folia.main.Definition method), 101
ancestors() (folia.main.DependenciesLayer method), 704
ancestors() (folia.main.Dependency method), 564
ancestors() (folia.main.DependencyDependent method), 818
ancestors() (folia.main.Division method), 115
ancestors() (folia.main.DOMainsAnnotation method), 441
ancestors() (folia.main.EntitiesLayer method), 717
ancestors() (folia.main.Entity method), 577
ancestors() (folia.main.Entry method), 129
ancestors() (folia.main.Event method), 143
ancestors() (folia.main.Example method), 157
ancestors() (folia.main.Figure method), 171
ancestors() (folia.main.Gap method), 185
ancestors() (folia.main.Head method), 197
ancestors() (folia.main.Headspan method), 830
ancestors() (folia.main.LangAnnotation method), 465
ancestors() (folia.main.LemmaAnnotation method), 477
ancestors() (folia.main.Linebreak method), 211
ancestors() (folia.main.List method), 225
ancestors() (folia.main.ListItem method), 239
ancestors() (folia.main.Note method), 253
ancestors() (folia.main.Observation method), 590
ancestors() (folia.main.ObservationLayer method), 529
ancestors() (folia.main.Paragraph method), 267
ancestors() (folia.main.Part method), 281
ancestors() (folia.main.PosAnnotation method), 525
ancestors() (folia.main.Predicate method), 453
ancestors() (folia.main.Predicate method), 602
ancestors() (folia.main.Quote method), 295
ancestors() (folia.main.Reference method), 309
ancestors() (folia.main.Row method), 323
ancestors() (folia.main.SemanticRole method), 653
ancestors() (folia.main.SemanticRolesLayer method), 779
ancestors() (folia.main.Sentence method), 338
ancestors() (folia.main.Sentiment method), 615
ancestors() (folia.main.SentimentLayer method), 742
ancestors() (folia.main.Statement method), 628
ancestors() (folia.main.StatementLayer method), 754
ancestors() (folia.main.SubjectivityAnnotation method), 501
ancestors() (folia.main.SyntacticUnit method), 641
ancestors() (folia.main.SyntaxLayer method), 767
ancestors() (folia.main.Table method), 353
ancestors() (folia.main.TableHead method), 381
ancestors() (folia.main.Term method), 367
ancestors() (folia.main.Text method), 395
ancestors() (folia.main.TextContent method), 513
ancestors() (folia.main.TimeSegment method), 666
ancestors() (folia.main.TimingLayer method), 792
ancestors() (folia.main.Whitespace method), 409
ancestors() (folia.main.Word method), 424
annotation() (folia.main.AbstractAnnotationLayer method), 59
annotation() (folia.main.AbstractSpanAnnotation method), 47
annotation() (folia.main.AbstractStructureElement method), 33
annotation() (folia.main.Cell method), 87
annotation() (folia.main.Chunk method), 538
annotation() (folia.main.ChunkingLayer method), 679
annotation() (folia.main.CoreferenceChain method), 551
annotation() (folia.main.CoreferenceLayer method), 692
annotation() (folia.main.CoreferenceLink method), 805
annotation() (folia.main.Dependency method), 101
annotation() (folia.main.DependenciesLayer method), 704
annotation() (folia.main.DependencyDependent method), 818
annotation() (folia.main.Division method), 116
annotation() (folia.main.EntitiesLayer method), 717
annotation() (folia.main.Entry method), 577
annotation() (folia.main.Entry method), 130
annotation() (folia.main.Event method), 143
annotation() (folia.main.Example method), 157
annotation() (folia.main.Figure method), 171
annotation() (folia.main.Gap method), 185
annotation() (folia.main.Head method), 197
annotation() (folia.main.Headspan method), 830
annotation() (folia.main.LangAnnotation method), 465
annotation() (folia.main.LemmaAnnotation method), 477
annotation() (folia.main.Linebreak method), 211
annotation() (folia.main.List method), 225
annotation() (folia.main.ListItem method), 239
annotation() (folia.main.Note method), 253
annotation() (folia.main.Observation method), 590
annotation() (folia.main.ObservationLayer method), 529
annotation() (folia.main.Paragraph method), 267
annotation() (folia.main.Part method), 281
annotation() (folia.main.PosAnnotation method), 525
annotation() (folia.main.Predicate method), 453
annotation() (folia.main.Predicate method), 602
annotation() (folia.main.Quote method), 295
annotation() (folia.main.Reference method), 309
annotation() (folia.main.Row method), 323
annotation() (folia.main.SemanticRole method), 653
annotation() (folia.main.SemanticRolesLayer method), 779
annotation() (folia.main.Sentence method), 338
annotation() (folia.main.Sentiment method), 615
annotation() (folia.main.SentimentLayer method), 742
annotation() (folia.main.Statement method), 628
annotation() (folia.main.StatementLayer method), 754
annotation() (folia.main.SubjectivityAnnotation method), 501
annotation() (folia.main.SyntacticUnit method), 641
annotation() (folia.main.SyntaxLayer method), 767
annotation() (folia.main.Table method), 353
annotation() (folia.main.TableHead method), 381
annotation() (folia.main.Term method), 367
annotation() (folia.main.Text method), 395
annotation() (folia.main.TextContent method), 513
annotation() (folia.main.TimeSegment method), 666
annotation() (folia.main.TimingLayer method), 792
annotation() (folia.main.Whitespace method), 409
annotation() (folia.main.Word method), 424
annotation() (folia.main.AbstractAnnotationLayer method), 59
annotation() (folia.main.AbstractSpanAnnotation method), 47
annotation() (folia.main.AbstractStructureElement method), 33
annotation() (folia.main.Cell method), 87
annotation() (folia.main.Chunk method), 538
annotation() (folia.main.ChunkingLayer method), 679
annotation() (folia.main.CoreferenceChain method), 551
annotation() (folia.main.CoreferenceLayer method), 692
annotation() (folia.main.CoreferenceLink method), 805
annotation() (folia.main.Dependency method), 101
annotation() (folia.main.DependenciesLayer method), 704
annotation() (folia.main.DependencyDependent method), 818
annotation() (folia.main.Division method), 116
annotation() (folia.main.EntitiesLayer method), 717
annotation() (folia.main.Entry method), 577
annotation() (folia.main.Entry method), 130
annotation() (folia.main.Event method), 143
annotation() (folia.main.Example method), 157
annotation() (folia.main.Figure method), 171
annotation() (folia.main.Gap method), 185
annotation() (folia.main.Head method), 197
annotation() (folia.main.Headspan method), 830
annotation() (folia.main.List method), 225
annotation() (folia.main.ListItem method), 239
annotation() (folia.main.Note method), 253
annotation() (folia.main.Observation method), 590
<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>Method/Class Referenced</th>
</tr>
</thead>
<tbody>
<tr>
<td>folia.main.ObservationLayer method</td>
<td>730</td>
</tr>
<tr>
<td>folia.main.Paragraph method</td>
<td>267</td>
</tr>
<tr>
<td>folia.main.Part method</td>
<td>281</td>
</tr>
<tr>
<td>folia.main.Predicate method</td>
<td>602</td>
</tr>
<tr>
<td>folia.main.Quote method</td>
<td>295</td>
</tr>
<tr>
<td>folia.main.Reference method</td>
<td>310</td>
</tr>
<tr>
<td>folia.main.Row method</td>
<td>323</td>
</tr>
<tr>
<td>folia.main.SemanticRole method</td>
<td>653</td>
</tr>
<tr>
<td>folia.main.SemanticRolesLayer method</td>
<td>780</td>
</tr>
<tr>
<td>folia.main.Sentence method</td>
<td>338</td>
</tr>
<tr>
<td>folia.main.Sentiment method</td>
<td>615</td>
</tr>
<tr>
<td>folia.main.SentimentLayer method</td>
<td>742</td>
</tr>
<tr>
<td>folia.main.Statement method</td>
<td>628</td>
</tr>
<tr>
<td>folia.main.StatementLayer method</td>
<td>755</td>
</tr>
<tr>
<td>folia.main.SyntaxLayer method</td>
<td>641</td>
</tr>
<tr>
<td>folia.main.Table method</td>
<td>767</td>
</tr>
<tr>
<td>folia.main.TableHead method</td>
<td>381</td>
</tr>
<tr>
<td>folia.main.Term method</td>
<td>367</td>
</tr>
<tr>
<td>folia.main.Text method</td>
<td>395</td>
</tr>
<tr>
<td>folia.main.TimeSegment method</td>
<td>666</td>
</tr>
<tr>
<td>folia.main.TimingLayer method</td>
<td>792</td>
</tr>
<tr>
<td>folia.main.Whitespace method</td>
<td>409</td>
</tr>
<tr>
<td>folia.main.Word method</td>
<td>425</td>
</tr>
<tr>
<td>folia.main.AbstractAnnotationLayer method</td>
<td>59</td>
</tr>
<tr>
<td>folia.main.AbstractSpanAnnotationLayer method</td>
<td>47</td>
</tr>
<tr>
<td>folia.main.AbstractStructureElement method</td>
<td>33</td>
</tr>
<tr>
<td>folia.main.Cell method</td>
<td>88</td>
</tr>
<tr>
<td>folia.main.Chunk method</td>
<td>538</td>
</tr>
<tr>
<td>folia.main.ChunkingLayer method</td>
<td>679</td>
</tr>
<tr>
<td>folia.main.CoreferenceChain method</td>
<td>551</td>
</tr>
<tr>
<td>folia.main.CoreferenceLayer method</td>
<td>692</td>
</tr>
<tr>
<td>folia.main.CoreferenceLink method</td>
<td>805</td>
</tr>
<tr>
<td>folia.main.DependenciesLayer method</td>
<td>705</td>
</tr>
<tr>
<td>folia.main.Dependency method</td>
<td>564</td>
</tr>
<tr>
<td>folia.main.DependencyDependent method</td>
<td>818</td>
</tr>
<tr>
<td>folia.main.Division method</td>
<td>116</td>
</tr>
<tr>
<td>folia.main.EntitiesLayer method</td>
<td>717</td>
</tr>
<tr>
<td>folia.main.Entity method</td>
<td>577</td>
</tr>
<tr>
<td>folia.main.Event method</td>
<td>130</td>
</tr>
<tr>
<td>folia.main.Example method</td>
<td>144</td>
</tr>
<tr>
<td>folia.main.Experiment method</td>
<td>158</td>
</tr>
<tr>
<td>folia.main.Figure method</td>
<td>172</td>
</tr>
<tr>
<td>folia.main.Head method</td>
<td>198</td>
</tr>
<tr>
<td>folia.main.Headspace method</td>
<td>831</td>
</tr>
<tr>
<td>folia.main.Linebreak method</td>
<td>212</td>
</tr>
<tr>
<td>folia.main.List method</td>
<td>226</td>
</tr>
<tr>
<td>folia.main.ListItem method</td>
<td>240</td>
</tr>
<tr>
<td>folia.main.Note method</td>
<td>254</td>
</tr>
<tr>
<td>folia.main.Observation method</td>
<td>590</td>
</tr>
<tr>
<td>folia.main.Paragraph method</td>
<td>730</td>
</tr>
<tr>
<td>folia.main.Part method</td>
<td>268</td>
</tr>
<tr>
<td>folia.main.Predicate method</td>
<td>282</td>
</tr>
<tr>
<td>folia.main.Quote method</td>
<td>296</td>
</tr>
<tr>
<td>folia.main.Reference method</td>
<td>310</td>
</tr>
<tr>
<td>folia.main.Row method</td>
<td>324</td>
</tr>
<tr>
<td>folia.main.SemanticRole method</td>
<td>653</td>
</tr>
<tr>
<td>folia.main.Sentiment method</td>
<td>780</td>
</tr>
<tr>
<td>folia.main.SentimentLayer method</td>
<td>339</td>
</tr>
<tr>
<td>folia.main.Statement method</td>
<td>615</td>
</tr>
<tr>
<td>folia.main.StatementLayer method</td>
<td>742</td>
</tr>
<tr>
<td>folia.main.StatementLayer method</td>
<td>628</td>
</tr>
<tr>
<td>folia.main.StatementLayer method</td>
<td>755</td>
</tr>
<tr>
<td>folia.main.SyntaxLayer method</td>
<td>641</td>
</tr>
<tr>
<td>folia.main.Table method</td>
<td>767</td>
</tr>
<tr>
<td>folia.main.TableHead method</td>
<td>354</td>
</tr>
<tr>
<td>folia.main.TimingLayer method</td>
<td>382</td>
</tr>
<tr>
<td>folia.main.Term method</td>
<td>368</td>
</tr>
<tr>
<td>folia.main.Text method</td>
<td>396</td>
</tr>
<tr>
<td>folia.main.TimeSegment method</td>
<td>792</td>
</tr>
<tr>
<td>folia.main.Whitespace method</td>
<td>410</td>
</tr>
<tr>
<td>folia.main.Word method</td>
<td>425</td>
</tr>
</tbody>
</table>

Index 873
<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>lia.main.AbstractSpanAnnotation attribute</td>
<td>45</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.AbstractStructureElement attribute)</td>
<td>31</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.AbstractTextMarkup attribute)</td>
<td>70</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Cell attribute)</td>
<td>86</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Chunk attribute)</td>
<td>537</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.ChunkingLayer attribute)</td>
<td>678</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.CoreferenceChain attribute)</td>
<td>550</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.CoreferenceLayer attribute)</td>
<td>690</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.CoreferenceLink attribute)</td>
<td>100</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Definition attribute)</td>
<td>316</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.DependenciesLayer attribute)</td>
<td>703</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Dependency attribute)</td>
<td>562</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.DependencyDependent attribute)</td>
<td>816</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Division attribute)</td>
<td>114</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.DomainAnnotation attribute)</td>
<td>440</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.EntitiesLayer attribute)</td>
<td>716</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Entity attribute)</td>
<td>575</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Entry attribute)</td>
<td>128</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Event attribute)</td>
<td>142</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Document attribute)</td>
<td>156</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Figure attribute)</td>
<td>70</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Gap attribute)</td>
<td>183</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Head attribute)</td>
<td>196</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Headspan attribute)</td>
<td>829</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.LangAnnotation attribute)</td>
<td>464</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.LemmaAnnotation attribute)</td>
<td>476</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Linebreak attribute)</td>
<td>210</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.List attribute)</td>
<td>224</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.ListItem attribute)</td>
<td>238</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Note attribute)</td>
<td>252</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Observation attribute)</td>
<td>588</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.ObservationLayer attribute)</td>
<td>728</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (lia.main.Paragraph attribute)</td>
<td>266</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Part attribute)</td>
<td>280</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.PhraseAnnotation attribute)</td>
<td>523</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.PhraseAnnotation attribute)</td>
<td>452</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Predicate attribute)</td>
<td>601</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.BodyNote attribute)</td>
<td>294</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.BodyReference attribute)</td>
<td>308</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Row attribute)</td>
<td>322</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SemanticRole attribute)</td>
<td>652</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SemanticRolesLayer attribute)</td>
<td>778</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SenseAnnotation attribute)</td>
<td>488</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Statement attribute)</td>
<td>336</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.StatementLayer attribute)</td>
<td>614</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.StatementLayer attribute)</td>
<td>741</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.StatementLayer attribute)</td>
<td>626</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.StatementLayer attribute)</td>
<td>753</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SubjectivityAnnotation attribute)</td>
<td>500</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SyntacticUnit attribute)</td>
<td>639</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.SyntacticUnit attribute)</td>
<td>766</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Table attribute)</td>
<td>351</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.TableHead attribute)</td>
<td>379</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Term attribute)</td>
<td>365</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Text attribute)</td>
<td>393</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.TextContent attribute)</td>
<td>512</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.TimeSegment attribute)</td>
<td>665</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.TreeAnnotation attribute)</td>
<td>791</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.WhiteSpace attribute)</td>
<td>407</td>
</tr>
<tr>
<td>ANNOTATIONTYPE (folia.main.Word attribute)</td>
<td>422</td>
</tr>
<tr>
<td>annotator2processor() (folia.main.AbstractAnnotationLayer method)</td>
<td>60</td>
</tr>
<tr>
<td>annotator2processor() (folia.main.AbstractElement method)</td>
<td>20</td>
</tr>
<tr>
<td>annotator2processor() (folia.main.AbstractSpanAnnotation method)</td>
<td>47</td>
</tr>
</tbody>
</table>
annotator2processor() (folia.main.LangAnnotation method), 226
annotator2processor() (folia.main.List method), 240
annotator2processor() (folia.main.ListItem method), 539
annotator2processor() (folia.main.LemmaAnnotation method), 680
annotator2processor() (folia.main.LogicalStructure method), 692
annotator2processor() (folia.main.Note method), 805
annotator2processor() (folia.main.Observation method), 102
annotator2processor() (folia.main.ObservationLayer method), 254
annotator2processor() (folia.main.Paragraph method), 351
annotator2processor() (folia.main.Part method), 717
annotator2processor() (folia.main.Predicate method), 818
annotator2processor() (folia.main.Predicate method), 116
annotator2processor() (folia.main.Predicate method), 198
annotator2processor() (folia.main.Predicate method), 282
annotator2processor() (folia.main.PhonContent method), 324
annotator2processor() (folia.main.PosAnnotation method), 351
annotator2processor() (folia.main.Predicate method), 441
annotator2processor() (folia.main.Predicate method), 453
annotator2processor() (folia.main.Predicate method), 564
annotator2processor() (folia.main.Predicate method), 603
annotator2processor() (folia.main.Predicate method), 628
annotator2processor() (folia.main.Predicate method), 730
annotator2processor() (folia.main.Predicate method), 742
annotator2processor() (folia.main.Predicate method), 755
annotator2processor() (folia.main.Predicate method), 767
annotator2processor() (folia.main.Predicate method), 780
annotator2processor() (folia.main.Predicate method), 789
annotator2processor() (folia.main.Predicate method), 805
annotator2processor() (folia.main.Predicate method), 854
annotator2processor() (folia.main.Predicate method), 909
annotator2processor() (folia.main.Predicate method), 978
annotator2processor() (folia.main.Predicate method), 1047
annotator2processor() (folia.main.Predicate method), 1115
annotator2processor() (folia.main.Predicate method), 1183
annotator2processor() (folia.main.Predicate method), 1252
annotator2processor() (folia.main.Predicate method), 1320
annotator2processor() (folia.main.Predicate method), 1388
annotator2processor() (folia.main.Predicate method), 1456
annotator2processor() (folia.main.Predicate method), 1524
annotator2processor() (folia.main.Predicate method), 1592
annotator2processor() (folia.main.Predicate method), 1660
annotator2processor() (folia.main.Predicate method), 1728
annotator2processor() (folia.main.Predicate method), 1796
annotator2processor() (folia.main.Predicate method), 1864
annotator2processor() (folia.main.Predicate method), 1932
annotator2processor() (folia.main.Predicate method), 2000
annotator2processor() (folia.main.Predicate method), 2068
annotator2processor() (folia.main.Predicate method), 2136
annotator2processor() (folia.main.Predicate method), 2204
annotator2processor() (folia.main.Predicate method), 2272
annotator2processor() (folia.main.Predicate method), 2340
annotator2processor() (folia.main.Predicate method), 2408
annotator2processor() (folia.main.Predicate method), 2476
annotator2processor() (folia.main.Predicate method), 2544
annotator2processor() (folia.main.Predicate method), 2612
annotator2processor() (folia.main.Predicate method), 2680
annotator2processor() (folia.main.Predicate method), 2748
annotator2processor() (folia.main.Predicate method), 2816
annotator2processor() (folia.main.Predicate method), 2884
annotator2processor() (folia.main.Predicate method), 2952
annotator2processor() (folia.main.Predicate method), 3020
annotator2processor() (folia.main.Predicate method), 3088
annotator2processor() (folia.main.Predicate method), 3156
annotator2processor() (folia.main.Predicate method), 3224
annotator2processor() (folia.main.Predicate method), 3292
annotator2processor() (folia.main.Predicate method), 3360
annotator2processor() (folia.main.Predicate method), 3428
annotator2processor() (folia.main.Predicate method), 3496
annotator2processor() (folia.main.Predicate method), 3564
annotator2processor() (folia.main.Predicate method), 3632
annotator2processor() (folia.main.Predicate method), 3700
annotator2processor() (folia.main.Predicate method), 3768
annotator2processor() (folia.main.Predicate method), 3836
annotator2processor() (folia.main.Predicate method), 3904
annotator2processor() (folia.main.Predicate method), 3972
annotator2processor() (folia.main.Predicate method), 4040
annotator2processor() (folia.main.Predicate method), 4108
annotator2processor() (folia.main.Predicate method), 4176
annotator2processor() (folia.main.Predicate method), 4244
annotator2processor() (folia.main.Predicate method), 4312
annotator2processor() (folia.main.Predicate method), 4380
annotator2processor() (folia.main.Predicate method), 4448
annotator2processor() (folia.main.Predicate method), 4516
annotator2processor() (folia.main.Predicate method), 4584
annotator2processor() (folia.main.Predicate method), 4652
annotator2process
method), 368
append() (folia.main.Text method), 396
annotator2processor() (folia.main.TextContent method), 514
annotator2processor() (folia.main.TimeSegment method), 666
annotator2processor() (folia.main.TimingLayer method), 792
annotator2processor() (folia.main.Whitespace method), 410
annotator2processor() (folia.main.Word method), 425
append() (folia.main.AbstractAnnotationLayer method), 60
append() (folia.main.AbstractElement method), 20
append() (folia.main.AbstractSpanAnnotation method), 47
append() (folia.main.AbstractStructureElement method), 34
append() (folia.main.AbstractTextMarkup method), 72
append() (folia.main.Cell method), 88
append() (folia.main.Chunk method), 539
append() (folia.main.ChunkingLayer method), 680
append() (folia.main.CoreferenceChain method), 551
append() (folia.main.CoreferenceLayer method), 692
append() (folia.main.CoreferenceLink method), 805
append() (folia.main.Definition method), 102
append() (folia.main.DependenciesLayer method), 705
append() (folia.main.Dependency method), 564
append() (folia.main.DependencyDependent method), 818
append() (folia.main.Division method), 116
append() (folia.main.Document method), 8
append() (folia.main.DomainAnnotation method), 441
append() (folia.main.EntitiesLayer method), 717
append() (folia.main.Entity method), 577
append() (folia.main.Entry method), 130
append() (folia.main.Event method), 144
append() (folia.main.Example method), 158
append() (folia.main.Figure method), 172
append() (folia.main.Gap method), 185
append() (folia.main.Head method), 198
append() (folia.main.Headspan method), 831
append() (folia.main.LangAnnotation method), 465
append() (folia.main.LemmaAnnotation method), 477
append() (folia.main.Linebreak method), 212
append() (folia.main.List method), 226
append() (folia.main.ListItem method), 240
append() (folia.main.Note method), 254
append() (folia.main.Observation method), 590
append() (folia.main.ObservationLayer method), 730
append() (folia.main.Paragraph method), 268
append() (folia.main.Part method), 282
append() (folia.main.PronContent method), 525
append() (folia.main.PosAnnotation method), 453
append() (folia.main.Predicate method), 603
append() (folia.main.PredQuote method), 296
append() (folia.main.Reference method), 310
append() (folia.main.Row method), 324
append() (folia.main.SemanticRole method), 654
append() (folia.main.SemanticRolesLayer method), 780
append() (folia.main.SenseAnnotation method), 489
append() (folia.mainSentence method), 339
append() (folia.mainSentence method), 615
append() (folia.mainSentenceLayer method), 742
append() (folia.main.Statement method), 628
append() (folia.main.StatementLayer method), 755
append() (folia.main.SubjectivityAnnotation method), 501
append() (folia.main.SyntacticUnit method), 641
append() (folia.main.SyntaxLayer method), 767
append() (folia.main.Table method), 354
append() (folia.main.TableHead method), 382
append() (folia.main.Term method), 368
append() (folia.main.Text method), 396
append() (folia.main.TextContent method), 514
append() (folia.main.TimeSegment method), 666
append() (folia.main.TimingLayer method), 792
append() (folia.main.WhiteSpace method), 410
append() (folia.main.Word method), 426
attachexternal() (folia.main.Document method), 8
AUTH (folia.main.AbstractAnnotationLayer attribute), 58
AUTH (folia.main.AbstractElement attribute), 19
AUTH (folia.main.AbstractSpanAnnotation attribute), 45
AUTH (folia.main.AbstractStructureElement attribute), 31
AUTH (folia.main.AbstractTextMarkup attribute), 70
AUTH (folia.main.Cell attribute), 86
AUTH (folia.main.Chunk attribute), 537
AUTH (folia.main.ChunkingLayer attribute), 678
AUTH (folia.main.CoreferenceChain attribute), 550
AUTH (folia.main.CoreferenceLayer attribute), 691
AUTH (folia.main.CoreferenceLink attribute), 803
AUTH (folia.main.Definition attribute), 100
AUTH (folia.main.DependenciesLayer attribute), 703
AUTH (folia.main.Dependency attribute), 563
AUTH (folia.main.DependencyDependent attribute), 816
AUTH (folia.main.Division attribute), 114
AUTH (folia.main.DomainAnnotation attribute), 440
AUTH (folia.main.EntitiesLayer attribute), 716
AUTH (folia.main.Entity attribute), 575
AUTH (folia.main.Entry attribute), 128
AUTH (folia.main.Event attribute), 142
AUTH (folia.main.Example attribute), 156
context (folia.main.Observation method), 590
context (folia.main.ObservationLayer method), 730
context (folia.main.Paragraph method), 269
context (folia.main.Part method), 283
context (folia.main.PhonContent method), 525
context (folia.main.PosAnnotation method), 453
context (folia.main.Predicate method), 603
context (folia.main.Prompt method), 297
context (folia.main.Reference method), 311
context (folia.main.Row method), 325
context (folia.main.SemanticRole method), 654
context (folia.main.SentimentLayer method), 780
context (folia.main.SemanticRolesLayer method), 489
context (folia.main.Sentiment method), 339
context (folia.main.Statement method), 616
context (folia.main.StatementLayer method), 742
context (folia.main.StatementLayer method), 628
context (folia.main.StatementLayer method), 755
context (folia.main.StatementLayer method), 501
context (folia.main.StatementLayer method), 641
context (folia.main.SyntaxLayer method), 767
context (folia.main.Table method), 354
context (folia.main.TableHead method), 382
context (folia.main.Term method), 368
context (folia.main.Text method), 375
context (folia.main.TextContent method), 368
context (folia.main.TextContent method), 396
context (folia.main.TextContent method), 514
context (folia.main.TimeSegment method), 667
context (folia.main.TimeSegment method), 792
context (folia.main.WhiteSpace method), 410
context (folia.main.Word method), 426
context (folia.main.AbstractAnnotationLayer method), 60
context (folia.main.AbstractAnnotationLayer method), 20
context (folia.main.AbstractAnnotationLayer method), 47
context (folia.main.AbstractStructureElement method), 34
context (folia.main.AbstractTextMarkup method), 72
context (folia.main.Cell method), 89
context (folia.main.Chunk method), 539
context (folia.main.ChunkingLayer method), 680
context (folia.main.CoreferenceChain method), 552
context (folia.main.CoreferenceLayer method), 692
context (folia.main.CoreferenceLink method), 805
context (folia.main.Definition method), 103
context (folia.main.DependenciesLayer method), 705
context (folia.main.Dependency method), 564
context (folia.main.DependencyDependent method), 818
copy (folia.main.Division method), 117
copy (folia.main_DOMAINANNOTATION method), 441
copy (folia.main.EntitiesLayer method), 718

copy (folia.main.Entity method), 577
copy (folia.main.Entry method), 131
copy (folia.main.Event method), 145
copy (folia.main.Example method), 159
copy (folia.main.Figure method), 173
copy (folia.main.Gap method), 185
copy (folia.main.Headspace method), 199
copy (folia.main.Head method), 831
copy (folia.main.LangAnnotation method), 465
copy (folia.main.LemmaAnnotation method), 477
copy (folia.main.Linebreak method), 213
copy (folia.main.List method), 227
copy (folia.main.ListItem method), 241
copy (folia.main.Note method), 255
copy (folia.main.Observation method), 590
copy (folia.main.ObservationLayer method), 730
copy (folia.main.Paragraph method), 269
copy (folia.main.Part method), 283
copy (folia.main.Predicate method), 525
copy (folia.main.PosAnnotation method), 453
copy (folia.main.Predicate method), 603
copy (folia.main.Predicate method), 297
copy (folia.main.Predicate method), 311
copy (folia.main.Predicate method), 325
copy (folia.main.Predicate method), 654
copy (folia.main.Predicate method), 780
copy (folia.main.Predicate method), 742
copy (folia.main.Predicate method), 628
copy (folia.main.Predicate method), 755
copy (folia.main.Predicate method), 501
copy (folia.main.Prompt method), 641
copy (folia.main.Prompt method), 768
copy (folia.main.Prompt method), 354
copy (folia.main.TableHead method), 382
copy (folia.main.Text method), 368
copy (folia.main.TextContent method), 368
copy (folia.main.TextContent method), 396
copy (folia.main.TimeSegment method), 514
copy (folia.main.TimeSegment method), 667
copy (folia.main.TimeSegment method), 792
copy (folia.main.WhiteSpace method), 410
copy (folia.main.Word method), 426
copy (folia.main.AbstractAnnotationLayer method), 60
copy (folia.main.AbstractAnnotationLayer method), 20
copy (folia.main.AbstractAnnotationLayer method), 47
copy (folia.main.AbstractStructureElement method), 34
copy (folia.main.AbstractTextMarkup method), 72
copy (folia.main.Cell method), 89
copy (folia.main.Chunk method), 539
copy (folia.main.ChunkingLayer method), 680
copy (folia.main.CoreferenceChain method), 552
copy (folia.main.CoreferenceLayer method), 692
copy (folia.main.CoreferenceLink method), 805
copy (folia.main.Definition method), 103
copy (folia.main.DependenciesLayer method), 705
copy (folia.main.Dependency method), 564
copy (folia.main.DependencyDependent method), 818

copychildren() (folia.main.AbstractAnnotationLayer method), 60

copychildren() (folia.main.AbstractElement method), 21

copychildren() (folia.main.AbstractSpanAnnotation method), 47
copychildren() (folia.main.AbstractStructureElement method), 34
copychildren() (folia.main.AbstractTextMarkup method), 72
copychildren() (folia.main.Cell method), 89
copychildren() (folia.main.Chunk method), 539
copychildren() (folia.main.ChunkingLayer method), 680
copychildren() (folia.main.CoreferenceChain method), 552
copychildren() (folia.main.CoreferenceLayer method), 693
copychildren() (folia.main.CoreferenceLink method), 806
copychildren() (folia.main.Definition method), 103
copychildren() (folia.main.DependenciesLayer method), 705
copychildren() (folia.main.Dependency method), 565
copychildren() (folia.main.DependencyDependent method), 818
copychildren() (folia.main.Division method), 117
copychildren() (folia.main.DomainAnnotation method), 441
copychildren() (folia.main.EntitiesLayer method), 718
copychildren() (folia.main.Entity method), 578
copychildren() (folia.main.Entry method), 131
copychildren() (folia.main.Event method), 145
copychildren() (folia.main.Example method), 159
copychildren() (folia.main.Figure method), 173
copychildren() (folia.main.Figure method), 185
copychildren() (folia.main.Head method), 199
copychildren() (folia.main.HeadSpan method), 831
copychildren() (folia.main.LangAnnotation method), 465
copychildren() (folia.main.LemmaAnnotation method), 477
copychildren() (folia.main.Linebreak method), 213
copychildren() (folia.main.List method), 227
copychildren() (folia.main.ListItem method), 241
copychildren() (folia.main.Note method), 255
copychildren() (folia.main.Observation method), 590
copychildren() (folia.main.ObservationLayer method), 730
copychildren() (folia.main.Paragraph method), 269
copychildren() (folia.main.Part method), 283
copychildren() (folia.main.Phrase method), 526
copychildren() (folia.main.Phrase method), 453
copychildren() (folia.main.Predicate method), 603
copychildren() (folia.main.Quote method), 297
copychildren() (folia.main.Reference method), 311
copychildren() (folia.main.Row method), 325
copychildren() (folia.main.SemanticRole method), 654
copychildren() (folia.main.SemanticRolesLayer method), 780
copychildren() (folia.main.Sentence method), 340
copychildren() (folia.main.Sentiment method), 616
copychildren() (folia.main.SentimentLayer method), 743
copychildren() (folia.main.Statement method), 629
copychildren() (folia.main.StatementLayer method), 755
copychildren() (folia.main.SubjectivityAnnotation method), 501
copychildren() (folia.main.SyntacticUnit method), 641
copychildren() (folia.main.SyntaxLayer method), 768
copychildren() (folia.main.Table method), 354
copychildren() (folia.main.StatementLayer method), 382
copychildren() (folia.main.Term method), 368
copychildren() (folia.main.Text method), 396
copychildren() (folia.main.TextContent method), 514
copychildren() (folia.main.TimeSegment method), 667
copychildren() (folia.main.TimingLayer method), 793
copychildren() (folia.main.Whitespace method), 410
copychildren() (folia.main.Word method), 426
CoreferenceChain (class in folia.main), 547
CoreferenceLayer (class in folia.main), 688
CoreferenceLink (class in folia.main), 801
CoreferenceChain (class in folia.main), 547
CoreferenceLayer (class in folia.main), 688
CoreferenceLink (class in folia.main), 801
correct() (folia.main.AbstractAnnotationLayer method), 60
correct() (folia.main.AbstractAnnotationLayer method), 47
correct() (folia.main.AbstractStructureElement method), 34
correct() (folia.main.Cell method), 89
correct() (folia.main.Chunk method), 539
correct() (folia.main.ChunkingLayer method), 680
correct() (folia.main.CoreferenceChain method), 552
correct() (folia.main.CoreferenceLayer method), 693
correct() (folia.main.CoreferenceLink method), 806
correct() (folia.main.Definition method), 103
correct() (folia.main.DependenciesLayer method),
count() (folia.main.TextContent method), 514
count() (folia.main.TimeSegment method), 667
count() (folia.main.TimingLayer method), 793
count() (folia.main.Whitespace method), 411
count() (folia.main.Word method), 426
create() (folia.main.Document method), 8
date() (folia.main.Document method), 8
declare() (folia.main.Document method), 9
declared() (folia.main.Document method), 10
deepvalidation() (folia.main.AbstractAnnotationLayer method), 60
deepvalidation() (folia.main.AbstractElement method), 21
deepvalidation() (folia.main.AbstractSpanAnnotation method), 47
deepvalidation() (folia.main.AbstractStructureElement method), 34
deepvalidation() (folia.main.AbstractTextMarkup method), 72
deepvalidation() (folia.main.Cell method), 89
deepvalidation() (folia.main.Chunk method), 539
deepvalidation() (folia.main.ChunkingLayer method), 680
deepvalidation() (folia.main.CoreferenceChain method), 552
deepvalidation() (folia.main.CoreferenceLayer method), 693
deepvalidation() (folia.main.CoreferenceLink method), 806
deepvalidation() (folia.main.Definition method), 103
deepvalidation() (folia.main.DependenciesLayer method), 705
deepvalidation() (folia.main.Dependency method), 565
deepvalidation() (folia.main.DependencyDependent method), 819
deepvalidation() (folia.main.Division method), 117
deepvalidation() (folia.main.DomainAnnotation method), 442
deepvalidation() (folia.main.EntitiesLayer method), 718
deepvalidation() (folia.main.Entity method), 578
deepvalidation() (folia.main.Entry method), 131
deepvalidation() (folia.main.Event method), 145
deepvalidation() (folia.main.Example method), 159
deepvalidation() (folia.main.Figure method), 173
deepvalidation() (folia.main.Flap method), 185
deepvalidation() (folia.main.Head method), 199
deepvalidation() (folia.main.Headspace method), 831
deepvalidation() (folia.main.LangAnnotation method), 466
deepvalidation() (folia.main.LemmaAnnotation method), 478
deepvalidation() (folia.main.Linebreak method), 213
deepvalidation() (folia.main.List method), 227
deepvalidation() (folia.main.ListItem method), 241
deepvalidation() (folia.main.Note method), 255
deepvalidation() (folia.main.Observation method), 591
deepvalidation() (folia.main.ObservationLayer method), 730
deepvalidation() (folia.main.Paragraph method), 269
deepvalidation() (folia.main.Part method), 283
deepvalidation() (folia.main.PhonContent method), 526
deepvalidation() (folia.main.PosAnnotation method), 454
deepvalidation() (folia.main.Predicate method), 603
deepvalidation() (folia.main.Quote method), 297
deepvalidation() (folia.main.Reference method), 311
deepvalidation() (folia.main.Row method), 325
deepvalidation() (folia.main.SemanticRole method), 654
deepvalidation() (folia.main.SemanticRolesLayer method), 780
deepvalidation() (folia.main.SenseAnnotation method), 490
deepvalidation() (folia.main.Sentence method), 340
deepvalidation() (folia.main.Sentiment method), 616
deepvalidation() (folia.main.SentimentLayer method), 743
deepvalidation() (folia.main.Statement method), 629
deepvalidation() (folia.main.StatementLayer method), 755
deepvalidation() (folia.main.SubjectivityAnnotation method), 502
deepvalidation() (folia.main.SyntacticUnit method), 642
deepvalidation() (folia.main.SyntaxLayer method), 883
method), 768
deepvalidation() (folia.main.Table method), 355
deepvalidation() (folia.main.TableHead method), 383
deepvalidation() (folia.main.Term method), 369
deepvalidation() (folia.main.Text method), 397
deepvalidation() (folia.main.TextContent method), 514
deepvalidation() (folia.main.TimeSegment method), 667
deepvalidation() (folia.main.TimingLayer method), 793
deepvalidation() (folia.main.Whitespace method), 411
deepvalidation() (folia.main.Word method), 426
defaultannotator() (folia.main.Document method), 10
defaultannotator() (folia.main.Document method), 11
defaultdatetime() (folia.main.Document method), 11
defaultset() (folia.main.Document method), 11
Definition (class in folia.main), 97
deleteword () (folia.mainSentence method), 340
DependenciesLayer (class in folia.main), 700
Dependency (class in folia.main), 560
DependencyDependent (class in folia.main), 813
dependent () (folia.main.Dependency method), 565
depthfirstsearch() (folia.main.AbstractAnnotationLayer method), 60
depthfirstsearch() (folia.main.AbstractElement method), 21
depthfirstsearch() (folia.main.AbstractSpanAnnotation method), 47
depthfirstsearch() (folia.main.AbstractStructureElement method), 34
depthfirstsearch() (folia.main.AbstractTextMarkup method), 72
depthfirstsearch() (folia.main.Cell method), 89
depthfirstsearch() (folia.main.Chunk method), 539
depthfirstsearch() (folia.main.ChunkingLayer method), 680
depthfirstsearch() (folia.main.CoreferenceChain method), 552
depthfirstsearch() (folia.main.CoreferenceLayer method), 693
depthfirstsearch() (folia.main.CoreferenceLink method), 806
depthfirstsearch() (folia.main.Definition method), 103
depthfirstsearch() (folia.main.DependenciesLayer method), 705
depthfirstsearch() (folia.main.Dependency method), 565
depthfirstsearch() (folia.main.DependencyDependent method), 819
depthfirstsearch() (folia.main.Division method), 117
depthfirstsearch() (folia.main.DominAntLayer method), 442
depthfirstsearch() (folia.main.EntitiesLayer method), 718
depthfirstsearch() (folia.main.Entity method), 578
depthfirstsearch() (folia.main.Event method), 145
depthfirstsearch() (folia.main.Example method), 159
depthfirstsearch() (folia.main.Figure method), 173
depthfirstsearch() (folia.main.Gap method), 186
depthfirstsearch() (folia.main.Head method), 199
depthfirstsearch() (folia.main.HeadSpan method), 831
depthfirstsearch() (folia.main.LangAnnotation method), 466
depthfirstsearch() (folia.main.LemmaAnnotation method), 478
depthfirstsearch() (folia.main.Linebreak method), 213
depthfirstsearch() (folia.main.List method), 241
depthfirstsearch() (folia.main.Note method), 255
depthfirstsearch() (folia.main.Observation method), 591
depthfirstsearch() (folia.main.ObservationLayer method), 730
depthfirstsearch() (folia.mainParagraph method), 269
depthfirstsearch() (folia.main.Part method), 526
depthfirstsearch() (folia.main.PhraseContent method), 283
depthfirstsearch() (folia.main.PosAnnotation method), 544
depthfirstsearch() (folia.main.Predicate method), 603
depthfirstsearch() (folia.main.Quote method), 297
Index 885

depthfirstsearch() (folia.main.Reference method), 311
depthfirstsearch() (folia.main.Row method), 325
depthfirstsearch() (folia.main.SemanticRole method), 654
depthfirstsearch() (folia.main.SemanticRolesLayer method), 780
depthfirstsearch() (folia.main.SenseAnnotation method), 490
depthfirstsearch() (folia.main.Statement method), 340
depthfirstsearch() (folia.main.Sentiment method), 616
depthfirstsearch() (folia.main.SentimentLayer method), 743
depthfirstsearch() (folia.main.StatementLayer method), 629
depthfirstsearch() (folia.main.StatementLayer method), 755
depthfirstsearch() (folia.main.SubjectivityAnnotation method), 502
depthfirstsearch() (folia.main.SyntacticUnit method), 642
depthfirstsearch() (folia.main.SyntaxLayer method), 768
depthfirstsearch() (folia.main.Table method), 355
depthfirstsearch() (folia.main.TableHead method), 383
depthfirstsearch() (folia.main.Term method), 369
depthfirstsearch() (folia.main.Text method), 397
depthfirstsearch() (folia.main.TextContent method), 514
depthfirstsearch() (folia.main.TimeSegment method), 667
depthfirstsearch() (folia.main.TimingLayer method), 793
depthfirstsearch() (folia.main.Whitespace method), 411
depthfirstsearch() (folia.main.Word method), 426
description() (folia.main.AbstractAnnotationLayer method), 60
description() (folia.main.AbstractElement method), 21
description() (folia.main.AbstractSpanAnnotation method), 48
description() (folia.main.AbstractStructureElement method), 34
description() (folia.main.AbstractTextMarkup method), 73
description() (folia.main.Cell method), 89
description() (folia.main.Chunk method), 539
description() (folia.main.ChunkingLayer method), 680
description() (folia.main.CoreferenceChain method), 552
description() (folia.main.CoreferenceLayer method), 693
description() (folia.main.CoreferenceLink method), 806
description() (folia.main.Definition method), 103
description() (folia.main.DependenciesLayer method), 705
description() (folia.main.Dependency method), 565
description() (folia.main.DependencyDependent method), 819
description() (folia.main.Division method), 117
description() (folia.main.DomainAnnotation method), 442
description() (folia.main.EntitiesLayer method), 718
description() (folia.main.Entity method), 578
description() (folia.main.Entry method), 131
description() (folia.main.Event method), 145
description() (folia.main.Example method), 159
description() (folia.main.Figure method), 173
description() (folia.main.Gap method), 186
description() (folia.main.Head method), 199
description() (folia.main.Headspan method), 831
description() (folia.main.LangAnnotation method), 466
description() (folia.main.LemmaAnnotation method), 478
description() (folia.main.Linebreak method), 213
description() (folia.main.List method), 227
description() (folia.main.ListItem method), 241
description() (folia.main.Note method), 255
description() (folia.main.Observation method), 591
description() (folia.main.ObservationLayer method), 731
description() (folia.main.Paragraph method), 269
description() (folia.main.Part method), 283
description() (folia.main.Phrase method), 526
description() (folia.main.PosAnnotation method), 454
description() (folia.main.Predicate method), 603
description() (folia.main.Phrase method), 297
description() (folia.main.Reference method), 311
description() (folia.main.Row method), 325
description() (folia.main.SemanticRole method), 654
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>description() (folia.main.SemanticRolesLayer method)</td>
<td>781</td>
</tr>
<tr>
<td>description() (folia.mainSenseAnnotation method)</td>
<td>490</td>
</tr>
<tr>
<td>description() (folia.mainSentence method)</td>
<td>340</td>
</tr>
<tr>
<td>description() (folia.mainSentiment method)</td>
<td>616</td>
</tr>
<tr>
<td>description() (folia.mainSentimentLayer method)</td>
<td>743</td>
</tr>
<tr>
<td>description() (folia.mainStatement method)</td>
<td>629</td>
</tr>
<tr>
<td>description() (folia.mainStatementLayer method)</td>
<td>756</td>
</tr>
<tr>
<td>description() (folia.mainSubjectivityAnnotation method)</td>
<td>502</td>
</tr>
<tr>
<td>description() (folia.mainSyntaxUnit method)</td>
<td>642</td>
</tr>
<tr>
<td>description() (folia.mainSyntaxLayer method)</td>
<td>768</td>
</tr>
<tr>
<td>description() (folia.mainTable method)</td>
<td>355</td>
</tr>
<tr>
<td>description() (folia.mainTableHead method)</td>
<td>383</td>
</tr>
<tr>
<td>description() (folia.mainTerm method)</td>
<td>369</td>
</tr>
<tr>
<td>description() (folia.mainText method)</td>
<td>397</td>
</tr>
<tr>
<td>description() (folia.mainTextContent method)</td>
<td>514</td>
</tr>
<tr>
<td>description() (folia.mainTimeSegment method)</td>
<td>667</td>
</tr>
<tr>
<td>description() (folia.mainTimingLayer method)</td>
<td>793</td>
</tr>
<tr>
<td>description() (folia.mainWhitespace method)</td>
<td>411</td>
</tr>
<tr>
<td>Division (class in folia.main)</td>
<td>111</td>
</tr>
<tr>
<td>division() (folia.mainSentence method)</td>
<td>340</td>
</tr>
<tr>
<td>division() (folia.mainWord method)</td>
<td>426</td>
</tr>
<tr>
<td>Document (class in folia.main)</td>
<td>3</td>
</tr>
<tr>
<td>domain() (folia.mainWord method)</td>
<td>426</td>
</tr>
<tr>
<td>DomainAnnotation (class in folia.main)</td>
<td>437</td>
</tr>
<tr>
<td>done() (folia.mainDocument method)</td>
<td>11</td>
</tr>
<tr>
<td>feat() (folia.mainChunk method)</td>
<td>539</td>
</tr>
<tr>
<td>feat() (folia.mainChunkingLayer method)</td>
<td>680</td>
</tr>
<tr>
<td>feat() (folia.mainCoreferenceChain method)</td>
<td>552</td>
</tr>
<tr>
<td>feat() (folia.mainCoreferenceLayer method)</td>
<td>693</td>
</tr>
<tr>
<td>feat() (folia.mainCoreferenceLink method)</td>
<td>806</td>
</tr>
<tr>
<td>feat() (folia.mainDefinition method)</td>
<td>103</td>
</tr>
<tr>
<td>feat() (folia.mainDependenciesLayer method)</td>
<td>706</td>
</tr>
<tr>
<td>feat() (folia.mainDependency method)</td>
<td>565</td>
</tr>
<tr>
<td>feat() (folia.mainDependencyDependent method)</td>
<td>819</td>
</tr>
<tr>
<td>feat() (folia.mainDivision method)</td>
<td>117</td>
</tr>
<tr>
<td>feat() (folia.mainDomainAnnotation method)</td>
<td>442</td>
</tr>
<tr>
<td>feat() (folia.mainEntitiesLayer method)</td>
<td>718</td>
</tr>
<tr>
<td>feat() (folia.mainEntity method)</td>
<td>578</td>
</tr>
<tr>
<td>feat() (folia.mainEntry method)</td>
<td>131</td>
</tr>
<tr>
<td>feat() (folia.mainEvent method)</td>
<td>145</td>
</tr>
<tr>
<td>feat() (folia.mainExample method)</td>
<td>159</td>
</tr>
<tr>
<td>feat() (folia.mainFigure method)</td>
<td>173</td>
</tr>
<tr>
<td>feat() (folia.mainGap method)</td>
<td>186</td>
</tr>
<tr>
<td>feat() (folia.mainHead method)</td>
<td>199</td>
</tr>
<tr>
<td>feat() (folia.mainHeadspace method)</td>
<td>832</td>
</tr>
<tr>
<td>feat() (folia.mainLangAnnotation method)</td>
<td>466</td>
</tr>
<tr>
<td>feat() (folia.mainLemmaAnnotation method)</td>
<td>478</td>
</tr>
<tr>
<td>feat() (folia.mainLinebreak method)</td>
<td>213</td>
</tr>
<tr>
<td>feat() (folia.mainList method)</td>
<td>227</td>
</tr>
<tr>
<td>feat() (folia.mainListItem method)</td>
<td>241</td>
</tr>
<tr>
<td>feat() (folia.mainNote method)</td>
<td>255</td>
</tr>
<tr>
<td>feat() (folia.mainObservation method)</td>
<td>591</td>
</tr>
<tr>
<td>feat() (folia.mainObservationLayer method)</td>
<td>731</td>
</tr>
<tr>
<td>feat() (folia.mainParagraph method)</td>
<td>269</td>
</tr>
<tr>
<td>feat() (folia.mainPart method)</td>
<td>283</td>
</tr>
<tr>
<td>feat() (folia.mainPhonContent method)</td>
<td>526</td>
</tr>
<tr>
<td>feat() (folia.mainPosAnnotation method)</td>
<td>454</td>
</tr>
<tr>
<td>feat() (folia.mainPredicate method)</td>
<td>603</td>
</tr>
<tr>
<td>feat() (folia.mainQuote method)</td>
<td>297</td>
</tr>
<tr>
<td>feat() (folia.mainReference method)</td>
<td>311</td>
</tr>
<tr>
<td>feat() (folia.mainRow method)</td>
<td>325</td>
</tr>
<tr>
<td>feat() (folia.mainSemanticRole method)</td>
<td>654</td>
</tr>
<tr>
<td>feat() (folia.mainSemanticRolesLayer method)</td>
<td>781</td>
</tr>
<tr>
<td>feat() (folia.mainSenseAnnotation method)</td>
<td>490</td>
</tr>
<tr>
<td>feat() (folia.mainSentence method)</td>
<td>340</td>
</tr>
<tr>
<td>feat() (folia.mainSentiment method)</td>
<td>616</td>
</tr>
<tr>
<td>feat() (folia.mainSentimentLayer method)</td>
<td>743</td>
</tr>
<tr>
<td>feat() (folia.mainStatement method)</td>
<td>629</td>
</tr>
<tr>
<td>feat() (folia.mainStatementLayer method)</td>
<td>756</td>
</tr>
<tr>
<td>feat() (folia.mainSubjectivityAnnotation method)</td>
<td>502</td>
</tr>
<tr>
<td>feat() (folia.mainSyntaxUnit method)</td>
<td>642</td>
</tr>
<tr>
<td>feat() (folia.mainSyntaxLayer method)</td>
<td>768</td>
</tr>
<tr>
<td>feat() (folia.mainTable method)</td>
<td>355</td>
</tr>
<tr>
<td>feat() (folia.mainTableHead method)</td>
<td>383</td>
</tr>
<tr>
<td>feat() (folia.mainTerm method)</td>
<td>369</td>
</tr>
<tr>
<td>feat() (folia.mainText method)</td>
<td>397</td>
</tr>
<tr>
<td>feat() (folia.mainTextContent method)</td>
<td>514</td>
</tr>
</tbody>
</table>
feat()  (folia.main.TimeSegment method), 667
feat()  (folia.main.TimingLayer method), 793
feat()  (folia.main.Whitespace method), 411
feat()  (folia.main.Word method), 426
Figure (class in folia.main), 167
findcorrectionhandling()  (folia.main.AbstractAnnotationLayer method), 61
findcorrectionhandling()  (folia.main.AbstractElement method), 21
findcorrectionhandling()  (folia.main.AbstractSpanAnnotation method), 48
findcorrectionhandling()  (folia.main.AbstractStructureElement method), 35
findcorrectionhandling()  (folia.main.Cell method), 89
findcorrectionhandling()  (folia.main.Chunk method), 540
findcorrectionhandling()  (folia.main.ChunkingLayer method), 681
findcorrectionhandling()  (folia.main.CoreferenceChain method), 552
findcorrectionhandling()  (folia.main.CoreferenceLayer method), 693
findcorrectionhandling()  (folia.main.CoreferenceLink method), 806
findcorrectionhandling()  (folia.main.Definition method), 103
findcorrectionhandling()  (folia.main.DependenciesLayer method), 706
findcorrectionhandling()  (folia.main.Dependency method), 565
findcorrectionhandling()  (folia.main.DependencyDependent method), 819
findcorrectionhandling()  (folia.main.Division method), 117
findcorrectionhandling()  (folia.main.DomainAnnotation method), 442
findcorrectionhandling()  (folia.main.EntitiesLayer method), 718
findcorrectionhandling()  (folia.main.Entity method), 578
findcorrectionhandling()  (folia.main.Entry method), 131
findcorrectionhandling()  (folia.main.Event method), 145
findcorrectionhandling()  (folia.main.Example method), 159
findcorrectionhandling()  (folia.main.Figure method), 173
findcorrectionhandling()  (folia.main.Gap method), 186
findcorrectionhandling()  (folia.main.Head method), 199
findcorrectionhandling()  (folia.main.Headspace method), 832
findcorrectionhandling()  (folia.main.LangAnnotation method), 466
findcorrectionhandling()  (folia.main.LemmaAnnotation method), 478
findcorrectionhandling()  (folia.main.Linebreak method), 213
findcorrectionhandling()  (folia.main.List method), 227
findcorrectionhandling()  (folia.main.ListItem method), 241
findcorrectionhandling()  (folia.main.Note method), 255
findcorrectionhandling()  (folia.main.Observation method), 591
findcorrectionhandling()  (folia.main.ObservationLayer method), 731
findcorrectionhandling()  (folia.main.Paragraph method), 269
findcorrectionhandling()  (folia.main.Part method), 283
findcorrectionhandling()  (folia.main.PhraseAnnotation method), 526
findcorrectionhandling()  (folia.main.Predicate method), 544
findcorrectionhandling()  (folia.main.Predicate method), 604
findcorrectionhandling()  (folia.main.PhraseAnnotation method), 297
findcorrectionhandling()  (folia.main.Reference method), 311
findcorrectionhandling()  (folia.main.Row method), 325
findcorrectionhandling()  (folia.main.SemanticRole method), 655
findcorrectionhandling()  (folia.main.SemanticRolesLayer method), 781
findcorrectionhandling()  (folia.main.SenseAnnotation method), 490
findcorrectionhandling()  (folia.main.Statement method), 341
findcorrectionhandling()  (folia.main.Sentence method), 341
findcorrectionhandling()  (folia.main.Sentence method), 616
findcorrectionhandling()  (folia.main.Sentiment method), 743
findcorrectionhandling()  (folia.main.Statement method), 629
findreplaceables() (folia.main.Part class method), 283
findreplaceables() (folia.main.PhonContent class method), 526
findreplaceables() (folia.main.PosAnnotation class method), 454
findreplaceables() (folia.main.Predicate class method), 604
findreplaceables() (folia.main.Phrase class method), 297
findreplaceables() (folia.main.Reference class method), 311
findreplaceables() (folia.main.Row class method), 325
findreplaceables() (folia.main.SemanticRole class method), 655
findreplaceables() (foolia.main.SemanticRolesLayer class method), 781
findreplaceables() (foilia.main.SenseAnnotation class method), 490
findreplaceables() (foilia.main.Sentence class method), 341
findreplaceables() (foilia.main.Sentiment class method), 616
findreplaceables() (foilia.main.SentimentLayer class method), 743
findreplaceables() (foilia.main.Statement class method), 629
findreplaceables() (foilia.main.StatementLayer class method), 756
findreplaceables() (foilia.main.SubjectivityAnnotation class method), 502
findreplaceables() (foilia.main.SyntacticUnit class method), 642
findreplaceables() (foilia.main.SyntaxLayer class method), 768
findreplaceables() (foilia.main.Table class method), 355
findreplaceables() (foilia.main.TableHead class method), 383
findreplaceables() (foilia.main.Term class method), 369
findreplaceables() (foilia.main.Text class method), 397
findreplaceables() (foilia.main.TextContent class method), 515
findreplaceables() (foilia.main.TimeSegment class method), 667
findreplaceables() (foilia.main.TimingLayer class method), 793
findreplaceables() (foilia.main.Whitespace class method), 411
findreplaceables() (foilia.main.Word class method), 427
findspan() (foilia.main.AbstractAnnotationLayer class method), 61
findspan() (foilia.main.ChunkingLayer method), 681
findspan() (foilia.main.CoreferenceLayer method), 693
findspan() (foilia.main.DependenciesLayer method), 706
findspan() (foilia.main.EntitiesLayer method), 718
findspan() (foilia.main.ObservationLayer method), 731
findspan() (foilia.main.SemanticRolesLayer method), 781
findspan() (foilia.main.SentimentLayer method), 743
findspan() (foilia.main.StatementLayer method), 756
findspan() (foilia.main.SyntaxLayer method), 768
findspan() (foilia.main.TimingLayer method), 793
findspans() (foilia.main.Word method), 427
findwords() (foilia.main.Document method), 11
foilia.main (module), 1

G
Gap (class in foilia.main), 181
generate_id() (foilia.main.AbstractAnnotationLayer method), 61
generate_id() (foilia.main.AbstractSpanAnnotation method), 48
generate_id() (foilia.main.AbstractStructureElement method), 35
generate_id() (foilia.main.Cell method), 90
generate_id() (foilia.main.Chunk method), 540
generate_id() (foilia.main.ChunkingLayer method), 681
generate_id() (foilia.main.CoreferenceChain method), 553
generate_id() (foilia.main.CoreferenceLayer method), 693
generate_id() (foilia.main.CoreferenceLink method), 806
generate_id() (foilia.main.Definition method), 104
generate_id() (foilia.main.DependenciesLayer method), 706
generate_id() (foilia.main.Dependency method), 565
generate_id() (foilia.main.DependencyDependent method), 819
generate_id() (foilia.main.Division method), 118
generate_id() (foilia.main.DomainAnnotation method), 442
generate_id() (foilia.main.EntitiesLayer method), 719
generate_id() (foilia.main.Entity method), 578
generate_id() (folia.main.Entry method), 132
generate_id() (folia.main.Event method), 146
generate_id() (folia.main.Example method), 160
generate_id() (folia.main.Figure method), 174
generate_id() (folia.main.Head method), 200
generate_id() (folia.main.Headspan method), 832
generate_id() (folia.main.LangAnnotation method), 466
generate_id() (folia.main.LemmaAnnotation method), 478
generate_id() (folia.main.Linebreak method), 214
generate_id() (folia.main.List method), 228
generate_id() (folia.main.ListItem method), 242
generate_id() (folia.main.Note method), 256
generate_id() (folia.main.Observation method), 591
generate_id() (folia.main.ObservationLayer method), 731
generate_id() (folia.main.Paragraph method), 270
generate_id() (folia.main.Part method), 284
generate_id() (folia.main.PosAnnotation method), 454
generate_id() (folia.main.Predicate method), 604
generate_id() (folia.main.Quote method), 298
generate_id() (folia.main.Reference method), 312
generate_id() (folia.main.Row method), 326
generate_id() (folia.main.SemanticRole method), 655
generate_id() (folia.main.SemanticRolesLayer method), 781
generate_id() (folia.main.SenseAnnotation method), 490
generate_id() (folia.main.Sentence method), 341
generate_id() (folia.main.Sentiment method), 617
generate_id() (folia.main.SentimentLayer method), 744
generate_id() (folia.main.Statement method), 629
generate_id() (folia.main.StatementLayer method), 756
generate_id() (folia.main.SubjectivityAnnotation method), 502
generate_id() (folia.main.SyntacticUnit method), 642
generate_id() (folia.main.SyntaxLayer method), 769
generate_id() (folia.main.Table method), 355
generate_id() (folia.main.TableHead method), 383
generate_id() (folia.main.Term method), 369
generate_id() (folia.main.Text method), 397
generate_id() (folia.main.TimeSegment method), 668
generate_id() (folia.main.TimingLayer method), 794
generate_id() (folia.main.Whitespace method), 411
generate_id() (folia.main.Word method), 427
generate_id() (folia.main.Document method), 11
generate_id() (folia.main.Entity method), 146
generate_id() (folia.main.Event method), 427
generate_id() (folia.main.Word method), 427
generate_id() (folia.main.Document method), 12
generate_id() (folia.main.AbstractAnnotationLayer method), 61
generate_id() (folia.main.AbstractElement method), 21
generate_id() (folia.main.AbstractSpanAnnotation method), 48
generate_id() (folia.main.AbstractStructureElement method), 35
generate_id() (folia.main.AbstractTextMarkup method), 73
generate_id() (folia.main.Cell method), 90
generate_id() (folia.main.Chunk method), 540
generate_id() (folia.main.ChunkingLayer method), 681
generate_id() (folia.main.CoreferenceChain method), 553
generate_id() (folia.main.CoreferenceLayer method), 694
generate_id() (folia.main.CoreferenceLink method), 806
generate_id() (folia.main.Definition method), 104
generate_id() (folia.main.DependenciesLayer method), 706
generate_id() (folia.main.Dependency method), 565
generate_id() (folia.main.DependencyDependent method), 819
generate_id() (folia.main.Division method), 118
generate_id() (folia.main.DocumentAnnotation method), 442
generate_id() (folia.main.EntitiesLayer method), 719
generate_id() (folia.main.Entity method), 578
generate_id() (folia.main.Entry method), 132
generate_id() (folia.main.Event method), 146
generate_id() (folia.main.Example method), 160
generate_id() (folia.main.Figure method), 174
generate_id() (folia.main.Gap method), 186
generate_id() (folia.main.Head method), 200
generate_id() (folia.main.Headspan method), 832
generate_id() (folia.mainAGEMENT method), 466
generate_id() (folia.main.LemmaAnnotation method), 478
generate_id() (folia.main.Linebreak method), 214
generate_id() (folia.main.List method), 228
generate_id() (folia.main.ListItem method), 242
generate_id() (folia.main.Note method), 256
generate_id() (folia.main.Observation method), 591
generate_id() (folia.main.ObservationLayer method), 731
generate_id() (folia.main.Paragraph method), 270
generate_id() (folia.main.Part method), 284
generate_id() (folia.main.PhonContent method), 526
getindex() (folia.main.PosAnnotation method), 454
getindex() (folia.main.Predicate method), 604
getindex() (folia.main.Quote method), 298
getindex() (folia.main.Reference method), 312
getindex() (folia.main.Row method), 326
getindex() (folia.main.SemanticRole method), 655
getindex() (folia.main.SemanticRolesLayer method), 781
getindex() (folia.main.SenseAnnotation method), 490
getindex() (folia.main.Sentence method), 341
getindex() (folia.main.Sentiment method), 617
getindex() (folia.main.SentimentLayer method), 744
getindex() (folia.main.Statement method), 629
getindex() (folia.main.StatementLayer method), 756
getindex() (folia.main.SubjectivityAnnotation method), 502
getindex() (folia.main.SyntacticUnit method), 642
getindex() (folia.main.SyntacticUnit method), 769
getindex() (folia.main.Table method), 355
getindex() (folia.main.TableHead method), 383
getindex() (folia.main.Term method), 369
getindex() (folia.main.Text method), 397
getindex() (folia.main.TextContent method), 515
getindex() (folia.main.TimeSegment method), 668
getindex() (folia.main.TimingLayer method), 794
getindex() (folia.main.Whitespace method), 411
getindex() (folia.main.Word method), 427
getmetadata() (folia.main.AbstractAnnotationLayer method), 61
getmetadata() (folia.main.AbstractElement method), 21
getmetadata() (folia.main.AbstractSpanAnnotation method), 48
getmetadata() (folia.main.AbstractStructureElement method), 35
getmetadata() (folia.main.AbstractTextMarkup method), 73
getmetadata() (folia.main.Cell method), 90
getmetadata() (folia.main.Chunk method), 540
getmetadata() (folia.main.ChunkingLayer method), 681
getmetadata() (folia.main.CoreferenceChain method), 553
getmetadata() (folia.main.CoreferenceLayer method), 694
getmetadata() (folia.main.CoreferenceLink method), 806
getmetadata() (folia.main.Definition method), 104
getmetadata() (folia.main.DependenciesLayer method), 706
getmetadata() (folia.main.Dependency method), 566
getmetadata() (folia.main.DependencyDependent method), 819
getmetadata() (folia.main.Division method), 118
getmetadata() (folia.main.DomainAnnotation method), 442
getmetadata() (folia.main.EntitiesLayer method), 719
getmetadata() (folia.main.Entity method), 578
getmetadata() (folia.main.Entry method), 132
getmetadata() (folia.main.Event method), 146
getmetadata() (folia.main.Example method), 160
getmetadata() (folia.main.Figure method), 174
getmetadata() (folia.main.Gap method), 186
getmetadata() (folia.main.Head method), 200
getmetadata() (folia.main.Headspace method), 832
getmetadata() (folia.main.LangAnnotation method), 466
getmetadata() (folia.main.LemmaAnnotation method), 478
getmetadata() (folia.main.Linebreak method), 214
getmetadata() (folia.main.List method), 228
getmetadata() (folia.main.ListItem method), 242
getmetadata() (folia.main.Note method), 256
getmetadata() (folia.main.Observation method), 591
getmetadata() (folia.main.ObservationLayer method), 731
getmetadata() (folia.main.Paragraph method), 270
getmetadata() (folia.main.Part method), 284
getmetadata() (folia.main.PhonContent method), 526
getmetadata() (folia.main.PosAnnotation method), 454
getmetadata() (folia.main.Predicate method), 604
getmetadata() (folia.main.Quote method), 298
getmetadata() (folia.main.Reference method), 312
getmetadata() (folia.main.Row method), 326
getmetadata() (folia.main.SemanticRole method), 655
getmetadata() (folia.main.SemanticRolesLayer method), 781
getmetadata() (folia.main.SenseAnnotation method), 490
getmetadata() (folia.main.Sentence method), 341
getmetadata() (folia.main.Sentiment method), 617
getmetadata() (folia.main.SentimentLayer method), 744
getmetadata() (folia.main.Statement method), 629
getmetadata() (folia.main.StatementLayer method), 756
getmetadata() (folia.main.SubjectivityAnnotation method), 502
getmetadata() (folia.main.SyntacticUnit method), 642
getmetadata() (folia.main.Dependency method), 642
hasphon() (folia.main.Note method), 256
hasphon() (folia.main.Observation method), 591
hasphon() (folia.main.ObservationLayer method), 731
hasphon() (folia.main.Paragraph method), 270
hasphon() (folia.main.Part method), 284
hasphon() (folia.main.PhonContent method), 527
hasphon() (folia.main.Predicate method), 604
hasphon() (folia.main.Predicate method), 298
hasphon() (folia.main.Reference method), 312
hasphon() (folia.main.Row method), 326
hasphon() (folia.main.SemanticRole method), 655
hasphon() (folia.main.SemanticRolesLayer method), 781
hasphon() (folia.main.SemanticRolesLayer method), 781
hasphon() (folia.main.Sentiment method), 617
hasphon() (folia.main.SentimentLayer method), 744
hasphon() (folia.main.Statement method), 630
hasphon() (folia.main.StatementLayer method), 756
hasphon() (folia.main.SubjectivityAnnotation method), 502
hasphon() (folia.main.SyntaxLayer method), 642
hasphon() (folia.main.SyntaxLayer method), 769
hasphon() (folia.main.Table method), 355
hasphon() (folia.main.TableHead method), 383
hasphon() (folia.main.Term method), 369
hasphon() (folia.main.Text method), 397
hasphon() (folia.main.TextContent method), 515
hasphon() (folia.main.TimeSegment method), 668
hasphon() (folia.main.TimeSegment method), 794
hasphon() (folia.main.Whitespace method), 411
hasphon() (folia.main.Word method), 428
hasprocessors() (folia.main.Whitespace method), 428
hastext() (folia.main.Document method), 12
hastext() (folia.main.AbstractAnnotationLayer method), 62
hastext() (folia.main.AbstractElement method), 22
hastext() (folia.main.AbstractSpanAnnotation method), 49
hastext() (folia.main.AbstractStructureElement method), 36
hastext() (folia.main.AbstractTextMarkup method), 73
hastext() (folia.main.Cell method), 90
hastext() (folia.main.Chunk method), 540
hastext() (folia.main.ChunkingLayer method), 682
hastext() (folia.main.CoreferenceChain method), 553
hastext() (folia.main.CoreferenceLayer method), 694
hastext() (folia.main.CoreferenceLink method), 807
hastext() (folia.main.Definition method), 104
hastext() (folia.main.DependenciesLayer method), 707
hastext() (folia.main.Dependency method), 566
hastext() (folia.main.DependencyDependentmethod), 820
hastext() (folia.main.Division method), 118
hastext() (folia.main.DominantAnnotation method), 443
hastext() (folia.main.EntitiesLayer method), 719
hastext() (folia.main.Entry method), 132
hastext() (folia.main.Event method), 146
hastext() (folia.main.Event method), 160
hastext() (folia.main.Figure method), 174
hastext() (folia.main.Line method), 187
hastext() (folia.main.Head method), 200
hastext() (folia.main.HeadSpan method), 833
hastext() (folia.main.Linebreak method), 214
hastext() (folia.main.List method), 228
hastext() (folia.main.ListItem method), 242
hastext() (folia.main.Note method), 256
hastext() (folia.main.Observation method), 592
hastext() (folia.main.ObservationLayer method), 732
hastext() (folia.main.Paragraph method), 270
hastext() (folia.main.Paragraph method), 284
hastext() (folia.main.PhonContent method), 527
hastext() (folia.main.Predicate method), 455
hastext() (folia.main.Predicate method), 604
hastext() (folia.main.Predicate method), 298
hastext() (folia.main.Reference method), 312
hastext() (folia.main.Row method), 326
hastext() (folia.main.Reference method), 312
hastext() (folia.main.Region method), 479
hastext() (folia.main.SemanticRolesLayer method), 655
hastext() (folia.main.SemanticRolesLayer method), 782
hastext() (folia.main.SenseAnnotation method), 491
hastext() (folia.main.Sentence method), 341
hastext() (folia.main.Sentiment method), 617
hastext() (folia.main.SentimentLayer method), 744
hastext() (folia.main.Statement method), 630
hastext() (folia.main.Statement method), 757
hastext() (folia.main.Statement method), 503
hastext() (folia.main.SyntacticUnit method), 643
hastext() (folia.main.SyntaxLayer method), 769
hastext() (folia.main.Table method), 356
hastext() (folia.main.TableHead method), 384
hastext() (folia.main.Term method), 370
hastext() (folia.main.Text method), 398
hastext() (folia.main.TextContent method), 515
hastext() (folia.main.TextContent method), 515
hastext() (folia.main.TextContent method), 668
hastext() (folia.main.TimingLayer method), 794
hastext() (folia.main.TimingLayer method), 794
hastext() (folia.main.TimingLayer method), 794
hastext() (folia.main.TimingLayer method), 794
hasprocessors() (folia.main.Word method), 707
Head (class in folia.main), 193
Index 895
head() (folia.main.Dependency method), 566
head() (folia.main.Division method), 119
Headspan (class in folia.main), 826
HIDDEN (folia.main.AbstractAnnotationLayer attribute), 58
HIDDEN (folia.main.AbstractElement attribute), 19
HIDDEN (folia.main.AbstractSpanAnnotation attribute), 45
HIDDEN (folia.main.AbstractStructureElement attribute), 31
HIDDEN (folia.main.AbstractTextMarkup attribute), 70
HIDDEN (folia.main.Cell attribute), 86
HIDDEN (folia.main.Chunk attribute), 537
HIDDEN (folia.main.ChunkingLayer attribute), 678
HIDDEN (folia.main.CoreferenceChain attribute), 550
HIDDEN (folia.main.CoreferenceLayer attribute), 691
HIDDEN (folia.main.CoreferenceLink attribute), 803
HIDDEN (folia.main.Definition attribute), 100
HIDDEN (folia.main.DependenciesLayer attribute), 703
HIDDEN (folia.main.Dependency attribute), 563
HIDDEN (folia.main.DependencyDependent attribute), 816
HIDDEN (folia.main.Division attribute), 114
HIDDEN (folia.main.DomainAnnotation attribute), 440
HIDDEN (folia.main.EntitiesLayer attribute), 716
HIDDEN (folia.main.Entity attribute), 575
HIDDEN (folia.main.Entry attribute), 128
HIDDEN (folia.main.Event attribute), 142
HIDDEN (folia.main.Example attribute), 156
HIDDEN (folia.main.Figure attribute), 170
HIDDEN (folia.main.Gap attribute), 183
HIDDEN (folia.main.Head attribute), 196
HIDDEN (folia.main.Headspan attribute), 829
HIDDEN (folia.main.LangAnnotation attribute), 464
HIDDEN (folia.main.LemmaAnnotation attribute), 476
HIDDEN (folia.main.Linebreak attribute), 210
HIDDEN (folia.main.List attribute), 224
HIDDEN (folia.main.ListItem attribute), 238
HIDDEN (folia.main.Note attribute), 252
HIDDEN (folia.main.Observation attribute), 588
HIDDEN (folia.main.ObservationLayer attribute), 728
HIDDEN (folia.main.Paragraph attribute), 266
HIDDEN (folia.main.Part attribute), 280
HIDDEN (folia.main.PhonContent attribute), 524
HIDDEN (folia.main.PosAnnotation attribute), 452
HIDDEN (folia.main.Predicate attribute), 601
HIDDEN (folia.main.Quote attribute), 294
HIDDEN (folia.main.Reference attribute), 308
HIDDEN (folia.main.Row attribute), 322
HIDDEN (folia.main.SemanticRole attribute), 652
HIDDEN (folia.main.SemanticRolesLayer attribute), 778
HIDDEN (folia.main.SenseAnnotation attribute), 488
HIDDEN (folia.main.Statement attribute), 336
HIDDEN (folia.main.Sentence attribute), 614
HIDDEN (folia.main.SentimentLayer attribute), 741
HIDDEN (folia.main.Statement attribute), 627
HIDDEN (folia.main.StatementLayer attribute), 753
HIDDEN (folia.main.SubjectivityAnnotation attribute), 500
HIDDEN (folia.main.SyntacticUnit attribute), 639
HIDDEN (folia.main.SyntaxLayer attribute), 766
HIDDEN (folia.main.Table attribute), 351
HIDDEN (folia.main.TableHead attribute), 379
HIDDEN (folia.main.Term attribute), 365
HIDDEN (folia.main.Text attribute), 393
HIDDEN (folia.main.TextContent attribute), 512
HIDDEN (folia.main.TimeSegment attribute), 665
HIDDEN (folia.main.TimingLayer attribute), 791
HIDDEN (folia.main.Whitespace attribute), 407
HIDDEN (folia.main.Word attribute), 422

inCorrection() (folia.main.AbstractAnnotationLayer method), 62
inCorrection() (folia.main.AbstractElement method), 22
inCorrection() (folia.main.AbstractSpanAnnotation method), 49
inCorrection() (folia.main.AbstractStructureElement method), 36
inCorrection() (folia.main.AbstractTextMarkup method), 74
inCorrection() (folia.main.Cell method), 91
inCorrection() (folia.main.Chunk method), 541
inCorrection() (folia.main.ChunkingLayer method), 682
inCorrection() (folia.main.CoreferenceChain method), 553
inCorrection() (folia.main.CoreferenceLayer method), 694
inCorrection() (folia.main.CoreferenceLink method), 807
inCorrection() (folia.main.Definition method), 105
inCorrection() (folia.main.DependenciesLayer method), 707
inCorrection() (folia.main.Dependency method), 566
inCorrection() (folia.main.DependencyDependent method), 820
inCorrection() (folia.main.Division method), 119
inCorrection() (folia.main.DomainAnnotation method), 443
inCorrection() (folia.main.EntitiesLayer method), 719
inCorrection() (folia.main.Entity method), 579
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>folia.main.Entry method</td>
<td>133</td>
</tr>
<tr>
<td>folia.main.Event method</td>
<td>147</td>
</tr>
<tr>
<td>folia.main.Example method</td>
<td>161</td>
</tr>
<tr>
<td>folia.main.Figure method</td>
<td>175</td>
</tr>
<tr>
<td>folia.main.Gap method</td>
<td>187</td>
</tr>
<tr>
<td>folia.main.Head method</td>
<td>201</td>
</tr>
<tr>
<td>folia.main.Headspan method</td>
<td>833</td>
</tr>
<tr>
<td>folia.main.LangAnnotation method</td>
<td>467</td>
</tr>
<tr>
<td>folia.main.LemmaAnnotation method</td>
<td>479</td>
</tr>
<tr>
<td>folia.main.Linebreak method</td>
<td>215</td>
</tr>
<tr>
<td>folia.main.List method</td>
<td>229</td>
</tr>
<tr>
<td>folia.main.ListItem method</td>
<td>243</td>
</tr>
<tr>
<td>folia.main.Note method</td>
<td>257</td>
</tr>
<tr>
<td>folia.main.Observation method</td>
<td>592</td>
</tr>
<tr>
<td>folia.main.ObservationLayer method</td>
<td>732</td>
</tr>
<tr>
<td>folia.main.Paragraph method</td>
<td>271</td>
</tr>
<tr>
<td>folia.main.Part method</td>
<td>285</td>
</tr>
<tr>
<td>folia.main.Phrase method</td>
<td>327</td>
</tr>
<tr>
<td>folia.main.Predicate method</td>
<td>605</td>
</tr>
<tr>
<td>folia.main.PhraseContent method</td>
<td>656</td>
</tr>
<tr>
<td>folia.main.Sentiment method</td>
<td>617</td>
</tr>
<tr>
<td>folia.main.SentimentLayer method</td>
<td>744</td>
</tr>
<tr>
<td>folia.main.Statement method</td>
<td>630</td>
</tr>
<tr>
<td>folia.main.StatementLayer method</td>
<td>757</td>
</tr>
<tr>
<td>folia.main.SubjectivityAnnotation method</td>
<td>503</td>
</tr>
<tr>
<td>folia.main.SyntacticUnit method</td>
<td>643</td>
</tr>
<tr>
<td>folia.main.SyntaxLayer method</td>
<td>769</td>
</tr>
<tr>
<td>folia.main.Table method</td>
<td>356</td>
</tr>
<tr>
<td>folia.main.TableHead method</td>
<td>384</td>
</tr>
<tr>
<td>folia.main.Term method</td>
<td>370</td>
</tr>
<tr>
<td>folia.main.Text method</td>
<td>398</td>
</tr>
<tr>
<td>folia.main.TextContent method</td>
<td>516</td>
</tr>
</tbody>
</table>
insert() (folia.main.Row method), 327
insert() (folia.main.SemanticRole method), 656
insert() (folia.main.SemanticRolesLayer method), 782
insert() (folia.main.SenseAnnotation method), 491
insert() (folia.main.Sentence method), 342
insert() (folia.main.Sentiment method), 618
insert() (folia.main.SentimentLayer method), 745
insert() (folia.main.Statement method), 630
insert() (folia.main.StatementLayer method), 757
insert() (folia.main.SubjectivityAnnotation method), 503
insert() (folia.main.SyntacticUnit method), 643
insert() (folia.main.SyntaxLayer method), 770
insert() (folia.main.Table method), 356
insert() (folia.main.TableHead method), 384
insert() (folia.main.Term method), 370
insert() (folia.main.Text method), 398
insert() (folia.main.TextContent method), 516
insert() (folia.main.TimeSegment method), 669
insert() (folia.main.TimingLayer method), 795
insert() (folia.main.WhiteSpace method), 412
insert() (folia.main.Word method), 428
insertword() (folia.main.Sentence method), 342
insertwordleft() (folia.main.Sentence method), 342
items() (folia.main.AbstractAnnotationLayer method), 62
items() (folia.main.AbstractElement method), 22
items() (folia.main.AbstractSpanAnnotation method), 49
items() (folia.main.AbstractStructureElement method), 36
items() (folia.main.AbstractTextMarkup method), 74
items() (folia.main.Cell method), 91
items() (folia.main.Chunk method), 541
items() (folia.main.ChunkingLayer method), 682
items() (folia.main.CoreferenceChain method), 554
items() (folia.main.CoreferenceLayer method), 695
items() (folia.main.CoreferenceLink method), 807
items() (folia.main.Definition method), 105
items() (folia.main.DependenciesLayer method), 707
items() (folia.main.Dependency method), 566
items() (folia.main.DependencyDependent method), 820
items() (folia.main.Division method), 119
items() (folia.main.Document method), 12
items() (folia.main.DomainAnnotation method), 443
items() (folia.main.EntitiesLayer method), 720
items() (folia.main.Entity method), 579
items() (folia.main.Entry method), 133
items() (folia.main.Event method), 147
items() (folia.main.Example method), 161
items() (folia.main.Figure method), 175
items() (folia.main.Gap method), 187
items() (folia.main.Head method), 201
items() (folia.main.Headspace method), 833
items() (folia.main.LangAnnotation method), 467
items() (folia.main.LemmaAnnotation method), 479
items() (folia.main.Linebreak method), 215
items() (folia.main.List method), 229
items() (folia.main.ListItem method), 243
items() (folia.main.Note method), 257
items() (folia.main.Observation method), 592
items() (folia.main.ObservationLayer method), 732
items() (folia.main.Paragraph method), 271
items() (folia.main.Part method), 285
items() (folia.main.PhyContent method), 527
items() (folia.main.PosAnnotation method), 455
items() (folia.main.Predicate method), 605
items() (folia.main.Quote method), 299
items() (folia.main.Reference method), 313
items() (folia.main.Row method), 327
items() (folia.main.SemanticRole method), 656
items() (folia.main.SemanticRolesLayer method), 782
items() (folia.main.SenseAnnotation method), 491
items() (folia.main.Sentence method), 342
items() (folia.main.Sentiment method), 618
items() (folia.main.SentimentLayer method), 745
items() (folia.main.Statement method), 630
items() (folia.main.StatementLayer method), 757
items() (folia.main.SubjectivityAnnotation method), 503
items() (folia.main.SyntacticUnit method), 643
items() (folia.main.SyntaxLayer method), 770
items() (folia.main.Table method), 356
items() (folia.main.TableHead method), 384
items() (folia.main.Term method), 370
items() (folia.main.Text method), 398
items() (folia.main.TimeSegment method), 669
items() (folia.main.TimingLayer method), 795
items() (folia.main.WhiteSpace method), 412
items() (folia.main.Word method), 428
J
json() (folia.main.AbstractAnnotationLayer method), 62
json() (folia.main.AbstractElement method), 22
json() (folia.main.AbstractSpanAnnotation method), 49
json() (folia.main.AbstractStructureElement method), 36
json() (folia.main.AbstractTextMarkup method), 74
json() (folia.main.Cell method), 91
json() (folia.main.Chunk method), 541
json() (folia.main.ChunkingLayer method), 682
json() (folia.main.CoreferenceChain method), 554
json() (folia.main.CoreferenceLayer method), 695
json() (folia.main.CoreferenceLink method), 807
json() (folia.main.Definition method), 105
json() (folia.main.DependenciesLayer method), 707
json() (folia.main.Dependency method), 566
json() (folia.main.DependencyDependent method), 820
json() (folia.main.Division method), 119
json() (folia.main.Document method), 12
json() (folia.main.DomainAnnotation method), 443
json() (folia.main.EntitiesLayer method), 720
json() (folia.main.Entity method), 579
json() (folia.main.Entry method), 133
json() (folia.main.Event method), 147
json() (folia.main.Example method), 161
json() (folia.main.Figure method), 175
Index 899
LABEL (folia.main.Whitespace attribute), 407
LABEL (folia.main.Word attribute), 422
LangAnnotation (class in folia.main), 461
language() (folia.main.Document method), 12
layer() (folia.main.AbstractSpanAnnotation method), 49
layer() (folia.main.Chunk method), 541
layer() (folia.main.CoreferenceChain method), 554
layer() (folia.main.CoreferenceLink method), 808
layer() (folia.main.Dependency method), 567
layer() (folia.main.DependencyDependent method), 820
layer() (folia.main.Entity method), 580
layer() (folia.main.Headspan method), 833
layer() (folia.main.Observation method), 592
layer() (folia.main.Predicate method), 605
layer() (folia.main.SemanticRole method), 656
layer() (folia.main.Sentiment method), 618
layer() (folia.main.Statement method), 631
layer() (folia.main.SyntacticUnit method), 643
layer() (folia.main.TimeSegment method), 669
layers() (folia.main.AbstractStructureElement method), 36
layers() (folia.main.Cell method), 91
layers() (folia.main.Definition method), 105
layers() (folia.main.Division method), 119
layers() (folia.main.Entry method), 133
layers() (folia.main.Event method), 147
layers() (folia.main.Example method), 161
layers() (folia.main.Figure method), 175
layers() (folia.main.Head method), 201
layers() (folia.main.Linebreak method), 215
layers() (folia.main.List method), 229
layers() (folia.main.ListItem method), 243
layers() (folia.main.Note method), 257
layers() (folia.main.Paragraph method), 271
layers() (folia.main.Part method), 285
layers() (folia.main.Quote method), 299
layers() (folia.main.Reference method), 313
layers() (folia.main.Row method), 327
layers() (folia.main.Sentence method), 342
layers() (folia.main.Table method), 356
layers() (folia.main.TableHead method), 384
layers() (folia.main.Term method), 370
layers() (folia.main.Text method), 398
layers() (folia.main.Whitespace method), 412
layers() (folia.main.Word method), 429
leftcontext() (folia.main.AbstractAnnotationLayer method), 62
leftcontext() (folia.main.AbstractElement method), 23
leftcontext() (folia.main.AbstractSpanAnnotation method), 49
leftcontext() (folia.main.AbstractTextMarkup method), 74
leftcontext() (folia.main.Cell method), 91
leftcontext() (folia.main.Chunk method), 541
leftcontext() (folia.main.ChunkingLayer method), 682
leftcontext() (folia.main.CoreferenceChain method), 554
leftcontext() (folia.main.CoreferenceLayer method), 695
leftcontext() (folia.main.CoreferenceLink method), 808
leftcontext() (folia.main.Definition method), 105
leftcontext() (folia.main.DependenciesLayer method), 707
leftcontext() (folia.main.Dependency method), 567
leftcontext() (folia.main.DependencyDependent method), 820
leftcontext() (folia.main.Division method), 119
leftcontext() (folia.main.DomainAnnotation method), 443
leftcontext() (folia.main.EntitiesLayer method), 720
leftcontext() (folia.main.Entity method), 580
leftcontext() (folia.main.Entry method), 133
leftcontext() (folia.main.Event method), 147
leftcontext() (folia.main.Example method), 161
leftcontext() (folia.main.Figure method), 175
leftcontext() (folia.main.Gap method), 187
leftcontext() (folia.main.Head method), 201
leftcontext() (folia.main.Headspan method), 833
leftcontext() (folia.main.LangAnnotation method), 467
leftcontext() (folia.main.LemmaAnnotation method), 479
leftcontext() (folia.main.Linebreak method), 215
leftcontext() (folia.main.List method), 229
leftcontext() (folia.main.ListItem method), 243
leftcontext() (folia.main.Note method), 257
leftcontext() (folia.main.Observation method), 592
leftcontext() (folia.main.ObservationLayer method), 732
leftcontext() (folia.main.Paragraph method), 271
leftcontext() (folia.main.Part method), 285
leftcontext() (folia.main.PhonContent method), 528
leftcontext() (folia.main.PosAnnotation method), 455
leftcontext() (folia.main.Predicate method), 605
leftcontext() (folia.main.Quote method), 299
leftcontext() (folia.main.Reference method), 313
leftcontext() (folia.main.Row method), 327
leftcontext() (folia.main.SemanticRole method), 656
leftcontext() (folia.main.SemanticRolesLayer method), 782
leftcontext() (folia.main.SenseAnnotation method), 491
leftcontext() (folia.main.Statement method), 342
leftcontext() (folia.main.Sentiment method), 618
leftcontext() (folia.main.SentimentLayer method), 745
leftcontext() (folia.main.Statement method), 631
leftcontext() (folia.main.StatementLayer method), 757
leftcontext() (folia.main.SubjectivityAnnotation method), 503
leftcontext() (folia.main.SyntacticUnit method), 643
leftcontext() (folia.main.SyntaxLayer method), 770
leftcontext() (folia.main.Table method), 356
leftcontext() (folia.main.TableHead method), 384
leftcontext() (folia.main.Term method), 370
leftcontext() (folia.main.Text method), 398
leftcontext() (folia.main.TextContent method), 516
leftcontext() (folia.main.TimeSegment method), 669
leftcontext() (folia.main.TimingLayer method), 795
leftcontext() (folia.main.Whitespace method), 412
lemma() (folia.main.Word method), 429
LemmaAnnotation (class in folia.main), 473
license() (folia.main.Document method), 12
Linebreak (class in folia.main), 207
ListItem (class in folia.main), 221
ListItem (class in folia.main), 235
load() (folia.main.Document method), 12

M
mergewords() (folia.main.Statement method), 342
morpheme() (folia.main.Word method), 429
morphemes() (folia.main.Word method), 429

N
next() (folia.main.AbstractAnnotationLayer method), 62
next() (folia.main.AbstractElement method), 23
next() (folia.main.AbstractSpanAnnotation method), 49
next() (folia.main.AbstractStructureElement method), 36
next() (folia.main.AbstractTextMarkup method), 74
next() (folia.main.Cell method), 91
next() (folia.main.Chunk method), 541
next() (folia.main.ChunkingLayer method), 682
next() (folia.main.CoreferenceChain method), 554
next() (folia.main.CoreferenceLayer method), 695
next() (folia.main.CoreferenceLink method), 808
next() (folia.main.Definition method), 105
next() (folia.main.DependenciesLayer method), 707
next() (folia.main.Dependency method), 567
next() (folia.main.DependencyDependent method), 820
next() (folia.main.Division method), 119
next() (folia.main.DomainAnnotation method), 443
next() (folia.main.EntitiesLayer method), 720
next() (folia.main.Entity method), 580
next() (folia.main.Event method), 147
next() (folia.main.Example method), 161
next() (folia.main.Figure method), 175
next() (folia.main.Gap method), 187
next() (folia.main.Head method), 201
next() (folia.main.Headspan method), 833
next() (folia.main.LangAnnotation method), 467
next() (folia.main.LemmaAnnotation method), 479
next() (folia.main.Linebreak method), 215
next() (folia.main.List method), 229
next() (folia.main.ListItem method), 243
next() (folia.main.Note method), 257
next() (folia.main.Observation method), 592
next() (folia.main.ObservationLayer method), 732
next() (folia.main.Paragraph method), 271
next() (folia.main.Part method), 285
next() (folia.main.Phrase method), 528
next() (folia.main.PosAnnotation method), 455
next() (folia.main.Predicate method), 605
next() (folia.main.Quote method), 299
next() (folia.main.Reference method), 313
next() (folia.main.Row method), 327
next() (folia.main.SemanticRole method), 656
next() (folia.main.SemanticRolesLayer method), 782
next() (folia.main.SenseAnnotation method), 491
next() (folia.main.Statement method), 342
next() (folia.main.Sentiment method), 618
next() (folia.main.SentimentLayer method), 745
next() (folia.main.Statement method), 631
next() (folia.main.StatementLayer method), 757
next() (folia.main.SubjectivityAnnotation method), 503
next() (folia.main.SyntacticUnit method), 643
next() (folia.main.SyntaxLayer method), 770
next() (folia.main.Table method), 357
next() (folia.main.TableHead method), 385
next() (folia.main.Term method), 371
Observation (class in folia.main), 585
ObservationLayer (class in folia.main), 725
OCCURRENCES (folia.main.AbstractAnnotationLayer attribute), 58
OCCURRENCES (folia.main.AbstractElement attribute), 19
OCCURRENCES (folia.main.AbstractSpanAnnotation attribute), 45
OCCURRENCES (folia.main.AbstractStructureElement attribute), 31
OCCURRENCES (folia.main.AbstractTextMarkup attribute), 70
OCCURRENCES (folia.main.Cell attribute), 86
OCCURRENCES (folia.main.Chunk attribute), 537
OCCURRENCES (folia.main.ChunkingLayer attribute), 678
OCCURRENCES (folia.main.CoreferenceChain attribute), 550
OCCURRENCES (folia.main.CoreferenceLayer attribute), 691
OCCURRENCES (folia.main.CoreferenceLink attribute), 803
OCCURRENCES (folia.main.Definition attribute), 100
OCCURRENCES (folia.main.DependenciesLayer attribute), 703
OCCURRENCES (folia.main.Dependency attribute), 563
OCCURRENCES (folia.main.DependencyDependent attribute), 816
OCCURRENCES (folia.main.Division attribute), 114
OCCURRENCES (folia.main.DomainAnnotation attribute), 440
OCCURRENCES (folia.main.EntitiesLayer attribute), 716
OCCURRENCES (folia.main.Entity attribute), 576
OCCURRENCES (folia.main.Entry attribute), 128
OCCURRENCES (folia.main.Event attribute), 142
OCCURRENCES (folia.main.Example attribute), 156
OCCURRENCES (folia.main.Figure attribute), 170
OCCURRENCES (folia.main.Gap attribute), 183
OCCURRENCES (folia.main.Head attribute), 196
OCCURRENCES (folia.main.Headspan attribute), 829
OCCURRENCES (folia.main.LangAnnotation attribute), 464
OCCURRENCES (folia.main.LemmaAnnotation attribute), 476
OCCURRENCES (folia.main.Linebreak attribute), 210
OCCURRENCES (folia.main.List attribute), 224
OCCURRENCES (folia.main.ListItem attribute), 238
OCCURRENCES (folia.main.Note attribute), 252
OCCURRENCES (folia.main.Observation attribute), 588
OCCURRENCES (folia.main.ObservationLayer attribute), 728
OCCURRENCES (folia.main.Paragraph attribute), 266
OCCURRENCES (folia.main.Part attribute), 280
OCCURRENCES (folia.main.PhonContent attribute), 524
OCCURRENCES (folia.main.PosAnnotation attribute), 452
OCCURRENCES (folia.main.Predicate attribute), 601
OCCURRENCES (folia.main.Quote attribute), 294
OCCURRENCES (folia.main.Reference attribute), 308
OCCURRENCES (folia.main.Row attribute), 322
OCCURRENCES (folia.main.SemanticRole attribute), 652
OCCURRENCES (folia.main.SemanticRolesLayer attribute), 778
OCCURRENCES (folia.main.SenseAnnotation attribute), 488
OCCURRENCES (folia.main.Sentence attribute), 336
OCCURRENCES (folia.main.Sentiment attribute), 614
OCCURRENCES (folia.main.SentimentLayer attribute), 741
OCCURRENCES (folia.main.Statement attribute), 627
OCCURRENCES (folia.main.StatementLayer attribute), 753
OCCURRENCES (folia.main.SubjectivityAnnotation attribute), 500
OCCURRENCES (folia.main.SyntacticUnit attribute), 639
OCCURRENCES (folia.main.SyntaxLayer attribute), 766
OCCURRENCES (folia.main.Table attribute), 351
OCCURRENCES (folia.main.TableHead attribute), 379
OCCURRENCES (folia.main.Term attribute), 365
OCCURRENCES (folia.main.Text attribute), 393
OCCURRENCES (folia.main.TextContent attribute), 512
OCCURRENCES (folia.main.TimeSegment attribute), 665
OCCURRENCES (folia.main.TimingLayer attribute), 791
OCCURRENCES (folia.main.Whitespace attribute), 407
OCCURRENCES (folia.main.Word attribute), 422
OCCURRENCES_PER_SET (folia.main.AbstractAnnotationLayer attribute), 58
OCCURRENCES_PER_SET (folia.main.AbstractElement attribute), 19
OCCURRENCES_PER_SET (folia.main.AbstractSpanAnnotation attribute), 45
OCCURRENCES_PER_SET (folia.main.AbstractStructureElement attribute), 31
OCCURRENCES_PER_SET (folia.main.AbstractTextMarkup attribute), 71
<table>
<thead>
<tr>
<th>Attribute Description</th>
<th>Occurrences Per Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>folia.main.Cell attribute</td>
<td>86</td>
</tr>
<tr>
<td>folia.main.Chunk attribute</td>
<td>537</td>
</tr>
<tr>
<td>folia.main.ChunkingLayer attribute</td>
<td>678</td>
</tr>
<tr>
<td>folia.main.CoreferenceChain attribute</td>
<td>550</td>
</tr>
<tr>
<td>folia.main.CoreferenceLayer attribute</td>
<td>691</td>
</tr>
<tr>
<td>folia.main.CoreferenceLink attribute</td>
<td>804</td>
</tr>
<tr>
<td>folia.main.Definition attribute</td>
<td>100</td>
</tr>
<tr>
<td>folia.main.DependenciesLayer attribute</td>
<td>703</td>
</tr>
<tr>
<td>folia.main.Dependency attribute</td>
<td>563</td>
</tr>
<tr>
<td>folia.main.DependencyDependent attribute</td>
<td>816</td>
</tr>
<tr>
<td>folia.main.Division attribute</td>
<td>114</td>
</tr>
<tr>
<td>folia.main.DomainAnnotation attribute</td>
<td>440</td>
</tr>
<tr>
<td>folia.main.EntitiesLayer attribute</td>
<td>716</td>
</tr>
<tr>
<td>folia.main.Entity attribute</td>
<td>576</td>
</tr>
<tr>
<td>folia.main.Entry attribute</td>
<td>128</td>
</tr>
<tr>
<td>folia.main.Event attribute</td>
<td>142</td>
</tr>
<tr>
<td>folia.main.Example attribute</td>
<td>156</td>
</tr>
<tr>
<td>folia.main.Figure attribute</td>
<td>170</td>
</tr>
<tr>
<td>folia.main.Gap attribute</td>
<td>184</td>
</tr>
<tr>
<td>folia.main.Head attribute</td>
<td>196</td>
</tr>
<tr>
<td>folia.main.Headspan attribute</td>
<td>829</td>
</tr>
<tr>
<td>folia.main.LangAnnotation attribute</td>
<td>464</td>
</tr>
<tr>
<td>folia.main.LemmaAnnotation attribute</td>
<td>476</td>
</tr>
<tr>
<td>folia.main.Linebreak attribute</td>
<td>210</td>
</tr>
<tr>
<td>folia.main.List attribute</td>
<td>224</td>
</tr>
<tr>
<td>folia.main.ListItem attribute</td>
<td>238</td>
</tr>
<tr>
<td>folia.main.Note attribute</td>
<td>252</td>
</tr>
<tr>
<td>folia.main.Observation attribute</td>
<td>588</td>
</tr>
<tr>
<td>folia.main.ObservationLayer attribute</td>
<td>728</td>
</tr>
<tr>
<td>folia.main.Paragraph attribute</td>
<td>266</td>
</tr>
<tr>
<td>folia.main.Part attribute</td>
<td>280</td>
</tr>
<tr>
<td>folia.main.Phrase attribute</td>
<td>524</td>
</tr>
<tr>
<td>folia.main.PosAnnotation attribute</td>
<td>452</td>
</tr>
<tr>
<td>folia.main.Predicate attribute</td>
<td>601</td>
</tr>
<tr>
<td>folia.main.PhoneContent attribute</td>
<td>294</td>
</tr>
<tr>
<td>folia.main.Reference attribute</td>
<td>308</td>
</tr>
<tr>
<td>folia.main.Row attribute</td>
<td>322</td>
</tr>
<tr>
<td>folia.main.Sentence attribute</td>
<td>336</td>
</tr>
<tr>
<td>folia.main.Sentiment attribute</td>
<td>614</td>
</tr>
<tr>
<td>folia.main.SentimentLayer attribute</td>
<td>741</td>
</tr>
<tr>
<td>folia.main.StatementLayer attribute</td>
<td>627</td>
</tr>
<tr>
<td>folia.main.Statement attribute</td>
<td>753</td>
</tr>
<tr>
<td>folia.main.SubjectivityAnnotation attribute</td>
<td>500</td>
</tr>
<tr>
<td>folia.main.SyntaxUnit attribute</td>
<td>639</td>
</tr>
<tr>
<td>folia.main.Table attribute</td>
<td>766</td>
</tr>
<tr>
<td>folia.main.TableHead attribute</td>
<td>351</td>
</tr>
<tr>
<td>folia.main.Text attribute</td>
<td>379</td>
</tr>
<tr>
<td>folia.main.TextContent attribute</td>
<td>365</td>
</tr>
<tr>
<td>folia.main.TextContent attribute</td>
<td>393</td>
</tr>
</tbody>
</table>
tribute), 512
OCCURRENCES_PER_SET (folia.main.TimeSegment attribute), 665
OCCURRENCES_PER_SET (folia.main.TimingLayer attribute), 791
OCCURRENCES_PER_SET (folia.main.Whitespace attribute), 407
OCCURRENCES_PER_SET (folia.main.Word attribute), 422
OPTIONAL_ATTRIBS (folia.main.AbstractAnnotationLayer attribute), 58
OPTIONAL_ATTRIBS (folia.main.AbstractElement attribute), 19
OPTIONAL_ATTRIBS (folia.main.AbstractSpanAnnotation attribute), 45
OPTIONAL_ATTRIBS (folia.main.AbstractStructureElement attribute), 31
OPTIONAL_ATTRIBS (folia.main.AbstractTextMarkup attribute), 71
OPTIONAL_ATTRIBS (folia.main.Cell attribute), 86
OPTIONAL_ATTRIBS (folia.main.Chunk attribute), 537
OPTIONAL_ATTRIBS (folia.main.ChunkingLayer attribute), 678
OPTIONAL_ATTRIBS (folia.main.CoreferenceChain attribute), 550
OPTIONAL_ATTRIBS (folia.main.CoreferenceLayer attribute), 691
OPTIONAL_ATTRIBS (folia.main.CoreferenceLink attribute), 804
OPTIONAL_ATTRIBS (folia.main.Definition attribute), 100
OPTIONAL_ATTRIBS (folia.main.DependenciesLayer attribute), 703
OPTIONAL_ATTRIBS (folia.main.Dependency attribute), 563
OPTIONAL_ATTRIBS (folia.main.DependencyDependent attribute), 816
OPTIONAL_ATTRIBS (folia.main.Division attribute), 114
OPTIONAL_ATTRIBS (folia.main.DomainsAnnotation attribute), 440
OPTIONAL_ATTRIBS (folia.main.EntitiesLayer attribute), 716
OPTIONAL_ATTRIBS (folia.main.Entity attribute), 576
OPTIONAL_ATTRIBS (folia.main.Entry attribute), 128
OPTIONAL_ATTRIBS (folia.main.Event attribute), 142
OPTIONAL_ATTRIBS (folia.main.Example attribute), 156
OPTIONAL_ATTRIBS (folia.main.Figure attribute), 170
OPTIONAL_ATTRIBS (folia.main.Gap attribute), 184
OPTIONAL_ATTRIBS (folia.main.Head attribute), 196
OPTIONAL_ATTRIBS (folia.main.Headspan attribute), 829
OPTIONAL_ATTRIBS (folia.main.LangAnnotation attribute), 464
OPTIONAL_ATTRIBS (folia.main.LemmaAnnotation attribute), 476
OPTIONAL_ATTRIBS (folia.main.Linebreak attribute), 210
OPTIONAL_ATTRIBS (folia.main.List attribute), 224
OPTIONAL_ATTRIBS (folia.main.ListItem attribute), 238
OPTIONAL_ATTRIBS (folia.main.Note attribute), 252
OPTIONAL_ATTRIBS (folia.main.Observation attribute), 588
OPTIONAL_ATTRIBS (folia.main.ObservationLayer attribute), 728
OPTIONAL_ATTRIBS (folia.main.Paragraph attribute), 266
OPTIONAL_ATTRIBS (folia.main.Part attribute), 280
OPTIONAL_ATTRIBS (folia.main.PhraseContent attribute), 524
OPTIONAL_ATTRIBS (folia.main.PosAnnotation attribute), 452
OPTIONAL_ATTRIBS (folia.main.Predicate attribute), 601
OPTIONAL_ATTRIBS (folia.main.Punctuation attribute), 294
OPTIONAL_ATTRIBS (folia.main.Reference attribute), 308
OPTIONAL_ATTRIBS (folia.main.Row attribute), 322
OPTIONAL_ATTRIBS (folia.main.SemanticRole attribute), 652
OPTIONAL_ATTRIBS (folia.main.SemanticRolesLayer attribute), 778
OPTIONAL_ATTRIBS (folia.main.SenseAnnotation attribute), 488
OPTIONAL_ATTRIBS (folia.main.Statement attribute), 336
OPTIONAL_ATTRIBS (folia.main.StatementLayer attribute), 614
OPTIONAL_ATTRIBS (folia.main.Sentiment attribute), 741
OPTIONAL_ATTRIBS (folia.main.StatementLayer attribute), 627
OPTIONAL_ATTRIBS (folia.main.StatementLayer attribute), 753
OPTIONAL_ATTRIBS (folia.main.SubjectivityAnnotation attribute), 500
OPTIONAL_ATTRIBS (folia.main.SyntacticUnit attribute), 639
OPTIONAL_ATTRIBS (folia.main.SyntaxLayer attribute), 766
originaltext() (olia.mainWhitespace method), 413
originaltext() (olia.mainWord method), 429

Paragraph (Class in folia.main), 263
paragraph() (olia.mainSentence method), 343
paragraph() (olia.mainWord method), 429
paragraphs() (olia.mainAbstractStructureElement method), 37
paragraphs() (olia.mainCell method), 91
paragraphs() (olia.mainDefinition method), 105
paragraphs() (olia.mainDivision method), 119
paragraphs() (olia.mainDocument method), 12
paragraphs() (olia.mainEntry method), 133
paragraphs() (olia.mainEvent method), 147
paragraphs() (olia.mainExample method), 161
paragraphs() (olia.mainFigure method), 175
paragraphs() (olia.mainHead method), 201
paragraphs() (olia.mainLinebreak method), 215
paragraphs() (olia.mainList method), 229
paragraphs() (olia.mainListItem method), 243
paragraphs() (olia.mainNote method), 257
paragraphs() (olia.mainParagraph method), 271
paragraphs() (olia.mainPart method), 285
paragraphs() (olia.mainQuote method), 299
paragraphs() (olia.mainReference method), 313
paragraphs() (olia.mainRow method), 327
paragraphs() (olia.mainSentence method), 343
paragraphs() (olia.mainTable method), 357
paragraphs() (olia.mainTableHead method), 385
paragraphs() (olia.mainTerm method), 371
paragraphs() (olia.mainText method), 399
paragraphs() (olia.mainWhitespace method), 413
paragraphs() (olia.mainWord method), 429
parse() (olia.fqlQuery method), 850
parsecommonarguments() (olia.mainAbstractAnnotationLayer method), 63
parsecommonarguments() (olia.mainAbstractElement method), 23
parsecommonarguments() (olia.mainAbstractSpanAnnotation method), 50
parsecommonarguments() (olia.mainAbstractStructureElement method), 37
parsecommonarguments() (olia.mainAbstractTextMarkup method), 74
parsecommonarguments() (olia.mainCell method), 91
parsecommonarguments() (olia.mainChunk method), 541
parsecommonarguments() (olia.mainChunkingLayer method), 683
parsecommonarguments() (olia.mainCoreferenceChain method), 554
parsecommonarguments() (olia.mainCoreferenceLayer method), 695
parsecommonarguments() (olia.mainCoreferenceLink method), 808
parsecommonarguments() (olia.mainDefinition method), 105
parsecommonarguments() (olia.mainDependenciesLayer method), 708
parsecommonarguments() (olia.mainDependency method), 567
parsecommonarguments() (olia.mainDependencyDependent method), 821
parsecommonarguments() (olia.mainDivision method), 119
parsecommonarguments() (olia.mainDomainAnnotation method), 444
parsecommonarguments() (olia.mainEntitiesLayer method), 720
parsecommonarguments() (olia.mainEntity method), 580
parsecommonarguments() (olia.mainEntry method), 133
parsecommonarguments() (olia.mainEvent method), 147
parsecommonarguments() (olia.mainExample method), 161
parsecommonarguments() (olia.mainFigure method), 175
parsecommonarguments() (olia.mainGap method), 188
parsecommonarguments() (olia.mainHead method), 201
parsecommonarguments() (olia.mainHeadspan method), 834
parsecommonarguments() (olia.mainLangAnnotation method), 468
parsecommonarguments() (olia.mainLemmaAnnotation method), 480
parsecommonarguments() (olia.mainLinebreak method), 215
parsecommonarguments() (olia.mainList method), 229
parsecommonarguments() (olia.mainListItem method), 243
parsecommonarguments() (olia.mainNote method), 257
parsecommonarguments() (olia.mainObservation method), 593
phon() (folia.main.Statement method), 631
phon() (folia.main.StatementLayer method), 758
phon() (folia.main.SubjectivityAnnotation method), 504
phon() (folia.main.SyntacticUnit method), 644
phon() (folia.main.SyntaxLayer method), 770
phon() (folia.main.Table method), 357
phon() (folia.main.TableHead method), 385
phon() (folia.main.Term method), 371
phon() (folia.main.Text method), 399
phon() (folia.main.TextContent method), 516
phon() (folia.main.TimeSegment method), 669
phon() (folia.main.TimingLayer method), 795
phon() (folia.mainWhitespace method), 413
phon() (folia.main.Word method), 430
PHONCONTAINER (folia.main.AbstractAnnotationLayer attribute), 58
PHONCONTAINER (folia.main.AbstractElement attribute), 19
PHONCONTAINER (folia.main.AbstractSpanAnnotation attribute), 45
PHONCONTAINER (folia.main.AbstractStructureElement attribute), 31
PHONCONTAINER (folia.main.TextViewMarkup attribute), 71
PHONCONTAINER (folia.main.Cell attribute), 86
PHONCONTAINER (folia.main.Chunk attribute), 537
PHONCONTAINER (folia.main.ChunkingLayer attribute), 678
PHONCONTAINER (folia.main.CoreferenceChain attribute), 550
PHONCONTAINER (folia.main.CoreferenceLayer attribute), 691
PHONCONTAINER (folia.main.CoreferenceLink attribute), 804
PHONCONTAINER (folia.main.Definition attribute), 100
PHONCONTAINER (folia.main.DependenciesLayer attribute), 703
PHONCONTAINER (folia.main.Dependency attribute), 563
PHONCONTAINER (folia.main.DependencyDependent attribute), 816
PHONCONTAINER (folia.main.Division attribute), 114
PHONCONTAINER (folia.main.DomainAnnotation attribute), 440
PHONCONTAINER (folia.main.EntitiesLayer attribute), 716
PHONCONTAINER (folia.main.Entity attribute), 576
PHONCONTAINER (folia.main.Entry attribute), 128
PHONCONTAINER (folia.main.Event attribute), 142
PHONCONTAINER (folia.main.Example attribute), 156
PHONCONTAINER (folia.main.Figure attribute), 170
PHONCONTAINER (folia.main.Gap attribute), 184
PHONCONTAINER (folia.main.Head attribute), 196
PHONCONTAINER (folia.main.Headspan attribute), 829
PHONCONTAINER (folia.main.LangAnnotation attribute), 464
PHONCONTAINER (folia.main.LemmaAnnotation attribute), 476
PHONCONTAINER (folia.main.Linebreak attribute), 210
PHONCONTAINER (folia.main.List attribute), 224
PHONCONTAINER (folia.main.ListItem attribute), 238
PHONCONTAINER (folia.main.Note attribute), 252
PHONCONTAINER (folia.main.Observation attribute), 588
PHONCONTAINER (folia.main.ObservationLayer attribute), 728
PHONCONTAINER (folia.main.Paragraph attribute), 266
PHONCONTAINER (folia.main.Part attribute), 280
PHONCONTAINER (folia.main.PosAnnotation attribute), 524
PHONCONTAINER (folia.main.PosContent attribute), 452
PHONCONTAINER (folia.main.Predicate attribute), 601
PHONCONTAINER (folia.main.Quote attribute), 294
PHONCONTAINER (folia.main.Reference attribute), 308
PHONCONTAINER (folia.main.Row attribute), 322
PHONCONTAINER (folia.main.SemanticRole attribute), 652
PHONCONTAINER (folia.main.SemanticRolesLayer attribute), 778
PHONCONTAINER (folia.main.SenseAnnotation attribute), 488
PHONCONTAINER (folia.main.Sentence attribute), 336
PHONCONTAINER (folia.main.Sentiment attribute), 614
PHONCONTAINER (folia.main.SentimentLayer attribute), 741
PHONCONTAINER (folia.main.Statement attribute), 627
PHONCONTAINER (folia.main.StatementLayer attribute), 753
PHONCONTAINER (folia.main.SubjectivityAnnotation attribute), 500
PHONCONTAINER (folia.main.SyntacticUnit attribute), 639
PHONCONTAINER (folia.main.SyntaxLayer attribute), 766
PHONCONTAINER (folia.main.Table attribute), 352
PHONCONTAINER (folia.main.TableHead attribute), 380
PHONCONTAINER (folia.main.Term attribute), 366
PHONCONTAINER (folia.main.Text attribute), 394
PHONCONTAINER (folia.main.TextContent attribute), 512
PHONCONTAINER (folia.main.TimeSegment attribute), 665
PHONCONTAINER (folia.main.TimingLayer attribute), 791
PHONCONTAINER (folia.mainWhitespace attribute),
<table>
<thead>
<tr>
<th>Function</th>
<th>Module Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>postappend()</td>
<td>folia.main.AbstractTextMarkup method, 76</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Cell method, 93</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Chunk method, 543</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.ChunkingLayer method, 684</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.CoreferenceChain method, 555</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.CoreferenceLayer method, 696</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.CoreferenceLink method, 809</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Definition method, 107</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.DependenciesLayer method, 709</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Dependency method, 568</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.DependencyDependent method, 822</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Division method, 121</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.DomainAnnotation method, 445</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.EntitiesLayer method, 721</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Entitiy method, 581</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Entry method, 135</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Event method, 149</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Example method, 163</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Figure method, 177</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Gap method, 189</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Head method, 203</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Headspace method, 835</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.LangAnnotation method, 469</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.LemmaAnnotation method, 481</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Linebreak method, 217</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.List method, 231</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.ListItem method, 245</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Note method, 259</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Observation method, 594</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.ObservationLayer method, 734</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Paragraph method, 273</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Part method, 287</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.PhraseContent method, 528</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.PosAnnotation method, 457</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Predicate method, 607</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Quote method, 301</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Reference method, 315</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Row method, 329</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SemanticRole method, 658</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SemanticRolesLayer method, 784</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SenseAnnotation method, 493</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Sentence method, 344</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Sentiment method, 619</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SentimentLayer method, 746</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Statement method, 632</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.StatementLayer method, 759</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SubjectivityAnnotation method, 505</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SyntacticUnit method, 645</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.SyntacticLayer method, 771</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Table method, 358</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.TableHead method, 386</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Term method, 372</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Text method, 400</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.TextContent method, 517</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.TimeSegment method, 670</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.TimingLayer method, 796</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Whitespace method, 414</td>
</tr>
<tr>
<td>postappend()</td>
<td>folia.main.Word method, 431</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractAnnotationLayer method, 64</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractElement method, 24</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractSpanAnnotation method, 51</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractStructureElement method, 38</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractTextMarkup method, 76</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Cell method, 93</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Chunk method, 543</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.ChunkingLayer method, 684</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.CoreferenceLayer method, 556</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.CoreferenceLayer method, 696</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.CoreferenceLink method, 809</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Definition method, 107</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.DependenciesLayer method, 709</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Dependency method, 822</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.DependencyDependent method, 121</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.DomainAnnotation method, 445</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.LangAnnotation method, 469</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.SenseAnnotation method, 493</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Sentence method, 344</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Sentiment method, 619</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.SentimentLayer method, 746</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Statement method, 632</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.StatementLayer method, 759</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.SubjectivityAnnotation method, 505</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.SyntacticUnit method, 645</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.SyntacticLayer method, 771</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Table method, 358</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.TableHead method, 386</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Term method, 372</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Text method, 400</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.TextContent method, 517</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.TimeSegment method, 670</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.TimingLayer method, 796</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Whitespace method, 414</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.Word method, 431</td>
</tr>
<tr>
<td>precedes()</td>
<td>folia.main.AbstractAnnotationLayer method, 64</td>
</tr>
</tbody>
</table>
precedes() (folia.main.Gap method), 189
precedes() (folia.main.Head method), 203
precedes() (folia.main.Headspan method), 835
precedes() (folia.main.LangAnnotation method), 469
precedes() (folia.main.LemmaAnnotation method), 481
precedes() (folia.main.Linebreak method), 217
precedes() (folia.main.List method), 231
precedes() (folia.main.ListItem method), 245
precedes() (folia.main.Note method), 259
precedes() (folia.main.Observation method), 594
precedes() (folia.main.ObservationLayer method), 734
precedes() (folia.main.Paragraph method), 273
precedes() (folia.main.Part method), 287
precedes() (folia.main.PosAnnotation method), 457
precedes() (folia.main.Predicate method), 607
precedes() (folia.main.Referential method), 301
precedes() (folia.main.Statement method), 315
precedes() (folia.main.Row method), 329
precedes() (folia.main.SemanticRole method), 658
precedes() (folia.main.SemanticRolesLayer method), 784
precedes() (folia.main.SenseAnnotation method), 493
precedes() (folia.main.Sentence method), 344
precedes() (folia.main.Sentiment method), 620
precedes() (folia.main.SentimentLayer method), 747
precedes() (folia.main.Statement method), 632
precedes() (folia.main.StatementLayer method), 759
precedes() (folia.main.SubjectivityAnnotation method), 505
precedes() (folia.main.SyntacticUnit method), 645
precedes() (folia.main.SyntaxLayer method), 772
precedes() (folia.main.Table method), 358
precedes() (folia.main.TableHead method), 386
precedes() (folia.main.Term method), 372
precedes() (folia.main.Text method), 400
precedes() (folia.main.TextContent method), 517
precedes() (folia.main.TimeSegment method), 671
precedes() (folia.main.TimingLayer method), 797
precedes() (folia.mainWhitespace method), 414
precedes() (folia.main.Word method), 431
Predicate class in folia.main, 598
previous() (folia.main.AbstractAnnotationLayer method), 64
previous() (folia.main.AbstractAnnotationLayer method), 64
previous() (folia.main.AbstractElement method), 24
previous() (folia.main.AbstractSpanAnnotation method), 51
previous() (folia.main.AbstractStructureElement method), 38
previous() (folia.main.AbstractTextMarkup method), 76
previous() (folia.main.Cell method), 93
previous() (folia.main.Chunk method), 543
previous() (folia.main.ChunkingLayer method), 684
previous() (folia.main.CoreferenceChain method), 556
previous() (folia.main.CoreferenceLayer method), 697
previous() (folia.main.CoreferenceLink method), 809
previous() (folia.main.Definition method), 107
previous() (folia.main.DependenciesLayer method), 709
previous() (folia.main.Dependency method), 569
previous() (folia.main.DependencyDependent method), 822
previous() (folia.main.Division method), 121
previous() (folia.main.DomainAnnotation method), 445
previous() (folia.main.EntitiesLayer method), 722
previous() (folia.main.Entity method), 581
previous() (folia.main.Entry method), 135
previous() (folia.main.Event method), 149
previous() (folia.main.Example method), 163
previous() (folia.main.Figure method), 177
previous() (folia.main.Gap method), 189
previous() (folia.main.Head method), 203
previous() (folia.main.Headspace method), 835
previous() (folia.main.LangAnnotation method), 469
previous() (folia.main.LemmaAnnotation method), 481
previous() (folia.main.Linebreak method), 217
previous() (folia.main.List method), 231
previous() (folia.main.ListItem method), 245
previous() (folia.main.Note method), 259
previous() (folia.main.Observation method), 594
previous() (folia.main.ObservationLayer method), 734
previous() (folia.main.Paragraph method), 273
previous() (folia.main.Part method), 287
previous() (folia.main.PosAnnotation method), 529
previous() (folia.main.Predicate method), 543
previous() (folia.main.Predicate method), 543
previous() (folia.main.Phrase method), 607
previous() (folia.main.Referential method), 301
previous() (folia.main.Statement method), 315
previous() (folia.main.Row method), 329
previous() (folia.main.SemanticRole method), 658
previous() (folia.main.SemanticRolesLayer method), 784
previous() (folia.main.SenseAnnotation method), 493
previous() (folia.main.Sentence method), 344
previous() (folia.main.Sentiment method), 620
previous() (folia.main.SentimentLayer method), 747
previous() (folia.main.Statement method), 632
previous() (folia.main.StatementLayer method), 759
previous() (folia.main.SubjectivityAnnotation method), 505
previous() (folia.main.SyntacticUnit method), 645
previous() (folia.main.SyntaxLayer method), 772
previous() (folia.main.Table method), 358
previous() (folia.main.TableHead method), 386
previous() (folia.main.Term method), 372
previous() (folia.main.Text method), 400
previous() (folia.main.TextContent method), 517
previous() (folia.main.TimeSegment method), 671
previous() (folia.main.TimingLayer method), 797
previous() (folia.mainWhitespace method), 414
previous() (folia.main.Word method), 431
Predicate class in folia.main, 598
previous() (folia.main.AbstractAnnotationLayer method), 64
previous() (folia.main.AbstractElement method), 24
previous() (folia.main.AbstractSpanAnnotation method), 51
previous() (folia.main.AbstractStructureElement method), 38
previous() (folia.main.AbstractTextMarkup method), 76
previous() (folia.main.SubjectivityAnnotation method), 505
previous() (folia.main.SyntacticUnit method), 645
previous() (folia.main.SyntaxLayer method), 772
previous() (folia.main.Table method), 358
previous() (folia.main.TableHead method), 386
previous() (folia.main.Term method), 372
previous() (folia.main.Text method), 400
previous() (folia.main.TextContent method), 518
previous() (folia.main.TimeSegment method), 671
previous() (folia.main.TimingLayer method), 797
previous() (folia.main.Whitespace method), 414
previous() (folia.main.Word method), 431
PRIMEYELEMENT (folia.main.AbstractAnnotationLayer attribute), 58
PRIMEYELEMENT (folia.main.AbstractElement attribute), 19
PRIMEYELEMENT (folia.main.AbstractSpanAnnotation attribute), 45
PRIMEYELEMENT (folia.main.AbstractStructureElement attribute), 31
PRIMEYELEMENT (folia.main.AbstractTextMarkup attribute), 71
PRIMEYELEMENT (folia.main.Cell attribute), 86
PRIMEYELEMENT (folia.main.Chunk attribute), 537
PRIMEYELEMENT (folia.main.ChunkingLayer attribute), 678
PRIMEYELEMENT (folia.main.CoreferenceChain attribute), 550
PRIMEYELEMENT (folia.main.CoreferenceLayer attribute), 691
PRIMEYELEMENT (folia.main.CoreferenceLink attribute), 804
PRIMEYELEMENT (folia.main.Definition attribute), 100
PRIMEYELEMENT (folia.main.DependenciesLayer attribute), 703
PRIMEYELEMENT (folia.main.Dependency attribute), 563
PRIMEYELEMENT (folia.main.DependencyDependent attribute), 816
PRIMEYELEMENT (folia.main.Division attribute), 114
PRIMEYELEMENT (folia.main.DomainAnnotation attribute), 440
PRIMEYELEMENT (folia.main.EntitiesLayer attribute), 716
PRIMEYELEMENT (folia.main.Entity attribute), 576
PRIMEYELEMENT (folia.main.Entry attribute), 128
PRIMEYELEMENT (folia.main.Event attribute), 142
PRIMEYELEMENT (folia.main.Example attribute), 156
PRIMEYELEMENT (folia.main.Figure attribute), 170
PRIMEYELEMENT (folia.main.Gap attribute), 184
PRIMEYELEMENT (folia.main.Head attribute), 196
PRIMEYELEMENT (folia.main.Headspan attribute), 829
PRIMEYELEMENT (folia.main.LangAnnotation attribute), 464
PRIMEYELEMENT (folia.main.LemmaAnnotation attribute), 476
PRIMEYELEMENT (folia.main.Linebreak attribute), 210
PRIMEYELEMENT (folia.main.List attribute), 224
PRIMEYELEMENT (folia.main.ListItem attribute), 238
PRIMEYELEMENT (folia.main.Note attribute), 252
PRIMEYELEMENT (folia.main.Observation attribute), 588
PRIMEYELEMENT (folia.main.ObservationLayer attribute), 728
PRIMEYELEMENT (folia.main.Paragraph attribute), 266
PRIMEYELEMENT (folia.main.Part attribute), 280
PRIMEYELEMENT (folia.main.PhrASE attribute), 524
PRIMEYELEMENT (folia.main.PosAnnotation attribute), 452
PRIMEYELEMENT (folia.main.Predicate attribute), 601
PRIMEYELEMENT (folia.main.Quote attribute), 294
PRIMEYELEMENT (folia.main.Reference attribute), 308
PRIMEYELEMENT (folia.main.Row attribute), 652
PRIMEYELEMENT (folia.main.SemanticRole attribute), 778
PRIMEYELEMENT (folia.main.Sentiment attribute), 336
PRIMEYELEMENT (folia.main.SentimentLayer attribute), 614
PRIMEYELEMENT (folia.main.Statement attribute), 741
PRIMEYELEMENT (folia.main.StatementLayer attribute), 627
PRIMEYELEMENT (folia.main.StatementLayer attribute), 753
PRIMEYELEMENT (folia.main.SubjectivityAnnotation attribute), 500
PRIMEYELEMENT (folia.main.SyntacticUnit attribute), 639
PRIMEYELEMENT (folia.main.SyntaxLayer attribute), 766
PRIMEYELEMENT (folia.main.Table attribute), 352
PRIMEYELEMENT (folia.main.TableHead attribute), 380
PRIMARYELEMENT  (folia.main.Term attribute), 366
PRIMARYELEMENT  (folia.main.Text attribute), 394
PRIMARYELEMENT  (folia.main.TextContent attribute), 512
PRIMARYELEMENT  (folia.main.TimeSegment attribute), 665
PRIMARYELEMENT  (folia.main.TimingLayer attribute), 791
PRIMARYELEMENT  (folia.mainWhitespace attribute), 408
PRIMARYELEMENT  (folia.main.Word attribute), 422
PRINTABLE    (folia.main.AbstractAnnotationLayer attribute), 58
PRINTABLE    (folia.main.AbstractElement attribute), 19
PRINTABLE    (folia.main.AbstractSpanAnnotation attribute), 45
PRINTABLE    (folia.main.AbstractStructureElement attribute), 31
PRINTABLE    (folia.main.AbstractTextMarkup attribute), 71
PRINTABLE    (folia.main.Cell attribute), 86
PRINTABLE    (folia.main.Chunk attribute), 537
PRINTABLE    (folia.main.ChunkingLayer attribute), 678
PRINTABLE    (folia.main.CoreferenceChain attribute), 550
PRINTABLE    (folia.main.CoreferenceLayer attribute), 691
PRINTABLE    (folia.main.CoreferenceLink attribute), 804
PRINTABLE    (folia.main.Definition attribute), 100
PRINTABLE    (folia.main.DependenciesLayer attribute), 703
PRINTABLE    (folia.main.Dependency attribute), 563
PRINTABLE    (folia.main.DependencyDependent attribute), 816
PRINTABLE    (folia.main.Division attribute), 114
PRINTABLE    (folia.main.DomainAnnotation attribute), 440
PRINTABLE    (folia.main.EntitiesLayer attribute), 716
PRINTABLE    (folia.main.Entity attribute), 576
PRINTABLE    (folia.main.Entry attribute), 128
PRINTABLE    (folia.main.Event attribute), 142
PRINTABLE    (folia.main.Example attribute), 156
PRINTABLE    (folia.main.Figure attribute), 170
PRINTABLE    (folia.main.Gap attribute), 184
PRINTABLE    (folia.main.Head attribute), 196
PRINTABLE    (folia.main.Headspan attribute), 829
PRINTABLE    (folia.main.LangAnnotation attribute), 464
PRINTABLE    (folia.main.LemmaAnnotation attribute), 476
PRINTABLE    (folia.main.Linebreak attribute), 210
PRINTABLE    (folia.main.List attribute), 224
PRINTABLE    (folia.main.ListItem attribute), 238
PRINTABLE    (folia.main.Note attribute), 252
PRINTABLE    (folia.main.Observation attribute), 588
PRINTABLE    (folia.main.ObservationLayer attribute), 728
PRINTABLE    (folia.main.Paragraph attribute), 266
PRINTABLE    (folia.main.Part attribute), 280
PRINTABLE    (folia.main.PhonContent attribute), 524
PRINTABLE    (folia.main.PosAnnotation attribute), 452
PRINTABLE    (folia.main.Predicate attribute), 601
PRINTABLE    (folia.mainQuote attribute), 294
PRINTABLE    (folia.main.Reference attribute), 308
PRINTABLE    (folia.main.Row attribute), 322
PRINTABLE    (folia.main.SemanticRole attribute), 652
PRINTABLE    (folia.main.SemanticRolesLayer attribute), 778
PRINTABLE    (folia.main.SenseAnnotation attribute), 488
PRINTABLE    (folia.mainSentence attribute), 336
PRINTABLE    (folia.mainSentiment attribute), 614
PRINTABLE    (folia.mainSentimentLayer attribute), 741
PRINTABLE    (folia.main.Statement attribute), 627
PRINTABLE    (folia.main.StatementLayer attribute), 753
PRINTABLE    (folia.mainSubjectivityAnnotation attribute), 500
PRINTABLE    (folia.main.SyntacticUnit attribute), 639
PRINTABLE    (folia.main.SyntaxLayer attribute), 766
PRINTABLE    (folia.main.Table attribute), 352
PRINTABLE    (folia.main.TableHead attribute), 380
PRINTABLE    (folia.main.Term attribute), 366
PRINTABLE    (folia.main.Text attribute), 394
PRINTABLE    (folia.main.TextContent attribute), 512
PRINTABLE    (folia.main.TimeSegment attribute), 665
PRINTABLE    (folia.main.TimingLayer attribute), 791
PRINTABLE    (folia.mainWhitespace attribute), 408
PRINTABLE    (folia.main.Word attribute), 422
publisher()    (folia.main.Document method), 13

Q
Query  (class in folia.fql), 848
Quote    (class in folia.main), 291

R
Reference    (class in folia.main), 305
relaxng()    (folia.main.AbstractAnnotationLayer class method), 64
relaxng()    (folia.main.AbstractElement class method), 25
relaxng()    (folia.main.AbstractSpanAnnotation class method), 51
relaxng()    (folia.main.AbstractStructureElement class method), 38
relaxng()    (folia.main.AbstractTextMarkup class method), 76
relaxng()    (folia.main.Cell class method), 93
relaxng()    (folia.main.Chunk class method), 543
relaxng()    (folia.main.ChunkingLayer class method), 684
relaxng() (folia.main.CoreferenceChain class method), 556
relaxng() (folia.main.CoreferenceLayer class method), 697
relaxng() (folia.main.CoreferenceLink class method), 810
relaxng() (folia.main.Definition class method), 107
relaxng() (folia.main.DependenciesLayer class method), 709
relaxng() (folia.main.Dependency class method), 569
relaxng() (folia.main.DependencyDependent class method), 822
relaxng() (folia.main.Division class method), 121
relaxng() (folia.main.DomainAnnotation class method), 445
relaxng() (folia.main.EntitiesLayer class method), 722
relaxng() (folia.main.Entity class method), 582
relaxng() (folia.main.Entry class method), 135
relaxng() (folia.main.Event class method), 149
relaxng() (folia.main.Example class method), 163
relaxng() (folia.main.Figure class method), 177
relaxng() (folia.main.Gap class method), 189
relaxng() (folia.main.Head class method), 203
relaxng() (folia.main.Headspan class method), 835
relaxng() (folia.main.LangAnnotation class method), 469
relaxng() (folia.main.LemmaAnnotation class method), 481
relaxng() (folia.main.Linebreak class method), 217
relaxng() (folia.main.List class method), 231
relaxng() (folia.main.ListItem class method), 245
relaxng() (folia.main.Note class method), 259
relaxng() (folia.main.Observation class method), 594
relaxng() (folia.main.ObservationLayer class method), 734
relaxng() (folia.main.Paragraph class method), 273
relaxng() (folia.main.Part class method), 287
relaxng() (folia.main.Phrase class method), 529
relaxng() (folia.main.PosAnnotation class method), 457
relaxng() (folia.main.Predicate class method), 607
relaxng() (folia.main.Punctuation class method), 301
relaxng() (folia.main.Reference class method), 315
relaxng() (folia.main.Row class method), 329
relaxng() (folia.main.SemanticRole class method), 658
relaxng() (folia.main.SemanticRolesLayer class method), 784
relaxng() (folia.main.SenseAnnotation class method), 493
relaxng() (folia.mainSentence class method), 345
relaxng() (folia.main.Sentiment class method), 620
relaxng() (folia.main.SentenceLayer class method), 747
relaxng() (folia.main.Statement class method), 633
relaxng() (folia.main.StatementLayer class method), 759
relaxng() (folia.main.SubjectivityAnnotation class method), 505
relaxng() (folia.main.SyntaxLayer class method), 772
relaxng() (folia.main.Table class method), 359
relaxng() (folia.main.TableHead class method), 387
relaxng() (folia.main.Term class method), 373
relaxng() (folia.main.Text class method), 401
relaxng() (folia.main.TextContent class method), 518
relaxng() (folia.main.TimeSegment class method), 671
relaxng() (folia.main.TimingLayer class method), 797
relaxng() (folia.main.Whitespace class method), 415
relaxng() (folia.main.Word class method), 431
relaxng_backwards() (folia.main.AbstractAnnotationLayer class method), 64
relaxng_backwards() (folia.main.AbstractElement class method), 25
relaxng_backwards() (folia.main.AbstractSpanAnnotation class method), 51
relaxng_backwards() (folia.main.AbstractStructureElement class method), 38
relaxng_backwards() (folia.main.AbstractTextMarkup class method), 76
relaxng_backwards() (folia.main.Cell class method), 93
relaxng_backwards() (folia.main.Chunk class method), 543
relaxng_backwards() (folia.main.ChunkingLayer class method), 684
relaxng_backwards() (folia.main.CoreferenceChain class method), 556
relaxng_backwards() (folia.main.CoreferenceLayer class method), 697
relaxng_backwards() (folia.main.CoreferenceLink class method), 810
relaxng_backwards() (folia.main.Definition class method), 107
relaxng_backwards() (folia.main.DependenciesLayer class method), 709
relaxng_backwards() (folia.main.Dependency class method), 569
relaxng_backwards() (folia.main.DependencyDependent class method), 822
relaxng_backwards() (folia.main.Division class method), 121
relaxng_backwards() (folia.main.DomainAnnotation class method), 445
relaxng_backwards() (folia.main.EntitiesLayer class method), 722
relaxng_backwards() (folia.main.Entity class method), 582
relaxng_backwards() (folia.main.Entry class method), 135
relaxng_backwards() (folia.main.Event class method), 149
relaxng_backwards() (folia.main.Example class method), 163
relaxng_backwards() (folia.main.Figure class method), 177
relaxng_backwards() (folia.main.Gap class method), 189
relaxng_backwards() (folia.main.Head class method), 203
relaxng_backwards() (folia.main.Headspan class method), 835
relaxng_backwards() (folia.main.LangAnnotation class method), 469
relaxng_backwards() (folia.main.LemmaAnnotation class method), 481
relaxng_backwards() (folia.main.Linebreak class method), 217
relaxng_backwards() (folia.main.List class method), 231
relaxng_backwards() (folia.main.ListItem class method), 245
relaxng_backwards() (folia.main.Note class method), 259
relaxng_backwards() (folia.main.Observation class method), 594
relaxng_backwards() (folia.main.ObservationLayer class method), 734
relaxng_backwards() (folia.main.Paragraph class method), 273
relaxng_backwards() (folia.main.Part class method), 287
relaxng_backwards() (folia.main.Phrase class method), 529
relaxng_backwards() (folia.main.PosAnnotation class method), 457
relaxng_backwards() (folia.main.Predicate class method), 607
relaxng_backwards() (folia.main.Quote class method), 301
relaxng_backwards() (folia.main.Reference class method), 315
relaxng_backwards() (folia.main.Row class method), 329
relaxng_backwards() (folia.main.SemanticRole class method), 658
relaxng_backwards() (folia.main.SemanticRolesLayer class method), 784
relaxng_backwards() (folia.main.SenseAnnotation class method), 493
relaxng_backwards() (folia.main.Sentence class method), 345
relaxng_backwards() (folia.main.Sentiment class method), 620
relaxng_backwards() (folia.main.SentimentLayer class method), 747
relaxng_backwards() (folia.main.Statement class method), 633
relaxng_backwards() (folia.main.StatementLayer class method), 759
relaxng_backwards() (folia.main.SubjectivityAnnotation class method), 505
relaxng_backwards() (folia.main.SyntacticUnit class method), 645
relaxng_backwards() (folia.main.SyntaxLayer class method), 772
relaxng_backwards() (folia.main.Table class method), 359
relaxng_backwards() (folia.main.TableHead class method), 387
relaxng_backwards() (folia.main.Term class method), 373
relaxng_backwards() (folia.main.Text class method), 401
relaxng_backwards() (folia.main.TextContent class method), 518
relaxng_backwards() (folia.main.TimeSegment class method), 671
relaxng_backwards() (folia.main.TimingLayer class method), 797
relaxng_backwards() (folia.main.Whitespace class method), 415
relaxng_backwards() (folia.main.Word class method), 431
remove() (folia.main.AbstractAnnotationLayer method), 64
replace() (folia.main.Paragraph method), 273
replace() (folia.main.Part method), 287
replace() (folia.main.PhraseContent method), 529
replace() (folia.main.Predicate method), 458
replace() (folia.main.Predicate method), 607
replace() (folia.main.Quote method), 301
replace() (folia.main.Row method), 329
replace() (folia.main.SemanticRole method), 658
replace() (folia.main.SemanticRolesLayer method), 784
replace() (folia.main.Sentence method), 494
replace() (folia.main.Sentence method), 345
replace() (folia.main.Sentiment method), 620
replace() (folia.main.SentimentLayer method), 747
replace() (folia.main.Statement method), 633
replace() (folia.main.StatementLayer method), 759
replace() (folia.main.SubjectivityAnnotation method), 506
replace() (folia.main.SyntheticUnit method), 646
replace() (folia.main.SyntaxLayer method), 772
replace() (folia.main.Table method), 359
replace() (folia.main.TableHead method), 387
replace() (folia.main.Term method), 373
replace() (folia.main.Text method), 401
replace() (folia.main.TextContent method), 518
replace() (folia.main.TimeSegment method), 671
replace() (folia.main.TimingLayer method), 797
replace() (folia.main.Whitespace method), 415
replace() (folia.main.Word method), 431
REQUIRED_ATTRIBS (folia.main.AbstractAnnotationLayer attribute), 58
REQUIRED_ATTRIBS (folia.main.AbstractElement attribute), 19
REQUIRED_ATTRIBS (folia.main.AbstractSpanAnnotation attribute), 45
REQUIRED_ATTRIBS (folia.main.AbstractStructureElement attribute), 31
REQUIRED_ATTRIBS (folia.main.AbstractTextMarkup attribute), 71
REQUIRED_ATTRIBS (folia.main.Cell attribute), 86
REQUIRED_ATTRIBS (folia.main.Chunk attribute), 537
REQUIRED_ATTRIBS (folia.main.ChunkingLayer attribute), 678
REQUIRED_ATTRIBS (folia.main.CoreferenceChain attribute), 550
REQUIRED_ATTRIBS (folia.main.CoreferenceLayer attribute), 691
REQUIRED_ATTRIBS (folia.main.CoreferenceLink attribute), 804
REQUIRED_ATTRIBS (folia.main.Definition attribute), 100
REQUIRED_ATTRIBS (folia.main.DependenciesLayer attribute), 703
REQUIRED_ATTRIBS (folia.main.Dependency attribute), 563
REQUIRED_ATTRIBS (folia.main.DependencyDependent attribute), 816
REQUIRED_ATTRIBS (folia.main.Division attribute), 114
REQUIRED_ATTRIBS (folia.main.DomainAnnotation attribute), 440
REQUIRED_ATTRIBS (folia.main.EntitiesLayer attribute), 716
REQUIRED_ATTRIBS (folia.main.Entity attribute), 576
REQUIRED_ATTRIBS (folia.main.Event attribute), 128
REQUIRED_ATTRIBS (folia.main.Example attribute), 156
REQUIRED_ATTRIBS (folia.main.Figure attribute), 170
REQUIRED_ATTRIBS (folia.main.Gap attribute), 184
REQUIRED_ATTRIBS (folia.main.Head attribute), 196
REQUIRED_ATTRIBS (folia.main.Headspan attribute), 829
REQUIRED_ATTRIBS (folia.main.LangAnnotation attribute), 464
REQUIRED_ATTRIBS (folia.main.LemmaAnnotation attribute), 476
REQUIRED_ATTRIBS (folia.main.Linebreak attribute), 210
REQUIRED_ATTRIBS (folia.main.List attribute), 224
REQUIRED_ATTRIBS (folia.main.ListItem attribute), 238
REQUIRED_ATTRIBS (folia.main.Note attribute), 252
REQUIRED_ATTRIBS (folia.main.Observation attribute), 588
REQUIRED_ATTRIBS (folia.main.ObservationLayer attribute), 728
REQUIRED_ATTRIBS (folia.main.Paragraph attribute), 266
REQUIRED_ATTRIBS (folia.main.Part attribute), 280
REQUIRED_ATTRIBS (folia.main.PhraseContent attribute), 524
REQUIRED_ATTRIBS (folia.main.PosAnnotation attribute), 452
REQUIRED_ATTRIBS (folia.main.Predicate attribute), 601
REQUIRED_ATTRIBS (folia.main.Quote attribute), 294
REQUIRED_ATTRIBS (folia.main.Reference attribute), 308
REQUIRED_ATTRIBS (folia.main.Row attribute), 322
REQUIRED_ATTRIBS (folia.main.SemanticRole at-
tribute), 652
REQUIRED_ATTRIBS (folia.main.SemanticRolesLayer attribute), 778
REQUIRED_ATTRIBS (folia.main.SenseAnnotation attribute), 488
REQUIRED_ATTRIBS (folia.main.Sentence attribute), 336
REQUIRED_ATTRIBS (folia.main.Sentiment attribute), 614
REQUIRED_ATTRIBS (folia.main.SentimentLayer attribute), 741
REQUIRED_ATTRIBS (folia.main.Statement attribute), 627
REQUIRED_ATTRIBS (folia.main.StatementLayer attribute), 753
REQUIRED_ATTRIBS (folia.main.SubjectivityAnnotation attribute), 500
REQUIRED_ATTRIBS (folia.main.SyntacticUnit attribute), 639
REQUIRED_ATTRIBS (folia.main.SyntaxLayer attribute), 766
REQUIRED_ATTRIBS (folia.main.Table attribute), 352
REQUIRED_ATTRIBS (folia.main.TableHead attribute), 380
REQUIRED_ATTRIBS (folia.main.Term attribute), 366
REQUIRED_ATTRIBS (folia.main.Text attribute), 394
REQUIRED_ATTRIBS (folia.main.TextContent attribute), 512
REQUIRED_ATTRIBS (folia.main.TimeSegment attribute), 665
REQUIRED_ATTRIBS (folia.main.TimingLayer attribute), 791
REQUIRED_ATTRIBS (folia.main.Whitespace attribute), 408
REQUIRED_ATTRIBS (folia.main.Word attribute), 422
REQUIRED_DATA (folia.main.AbstractAnnotationLayer attribute), 58
REQUIRED_DATA (folia.main.AbstractElement attribute), 19
REQUIRED_DATA (folia.main.AbstractSpanAnnotation attribute), 45
REQUIRED_DATA (folia.main.AbstractStructureElement attribute), 31
REQUIRED_DATA (folia.main.AbstractTextMarkup attribute), 71
REQUIRED_DATA (folia.main.Cell attribute), 86
REQUIRED_DATA (folia.main.Chunk attribute), 537
REQUIRED_DATA (folia.main.ChunkingLayer attribute), 678
REQUIRED_DATA (folia.main.CoreferenceChain attribute), 550
REQUIRED_DATA (folia.main.CoreferenceLayer attribute), 691
REQUIRED_DATA (folia.main.CoreferenceLink attribute), 804
REQUIRED_DATA (folia.main.Definition attribute), 100
REQUIRED_DATA (folia.main.DependenciesLayer attribute), 703
REQUIRED_DATA (folia.main.Dependency attribute), 563
REQUIRED_DATA (folia.main.DependencyDependent attribute), 817
REQUIRED_DATA (folia.main.Division attribute), 114
REQUIRED_DATA (folia.main.DomainAnnotation attribute), 440
REQUIRED_DATA (folia.main.EntitiesLayer attribute), 716
REQUIRED_DATA (folia.main.Entity attribute), 576
REQUIRED_DATA (folia.main.Entry attribute), 128
REQUIRED_DATA (folia.main.Event attribute), 142
REQUIRED_DATA (folia.main.Example attribute), 156
REQUIRED_DATA (folia.main.Figure attribute), 170
REQUIRED_DATA (folia.main.Gap attribute), 184
REQUIRED_DATA (folia.main.Head attribute), 196
REQUIRED_DATA (folia.main.Headspan attribute), 829
REQUIRED_DATA (folia.main.LangSpan attribute), 464
REQUIRED_DATA (folia.main.LemmaAnnotation attribute), 476
REQUIRED_DATA (folia.main.Linebreak attribute), 210
REQUIRED_DATA (folia.main.List attribute), 224
REQUIRED_DATA (folia.main.ListItem attribute), 238
REQUIRED_DATA (folia.main.Note attribute), 252
REQUIRED_DATA (folia.main.Observation attribute), 589
REQUIRED_DATA (folia.main.ObservationLayer attribute), 728
REQUIRED_DATA (folia.main.Paragraph attribute), 266
REQUIRED_DATA (folia.main.Part attribute), 280
REQUIRED_DATA (folia.main.PhonContent attribute), 524
REQUIRED_DATA (folia.main.PosAnnotation attribute), 452
REQUIRED_DATA (folia.main.Predicate attribute), 601
REQUIRED_DATA (folia.main.Quote attribute), 294
REQUIRED_DATA (folia.main.Reference attribute), 308
REQUIRED_DATA (folia.main.Row attribute), 322
REQUIRED_DATA (folia.main.SemanticRole attribute), 652
REQUIRED_DATA (folia.main.SemanticRolesLayer attribute), 778
REQUIRED_DATA (folia.main.SenseAnnotation attribute), 488
REQUIRED_DATA (folia.main.Sentence attribute), 336
REQUIRED_DATA (folia.main.Sentiment attribute), 614
| REQUIRED_DATA | (folio.main.SentimentLayer attribute) | 710 | resolveoffsets() | (folio.main.Dependency method), 710 |
| REQUIRED_DATA | (folio.main.Statement attribute) | 627 | resolveoffsets() | (folio.main.Dependency method), 569 |
| REQUIRED_DATA | (folio.main.StatementLayer attribute) | 753 | resolveoffsets() | (folio.main.DependencyDependent method), 823 |
| REQUIRED_DATA | (folio.main.SubjectivityAnnotation attribute) | 500 | resolveoffsets() | (folio.main.Division method), 121 |
| REQUIRED_DATA | (folio.main.SyntacticUnit attribute) | 640 | resolveoffsets() | (folio.main.DomainAnnotation method), 446 |
| REQUIRED_DATA | (folio.main.SyntaxLayer attribute) | 766 | resolveoffsets() | (folio.main.EntitiesLayer method), 722 |
| REQUIRED_DATA | (folio.main.Table attribute) | 352 | resolveoffsets() | (folio.main.Entity method), 582 |
| REQUIRED_DATA | (folio.main.TableHead attribute) | 380 | resolveoffsets() | (folio.main.Entry method), 135 |
| REQUIRED_DATA | (folio.main.Term attribute) | 366 | resolveoffsets() | (folio.main.Event method), 149 |
| REQUIRED_DATA | (folio.main.Text attribute) | 394 | resolveoffsets() | (folio.main.Example method), 163 |
| REQUIRED_DATA | (folio.main.TextContent attribute) | 512 | resolveoffsets() | (folio.main.Gap method), 190 |
| REQUIRED_DATA | (folio.main.TimeSegment attribute) | 665 | resolveoffsets() | (folio.main.Head method), 203 |
| REQUIRED_DATA | (folio.main.TimingLayer attribute) | 791 | resolveoffsets() | (folio.main.Headspan method), 836 |
| REQUIRED_DATA | (folio.mainWhitespace attribute) | 408 | resolveoffsets() | (folio.main.LangAnnotation method), 470 |
| REQUIRED_DATA | (folio.main.Word attribute) | 422 | resolveoffsets() | (folio.main.LemmaAnnotation method), 482 |
| resolve() | (folio.main.AbstractTextMarkup method) | 76 | resolveoffsets() | (folio.main.Linebreak method), 217 |
| resolve() | (folio.main.Linebreak method) | 217 | resolveoffsets() | (folio.main.List method), 231 |
| resolve() | (folio.main.Reference method) | 315 | resolveoffsets() | (folio.main.ListItem method), 245 |
| resolveoffsets() | (folio.main.AbstractAnnotationLayer method) | 65 | resolveoffsets() | (folio.main.Note method), 259 |
| resolveoffsets() | (folio.main.AbstractElement method), 25 | resolveoffsets() | (folio.main.Observation method), 595 |
| resolveoffsets() | (folio.main.AbstractSpanAnnotation method) | 52 | resolveoffsets() | (folio.main.ObservationLayer method), 735 |
| resolveoffsets() | (folio.main.AbstractTextMarkup method) | 76 | resolveoffsets() | (folio.main.Phrase method), 287 |
| resolveoffsets() | (folio.main.Cell method) | 93 | resolveoffsets() | (folio.main.PosAnnotation method), 529 |
| resolveoffsets() | (folio.main.Chunk method) | 543 | resolveoffsets() | (folio.main.Predicate method), 458 |
| resolveoffsets() | (folio.main.ChunkingLayer method) | 685 | resolveoffsets() | (folio.main.Predicate method), 607 |
| resolveoffsets() | (folio.main.CoreferenceChain method), 556 | resolveoffsets() | (folio.main.Quote method), 301 |
| resolveoffsets() | (folio.main.CoreferenceLayer method), 697 | resolveoffsets() | (folio.main.Reference method), 315 |
| resolveoffsets() | (folio.main.CoreferenceLink method), 810 | resolveoffsets() | (folio.main.Row method), 329 |
| resolveoffsets() | (folio.main.Definition method), 107 | resolveoffsets() | (folio.main.SemanticRole method), 658 |
| resolveoffsets() | (folio.main.DependenciesLayer method) | 107 | resolveoffsets() | (folio.main.SemanticRolesLayer method), 785 |
| resolveoffsets() | (folio.main.DependenciesLayer method) | 107 | resolveoffsets() | (folio.main.SenseAnnotation method) |
rightcontext() (folia.main.TimeSegment method), 671
rightcontext() (folia.main.TimingLayer method), 797
rightcontext() (folia.main.Whitespace method), 415
rightcontext() (folia.main.Word method), 432
Row (class in folia.main), 319
S
save() (folia.main.Document method), 13
select() (folia.main.AbstractAnnotationLayer method), 65
select() (folia.main.AbstractElement method), 25
select() (folia.main.AbstractSpanAnnotation method), 52
select() (folia.main.AbstractStructureElement method), 39
select() (folia.main.AbstractTextMarkup method), 77
select() (folia.main.Cell method), 94
select() (folia.main.Chunk method), 544
select() (folia.main.ChunkingLayer method), 685
select() (folia.main.CoreferenceChain method), 556
select() (folia.main.CoreferenceLayer method), 697
select() (folia.main.CoreferenceLink method), 810
select() (folia.main.Definition method), 108
select() (folia.main.DependenciesLayer method), 710
select() (folia.main.Dependency method), 569
select() (folia.main.DependencyDependent method), 823
select() (folia.main.Division method), 122
select() (folia.main.Document method), 13
select() (folia.main.DomainAnnotation method), 446
select() (folia.main.EntitiesLayer method), 722
select() (folia.main.Entity method), 582
select() (folia.main.Entry method), 136
select() (folia.main.Event method), 150
select() (folia.main.Example method), 164
select() (folia.main.Figure method), 178
select() (folia.main.Gap method), 190
select() (folia.main.Head method), 204
select() (folia.main.Headers method), 836
select() (folia.main.LangAnnotation method), 470
select() (folia.main.LemmaAnnotation method), 482
select() (folia.main.Linebreak method), 217
select() (folia.main.List method), 232
select() (folia.main.ListItem method), 246
select() (folia.main.Note method), 260
select() (folia.main.Observation method), 595
select() (folia.main.ObservationLayer method), 735
select() (folia.main.Paragraph method), 274
select() (folia.main.Part method), 288
select() (folia.main.PhraseContent method), 529
select() (folia.main.PosAnnotation method), 458
select() (folia.main.Predicate method), 608
select() (folia.main.Phrase method), 302
select() (folia.main.Reference method), 316
select() (folia.main.Row method), 330
select() (folia.main.SemanticRole method), 659
select() (folia.main.SemanticRolesLayer method), 785
select() (folia.main.SenseAnnotation method), 494
select() (folia.main.Sentence method), 345
select() (folia.main.Sentiment method), 620
select() (folia.main.SentimentLayer method), 747
select() (folia.main.Statement method), 633
select() (folia.main.StatementLayer method), 760
select() (folia.main.SubjectivityAnnotation method), 506
select() (folia.main.SyntacticUnit method), 646
select() (folia.main.SyntaxLayer method), 772
select() (folia.main.Table method), 359
select() (folia.main.TableHead method), 387
select() (folia.main.Term method), 373
select() (folia.main.Text method), 401
select() (folia.main.TextContent method), 518
select() (folia.main.Timsegment method), 671
select() (folia.main.TimingLayer method), 797
select() (folia.mainWhitespace method), 415
select() (folia.main.Word method), 432
SemanticRole (class in folia.main), 649
SemanticRolesLayer (class in folia.main), 775
sense() (folia.main.Word method), 432
SenseAnnotation (class in folia.main), 485
Sentence (class in folia.main), 333
sentence() (folia.main.Word method), 432
sentences() (folia.main.AbstractStructureElement method), 39
sentences() (folia.main.Cell method), 94
sentences() (folia.main.Definition method), 108
sentences() (folia.main.Division method), 122
sentences() (folia.main.Document method), 13
sentences() (folia.main.DomainAnnotation method), 446
sentences() (folia.main.EntitiesLayer method), 722
sentences() (folia.main.Entity method), 582
sentences() (folia.main.Entry method), 136
sentences() (folia.main.Event method), 150
sentences() (folia.main.Example method), 164
sentences() (folia.main.Figure method), 178
sentences() (folia.main.Gap method), 190
sentences() (folia.main.Head method), 204
sentences() (folia.main.Headers method), 836
sentences() (folia.main.LangAnnotation method), 470
sentences() (folia.main.LemmaAnnotation method), 482
sentences() (folia.main.Linebreak method), 217
sentences() (folia.main.List method), 232
sentences() (folia.main.ListItem method), 246
sentences() (folia.main.Note method), 260
sentences() (folia.main.Observation method), 595
sentences() (folia.main.ObservationLayer method), 735
sentences() (folia.main.Paragraph method), 274
sentences() (folia.main.Part method), 288
sentences() (folia.main.Quote method), 302
sentences() (folia.main.Reference method), 316
sentences() (folia.main.Row method), 330
sentences() (folia.main.Sentence method), 346
Index
sort() (folia.main.TimeSegment method), 672
sort() (folia.main.TimingLayer method), 798
SPEAKABLE (folia.main.AbstractAnnotationLayer attribute), 58
SPEAKABLE (folia.main.AbstractElement attribute), 19
SPEAKABLE (folia.main.AbstractSpanAnnotation attribute), 45
SPEAKABLE (folia.main.AbstractStructureElement attribute), 31
SPEAKABLE (folia.main.AbstractTextMarkup attribute), 71
SPEAKABLE (folia.main.Cell attribute), 86
SPEAKABLE (folia.main.Chunk attribute), 537
SPEAKABLE (folia.main.ChunkingLayer attribute), 678
SPEAKABLE (folia.main.Definition attribute), 550
SPEAKABLE (folia.main.CoreferenceLayer attribute), 691
SPEAKABLE (folia.main.CoreferenceLink attribute), 804
SPEAKABLE (folia.main.Definition attribute), 100
SPEAKABLE (folia.main.DependenciesLayer attribute), 703
SPEAKABLE (folia.main.Dependency attribute), 563
SPEAKABLE (folia.main.DependencyDependent attribute), 817
SPEAKABLE (folia.main.Division attribute), 114
SPEAKABLE (folia.main.DomainAnnotation attribute), 440
SPEAKABLE (folia.main.EntitiesLayer attribute), 716
SPEAKABLE (folia.main.Entry attribute), 576
SPEAKABLE (folia.main.Entry attribute), 128
SPEAKABLE (folia.main.Event attribute), 142
SPEAKABLE (folia.main.Example attribute), 156
SPEAKABLE (folia.main.Figure attribute), 170
SPEAKABLE (folia.main.Gap attribute), 184
SPEAKABLE (folia.main.Head attribute), 196
SPEAKABLE (folia.main.Headspan attribute), 829
SPEAKABLE (folia.main.LangAnnotation attribute), 464
SPEAKABLE (folia.main.LemmaAnnotation attribute), 476
SPEAKABLE (folia.main.Linebreak attribute), 210
SPEAKABLE (folia.main.List attribute), 224
SPEAKABLE (folia.main.ListItem attribute), 238
SPEAKABLE (folia.main.Note attribute), 252
SPEAKABLE (folia.main.Observation attribute), 589
SPEAKABLE (folia.main.ObservationLayer attribute), 728
SPEAKABLE (folia.main.Paragraph attribute), 266
SPEAKABLE (folia.main.Part attribute), 280
SPEAKABLE (folia.main.PhonContent attribute), 524
SPEAKABLE (folia.main.PosAnnotation attribute), 452
SPEAKABLE (folia.main.Predicate attribute), 601
SPEAKABLE (folia.main.Quote attribute), 294
SPEAKABLE (folia.main.Reference attribute), 308
SPEAKABLE (folia.main.Row attribute), 322
SPEAKABLE (folia.main.SenseAnnotation attribute), 488
SPEAKABLE (folia.main.Sentiment attribute), 614
SPEAKABLE (folia.main.SentimentLayer attribute), 741
SPEAKABLE (folia.main.Statement attribute), 627
SPEAKABLE (folia.main.StatementLayer attribute), 753
SPEAKABLE (folia.main.SubjectivityAnnotation attribute), 500
SPEAKABLE (folia.main.SyntaxLayer attribute), 640
SPEAKABLE (folia.main.SyntaxLayer attribute), 766
SPEAKABLE (folia.main.Table attribute), 352
SPEAKABLE (folia.main.TableHead attribute), 380
SPEAKABLE (folia.main.Text attribute), 394
SPEAKABLE (folia.main.TextContent attribute), 512
SPEAKABLE (folia.main.TimeSegment attribute), 665
SPEAKABLE (folia.main.TimingLayer attribute), 791
SPEAKABLE (folia.mainWhitespace attribute), 408
SPEAKABLE (folia.main.Word attribute), 422
speech_speaker() (folia.main.AbstractAnnotationLayer method), 66
speech_speaker() (folia.main.AbstractElement method), 26
speech_speaker() (folia.main.AbstractSpanAnnotation method), 53
speech_speaker() (folia.main.AbstractStructureElement method), 40
speech_speaker() (folia.main.AbstractTextMarkup method), 77
speech_speaker() (folia.main.Cell method), 95
speech_speaker() (folia.main.Chunk method), 545
speech_speaker() (folia.main.ChunkingLayer method), 686
speech_speaker() (folia.main.CoreferenceChain method), 557
speech_speaker() (folia.main.CoreferenceLayer method), 698
speech_speaker() (folia.main.CoreferenceLink method), 811
speech_speaker() (folia.main.Definition method), 109
speech_speaker() (folia.main.DependenciesLayer method), 711
speech_speaker() (folia.main.Dependency method), 570
speech_speaker() (folia.main.DependencyDependent method),
speech_speaker() (folia.main.Division method), 748
speech_speaker() (folia.main.Statement method), 634
speech_speaker() (folia.main.StatementLayer method), 761
speech_speaker() (folia.main.SubjectivityAnnotation method), 507
speech_speaker() (folia.main.SyntacticUnit method), 647
speech_speaker() (folia.main.SyntaxLayer method), 773
speech_speaker() (folia.main.Table method), 360
speech_speaker() (folia.main.TableHead method), 388
speech_speaker() (folia.main.Terms method), 402
speech_speaker() (folia.main.Text method), 519
speech_speaker() (folia.main.TimeSegment method), 672
speech_speaker() (folia.main.TimingLayer method), 798
speech_speaker() (folia.main.Whitespace method), 416
speech_speaker() (folia.main.Word method), 433
speech_speaker() (folia.main.AbstractAnnotationLayer method), 66
speech_speaker() (folia.main.AbstractElement method), 26
speech_speaker() (folia.main.AbstractSpanAnnotation method), 53
speech_speaker() (folia.main.AbstractStructureElement method), 40
speech_speaker() (folia.main.AbstractTextMarkup method), 78
speech_speaker() (folia.main.Cell method), 95
speech_speaker() (folia.main.Chunk method), 545
speech_speaker() (folia.main.ChunkingLayer method), 686
speech_speaker() (folia.main.CoreferenceChain method), 558
speech_speaker() (folia.main.CoreferenceLayer method), 698
speech_speaker() (folia.main.DependenciesLayer method), 771
speech_speaker() (folia.main.Dependency method), 570
speech_speaker() (folia.main.DependencyDependent method), 824
speech_speaker() (folia.main.Division method), 123
Index

Folia Python Library Documentation, Release v2.2.2, Folia v2.2.1

speech_src() (folia.main.DomAnnot method), 447
speech_src() (folia.main.EntLayer method), 723
speech_src() (folia.main.Ent method), 583
speech_src() (folia.main.Entry method), 137
speech_src() (folia.main.Event method), 151
speech_src() (folia.main.ExExample method), 165
speech_src() (folia.main.Fig method), 179
speech_src() (folia.main.Gap method), 191
speech_src() (folia.main.H Method), 205
speech_src() (folia.main.H Span method), 837
speech_src() (folia.main.LangAnnot method), 471
speech_src() (folia.main.LemmaAnnot method), 483
speech_src() (folia.main.LineBreak method), 219
speech_src() (folia.main.List method), 233
speech_src() (folia.main.ListItem method), 247
speech_src() (folia.main.Note method), 261
speech_src() (folia.main.Observation method), 596
speech_src() (folia.main.ObservationLayer method), 736
speech_src() (folia.main.Paragraph method), 275
speech_src() (folia.main.Part method), 289
speech_src() (folia.main.PhContent method), 530
speech_src() (folia.main.PosAnnot method), 459
speech_src() (folia.main.Predicate method), 609
speech_src() (folia.main.Quote method), 303
speech_src() (folia.main.Reference method), 317
speech_src() (folia.main.Row method), 331
speech_src() (folia.main.SemRole method), 660
speech_src() (folia.main.SemRolesLayer method), 786
speech_src() (folia.main.SenseAnnot method), 495
speech_src() (folia.main.Sent method), 346
speech_src() (folia.main.Sentiment method), 622
speech_src() (folia.main.SentimentLayer method), 748
speech_src() (folia.main.Statement method), 634
speech_src() (folia.main.StatementLayer method), 761
speech_src() (folia.main.SubjAnnot method), 507
speech_src() (folia.main.SysUnit method), 647
speech_src() (folia.main.SysLayer method), 773
speech_src() (folia.main.SysHead method), 360
speech_src() (folia.main.SysTerm method), 374
speech_src() (folia.main.Text method), 402
speech_src() (folia.main.TextContent method), 519
speech_src() (folia.main.TimeSegment method), 673
speech_src() (folia.main.TimeLayer method), 798
speech_src() (folia.main.White method), 416
speech_src() (folia.main.Word method), 433
split() (folia.main.Word method), 433
splitword() (folia.main.Sent method), 346
Statement (class in folia.main), 624
StatementLayer (class in folia.main), 750
stricttext() (folia.main.AbsAnnotLayer method), 66
stricttext() (folia.main.AbsElement method), 26
stricttext() (folia.main.AbsSpanAnnot method), 53
stricttext() (folia.main.AbsStructureElem method), 40
stricttext() (folia.main.AbsTextMarkUp method), 78
stricttext() (folia.main.Cell method), 95
stricttext() (folia.main.Chunk method), 545
stricttext() (folia.main.ChunkingLayer method), 686
stricttext() (folia.main.CorefChain method), 558
stricttext() (folia.main.CorefLayer method), 698
stricttext() (folia.main.Def method), 811
stricttext() (folia.main.DefLayer method), 109
stricttext() (folia.main.DepsLayer method), 711
stricttext() (folia.main.Dependency method), 571
stricttext() (folia.main.DependencyDep method), 824
stricttext() (folia.main.Div method), 123
stricttext() (folia.main.DomAnnot method), 447
stricttext() (folia.main.EntLayer method), 724
stricttext() (folia.main.Ent method), 584
stricttext() (folia.main.Entry method), 137
stricttext() (folia.main.Event method), 151
stricttext() (folia.main.Example method), 165
stricttext() (folia.main.Fig method), 179
stricttext() (folia.main.Gap method), 191
stricttext() (folia.main.Head method), 205
stricttext() (folia.main.HSpan method), 837
stricttext() (folia.main.LAnnot method), 471
stricttext() (folia.main.LemmaAnnot method), 483
stricttext() (folia.main.LineBreak method), 219
stricttext() (folia.main.List method), 233
stricttext() (folia.main.ListItem method), 247
stricttext() (folia.main.Note method), 261
stricttext() (folia.main.Observation method), 596
stricttext () (folia.main.ObservationLayer method), 736
stricttext () (folia.main.Paragraph method), 275
stricttext () (folia.main.Part method), 289
stricttext () (folia.main.Phrase method), 531
stricttext () (folia.main.PosAnnotation method), 459
stricttext () (folia.main.Predicate method), 609
stricttext () (folia.main.Quote method), 303
stricttext () (folia.main.Reference method), 317
stricttext () (folia.main.Row method), 331
stricttext () (folia.main.SemanticRole method), 660
stricttext () (folia.main.SemanticRolesLayer method), 786
stricttext () (folia.main.SenseAnnotation method), 495
stricttext () (folia.main.Sentence method), 346
stricttext () (folia.main.Sentiment method), 622
stricttext () (folia.main.SentimentLayer method), 749
stricttext () (folia.main.Statement method), 635
stricttext () (folia.main.StatementLayer method), 761
stricttext () (folia.main.SubjectivityAnnotation method), 507
stricttext () (folia.main.SyntacticUnit method), 647
stricttext () (folia.main.SyntaxLayer method), 774
stricttext () (folia.main.Table method), 360
stricttext () (folia.main.TableHead method), 388
stricttext () (folia.main.Term method), 374
stricttext () (folia.main.Text method), 402
stricttext () (folia.main.TextContent method), 519
stricttext () (folia.main.TimeSegment method), 673
stricttext () (folia.main.TimingLayer method), 799
stricttext () (folia.main.Whitespace method), 416
stricttext () (folia.main.Word method), 433
SubjectivityAnnotation (class in folia.main), 497
SUBSET (folia.main.AbstractAnnotationLayer attribute), 58
SUBSET (folia.main.AbstractElement attribute), 19
SUBSET (folia.main.AbstractSpanAnnotation attribute), 45
SUBSET (folia.main.AbstractStructureElement attribute), 31
SUBSET (folia.main.AbstractTextMarkup attribute), 71
SUBSET (folia.main.Cell attribute), 86
SUBSET (folia.main.Chunk attribute), 537
SUBSET (folia.main.ChunkingLayer attribute), 678
SUBSET (folia.main.CoreferenceChain attribute), 550
SUBSET (folia.main.CoreferenceLayer attribute), 691
SUBSET (folia.main.CoreferenceLink attribute), 804
SUBSET (folia.main.Definition attribute), 100
SUBSET (folia.main.DependenciesLayer attribute), 703
SUBSET (folia.main.Dependency attribute), 563
SUBSET (folia.main.DependencyDependent attribute), 817
SUBSET (folia.main.Division attribute), 114
SUBSET (folia.main.DomainAnnotation attribute), 440
SUBSET (folia.main.EntitiesLayer attribute), 716
SUBSET (folia.main.Entity attribute), 576
SUBSET (folia.main.Entry attribute), 128
SUBSET (folia.main.Event attribute), 142
SUBSET (folia.main.Example attribute), 156
SUBSET (folia.main.Figure attribute), 170
SUBSET (folia.main.Gap attribute), 184
SUBSET (folia.main.Head attribute), 196
SUBSET (folia.main.Headspan attribute), 829
SUBSET (folia.main.LangAnnotation attribute), 464
SUBSET (folia.main.LemmaAnnotation attribute), 476
SUBSET (folia.main.Linebreak attribute), 210
SUBSET (folia.main.List attribute), 224
SUBSET (folia.main.ListItem attribute), 238
SUBSET (folia.main.Note attribute), 252
SUBSET (folia.main.Observation attribute), 589
SUBSET (folia.main.ObservationLayer attribute), 728
SUBSET (folia.main.Paragraph attribute), 266
SUBSET (folia.main.ParagraphLayer attribute), 280
SUBSET (folia.main.PhonContent attribute), 524
SUBSET (folia.main.PosAnnotation attribute), 452
SUBSET (folia.main.Predicate attribute), 601
SUBSET (folia.main.Phrase attribute), 294
SUBSET (folia.main.Reference attribute), 308
SUBSET (folia.main.Row attribute), 322
SUBSET (folia.main.SemanticRole attribute), 652
SUBSET (folia.main.SemanticRolesLayer attribute), 778
SUBSET (folia.main.SenseAnnotation attribute), 488
SUBSET (folia.main.Sentence attribute), 336
SUBSET (folia.main.Sentiment attribute), 614
SUBSET (folia.main.Statement attribute), 741
SUBSET (folia.main.StatementLayer attribute), 627
SUBSET (folia.main.StatementLayer attribute), 753
SUBSET (folia.main.SubjectivityAnnotation attribute), 500
SUBSET (folia.main.SyntacticUnit attribute), 640
SUBSET (folia.main.SyntaxLayer attribute), 766
SUBSET (folia.main.Table attribute), 352
SUBSET (folia.main.TableHead attribute), 380
SUBSET (folia.main.Term attribute), 366
SUBSET (folia.main.Text attribute), 394
SUBSET (folia.main.TextContent attribute), 512
SUBSET (folia.main.TimeSegment attribute), 665
SUBSET (folia.main.TimingLayer attribute), 791
SUBSET (folia.main.Whitespace attribute), 408
SUBSET (folia.main.Word attribute), 422
SyntaxUnit (class in folia.main), 636
SyntaxLayer (class in folia.main), 763
TEXTCONTAINER (folia.main.Event attribute), 142
TEXTCONTAINER (folia.main.Example attribute), 156
TEXTCONTAINER (folia.main.Figure attribute), 170
TEXTCONTAINER (folia.main.Gap attribute), 184
TEXTCONTAINER (folia.main.Head attribute), 196
TEXTCONTAINER (folia.main.Headspan attribute), 829
TEXTCONTAINER (folia.main.LangAnnotation attribute), 464
TEXTCONTAINER (folia.main.LemmaAnnotation attribute), 476
TEXTCONTAINER (folia.main.Linebreak attribute), 210
TEXTCONTAINER (folia.main.List attribute), 224
TEXTCONTAINER (folia.main.ListItem attribute), 238
TEXTCONTAINER (folia.main.Note attribute), 252
TEXTCONTAINER (folia.main.Observation attribute), 589
TEXTCONTAINER (folia.main.ObservationLayer attribute), 728
TEXTCONTAINER (folia.main.Paragraph attribute), 266
TEXTCONTAINER (folia.main.Part attribute), 280
TEXTCONTAINER (folia.main.Phrase attribute), 524
TEXTCONTAINER (folia.main.PosAnnotation attribute), 452
TEXTCONTAINER (folia.main.Predicate attribute), 601
TEXTCONTAINER (folia.main.Quote attribute), 294
TEXTCONTAINER (folia.main.Reference attribute), 308
TEXTCONTAINER (folia.main.Row attribute), 322
TEXTCONTAINER (folia.main.SemanticRole attribute), 652
TEXTCONTAINER (folia.main.SemanticRolesLayer attribute), 778
TEXTCONTAINER (folia.main.Sentence attribute), 336
TEXTCONTAINER (folia.main.Sentiment attribute), 614
TEXTCONTAINER (folia.main.SentimentLayer attribute), 741
TEXTCONTAINER (folia.main.Statement attribute), 627
TEXTCONTAINER (folia.main.StatementLayer attribute), 753
TEXTCONTAINER (folia.main.SubjectivityAnnotation attribute), 500
TEXTCONTAINER (folia.main.SyntaxUnit attribute), 640
TEXTCONTAINER (folia.main.SyntaxLayer attribute), 766
TEXTCONTAINER (folia.main.Table attribute), 352
TEXTCONTAINER (folia.main.TableHead attribute), 380
TEXTCONTAINER (folia.main.Term attribute), 366
TEXTCONTAINER (folia.main.Text attribute), 394
TEXTCONTAINER (folia.main.TextContent attribute), 512
TEXTCONTAINER (folia.main.TimeSegment attribute), 665
TEXTDELMITER (folia.main.Row attribute), 322
TEXTDELMITER (folia.main.SemanticRole attribute), 652
TEXTDELMITER (folia.main.SemanticRolesLayer attribute), 779
TEXTDELMITER (folia.main.SenseAnnotation attribute), 488
TEXTDELMITER (folia.main.Sentence attribute), 336
TEXTDELMITER (folia.main.Sentiment attribute), 614
TEXTDELMITER (folia.main.SentimentLayer attribute), 741
TEXTDELMITER (folia.main.Statement attribute), 627
TEXTDELMITER (folia.main.StatementLayer attribute), 754
TEXTDELMITER (folia.main.SubjectivityAnnotation attribute), 500
TEXTDELMITER (folia.main.SyntacticUnit attribute), 640
TEXTDELMITER (folia.main.SyntaxLayer attribute), 766
TEXTDELMITER (folia.main.Table attribute), 352
TEXTDELMITER (folia.main.TableHead attribute), 380
TEXTDELMITER (folia.main.Term attribute), 366
TEXTDELMITER (folia.main.Text attribute), 394
TEXTDELMITER (folia.main.TextContent attribute), 512
TEXTDELMITER (folia.main.TimeSegment attribute), 665
TEXTDELMITER (folia.main.TimingLayer attribute), 791
TEXTDELMITER (folia.mainWhitespace attribute), 408
TEXTDELMITER (folia.main.Word attribute), 422
textvalidation() (folia.main.AbstractAnnotationLayer method), 67
textvalidation() (folia.main.AbstractElement method), 27
textvalidation() (folia.main.AbstractSpanAnnotation method), 54
textvalidation() (folia.main.AbstractStructureElement method), 41
textvalidation() (folia.main.AbstractTextMarkup method), 79
textvalidation() (folia.main.Cell method), 96
textvalidation() (folia.main.Chunk method), 546
textvalidation() (folia.main.ChunkingLayer method), 687
textvalidation() (folia.main.CoreferenceChain method), 559
textvalidation() (folia.main.CoreferenceLayer method), 700
textvalidation() (folia.main.CoreferenceLink method), 813
textvalidation() (folia.main.Definition method), 110
textvalidation() (folia.main.DependenciesLayer method), 712
textvalidation() (folia.main.Dependency method), 572
textvalidation() (folia.main.DependencyDependent method), 825
textvalidation() (folia.main.Division method), 124
textvalidation() (folia.main.DomainAnnotation method), 448
textvalidation() (folia.main.EntitiesLayer method), 725
textvalidation() (folia.main.Entry method), 585
textvalidation() (folia.main.Entry method), 138
textvalidation() (folia.main.Event method), 152
textvalidation() (folia.main.Example method), 166
textvalidation() (folia.main.Figure method), 180
textvalidation() (folia.main.Gap method), 192
textvalidation() (folia.main.Head method), 206
textvalidation() (folia.main.Headspan method), 838
textvalidation() (folia.main.LangAnnotation method), 472
textvalidation() (folia.main.Linebreak method), 484
textvalidation() (folia.main.Linebreak method), 220
textvalidation() (folia.main.List method), 234
textvalidation() (folia.main.ListItem method), 248
textvalidation() (folia.main.Note method), 262
textvalidation() (folia.main.Observation method), 597
textvalidation() (folia.main.ObservationLayer method), 737
textvalidation() (folia.main.Paragraph method), 276
textvalidation() (folia.main.Part method), 290
textvalidation() (folia.main.Phrase method), 532
textvalidation() (folia.main.Predicate method), 610
textvalidation() (folia.main.Predicative method), 304
textvalidation() (folia.main.Reference method), 318
textvalidation() (folia.main.Row method), 332
FoLiA Python Library Documentation, Release v2.2.2, FoLiA v2.2.1

textvalidation() (folia.main.SemanticRole method), 661
textvalidation() (folia.main.SemanticRolesLayer method), 787
textvalidation() (folia.main.SenseAnnotation method), 496
textvalidation() (folia.main.Sentence method), 348
textvalidation() (folia.main.Sentiment method), 623
textvalidation() (folia.main.SentimentLayer method), 750
textvalidation() (folia.main.Statement method), 636
textvalidation() (folia.main.StatementLayer method), 762
textvalidation() (folia.main.SubjectivityAnnotation method), 508
textvalidation() (folia.main.SyntacticUnit method), 648
textvalidation() (folia.main.SyntaxLayer method), 775
textvalidation() (folia.main.Table method), 362
textvalidation() (folia.main.TableHead method), 390
textvalidation() (folia.main.Term method), 376
textvalidation() (folia.main.Text method), 404
textvalidation() (folia.main.TextContent method), 520
textvalidation() (folia.main.TimeSegment method), 674
textvalidation() (folia.main.TimingLayer method), 800
textvalidation() (folia.main.Whitespace method), 418
textvalidation() (folia.main.Word method), 434
TimeSegment (class in folia.main), 662
timingLayer (class in folia.main), 788
title() (folia.main.Document method), 13
toktext() (folia.main.AbstractAnnotationLayer method), 67
(toktext() (folia.main.AbstractElement method), 28
(toktext() (folia.main.AbstractSpanAnnotation method), 54
toktext() (folia.main.AbstractStructureElement method), 41
toktext() (folia.main.AbstractTextMarkup method), 79
(toktext() (folia.main.Cell method), 96
(toktext() (folia.main.Chunk method), 546
(toktext() (folia.main.ChunkingLayer method), 687
(toktext() (folia.main.CoreferenceChain method), 559
toktext() (folia.main.TextContent method), 520
toktext() (folia.main.TimeSegment method), 674
toktext() (folia.main.TimingLayer method), 800
toktext() (folia.main.WhiteSpace method), 418
toktext() (folia.main.Word method), 434

U
unalias() (folia.main.Document method), 13
updatetext() (folia.main.AbstractAnnotationLayer method), 67
updatetext() (folia.main.AbstractElement method), 28
updatetext() (folia.main.AbstractSpanAnnotation method), 54
updatetext() (folia.main.AbstractStructureElement method), 41
updatetext() (folia.main.AbstractTextMarkup method), 79
updatetext() (folia.main.Cell method), 96
updatetext() (folia.main.Chunk method), 546
updatetext() (folia.main.ChunkingLayer method), 687
updatetext() (folia.main.CoreferenceChain method), 559
updatetext() (folia.main.CoreferenceLayer method), 700
updatetext() (folia.main.CoreferenceLink method), 813
updatetext() (folia.main.Definition method), 110
updatetext() (folia.main.DependenciesLayer method), 712
updatetext() (folia.main.Dependency method), 572
updatetext() (folia.main.DependencyDependent method), 825
updatetext() (folia.main.Division method), 124
updatetext() (folia.main.DomainAnnotation method), 448
updatetext() (folia.main.EntitiesLayer method), 725
updatetext() (folia.main.Entity method), 585
updatetext() (folia.main.Entry method), 138
updatetext() (folia.main.Event method), 152
updatetext() (folia.main.Example method), 166
updatetext() (folia.main.Figure method), 180
updatetext() (folia.main.Figure method), 192
updatetext() (folia.main.Head method), 206
updatetext() (folia.main.Headspan method), 838
updatetext() (folia.main.LangAnnotation method), 472
updatetext() (folia.main.LemmaAnnotation method), 484
updatetext() (folia.main.Linebreak method), 420
updatetext() (folia.main.List method), 234
updatetext() (folia.main.ListItem method), 248
updatetext() (folia.main.Note method), 262
updatetext() (folia.main.Observation method), 597
updatetext() (folia.main.ObservationLayer method), 737
updatetext() (folia.main.Paragraph method), 276
updatetext() (folia.main.Part method), 290
updatetext() (folia.main.PhonContent method), 532
updatetext() (folia.main.PosAnnotation method), 460
updatetext() (folia.main.Predicate method), 610
updatetext() (folia.main.Quote method), 304
updatetext() (folia.main.Reference method), 318
updatetext() (folia.main.Row method), 332
updatetext() (folia.main.SemanticRole method), 661
updatetext() (folia.main.SemanticRolesLayer method), 787
updatetext() (folia.main.Sentence method), 496
updatetext() (folia.main.Sentence method), 348
updatetext() (folia.main.Sentiment method), 623
updatetext() (folia.main.SentimentLayer method), 750
updatetext() (folia.main.Statement method), 636
updatetext() (folia.main.StatementLayer method), 762
updatetext() (folia.main.SubjectivityAnnotation method), 508
updatetext() (folia.main.SyntacticUnit method), 648
updatetext() (folia.main.SyntacticUnitLayer method), 775
updatetext() (folia.main.Table method), 362
updatetext() (folia.main.TableHead method), 390
updatetext() (folia.main.Term method), 376
updatetext() (folia.main.Text method), 404
updatetext() (folia.main.TextContent method), 520
updatetext() (folia.main.TimeSegment method), 674
updatetext() (folia.main.TimingLayer method), 800
updatetext() (folia.main.WhiteSpace method), 418
updatetext() (folia.main.Word method), 434

W
Whitespace (class in folia.main), 404
Word (class in folia.main), 418
words() (folia.main.AbstractStructureElement method), 42
words() (folia.main.Cell method), 96
words() (folia.main.Definition method), 110
words() (folia.main.Division method), 124
words() (folia.main.Document method), 13
words() (folia.main.Entry method), 138
words() (folia.main.Event method), 152
words() (folia.main.Example method), 166
words() (folia.main.Figure method), 180
words() (folia.main.Head method), 206
words() (folia.main.Head method), 206
words() (folia.main.Linebreak method), 220
<table>
<thead>
<tr>
<th>XML Link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(folia.main.AbstractSpanAnnotation attribute), 46</td>
<td></td>
</tr>
<tr>
<td>(folia.main.AbstractStructureElement attribute), 32</td>
<td></td>
</tr>
<tr>
<td>(folia.main.AbstractTextMarkup attribute), 71</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Cell attribute), 86</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Chunk attribute), 537</td>
<td></td>
</tr>
<tr>
<td>(folia.main.ChunkingLayer attribute), 678</td>
<td></td>
</tr>
<tr>
<td>(folia.main.CoreferenceChain attribute), 550</td>
<td></td>
</tr>
<tr>
<td>(folia.main.CoreferenceLayer attribute), 691</td>
<td></td>
</tr>
<tr>
<td>(folia.main.CoreferenceLink attribute), 804</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Definition attribute), 100</td>
<td></td>
</tr>
<tr>
<td>(folia.main.DependenciesLayer attribute), 704</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Dependency attribute), 563</td>
<td></td>
</tr>
<tr>
<td>(folia.main.DependencyDependent attribute), 817</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Division attribute), 114</td>
<td></td>
</tr>
<tr>
<td>(folia.main.DomainAnnotation attribute), 440</td>
<td></td>
</tr>
<tr>
<td>(folia.main.EntitiesLayer attribute), 716</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Entity attribute), 576</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Entry attribute), 128</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Event attribute), 142</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Example attribute), 156</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Figure attribute), 170</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Gap attribute), 184</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Head attribute), 196</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Headspan attribute), 830</td>
<td></td>
</tr>
<tr>
<td>(folia.main.LangAnnotation attribute), 464</td>
<td></td>
</tr>
<tr>
<td>(folia.main.LemmaAnnotation attribute), 476</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Linebreak attribute), 210</td>
<td></td>
</tr>
<tr>
<td>(folia.main.List attribute), 224</td>
<td></td>
</tr>
<tr>
<td>(folia.main.ListItem attribute), 238</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Note attribute), 252</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Observation attribute), 589</td>
<td></td>
</tr>
<tr>
<td>(folia.main.ObservationLayer attribute), 729</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Paragraph attribute), 266</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Part attribute), 280</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Predicate attribute), 524</td>
<td></td>
</tr>
<tr>
<td>(folia.main.PosAnnotation attribute), 452</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Predicate attribute), 601</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Reference attribute), 308</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Row attribute), 322</td>
<td></td>
</tr>
<tr>
<td>(folia.main.SemanticRole attribute), 652</td>
<td></td>
</tr>
<tr>
<td>(folia.main.SemanticRolesLayer attribute), 779</td>
<td></td>
</tr>
<tr>
<td>(folia.main.SenseAnnotation attribute), 488</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Sentence attribute), 337</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Sentiment attribute), 614</td>
<td></td>
</tr>
<tr>
<td>(folia.main.StatementLayer attribute), 741</td>
<td></td>
</tr>
<tr>
<td>(folia.main.Statement attribute), 627</td>
<td></td>
</tr>
<tr>
<td>(folia.main.StatementLayer attribute), 754</td>
<td></td>
</tr>
<tr>
<td>(folia.main.SubjectivityAnnotation attribute), 500</td>
<td></td>
</tr>
<tr>
<td>(folia.main.SyntacticUnit attribute), 640</td>
<td></td>
</tr>
<tr>
<td>(folia.main_SyntaxLayer attribute), 766</td>
<td></td>
</tr>
</tbody>
</table>
XMLTAG (folia.main.AbstractElement attribute), 19
XMLTAG (folia.main.AbstractSpanAnnotation attribute), 46
XMLTAG (folia.main.AbstractStructureElement attribute), 32
XMLTAG (folia.main.AbstractTextMarkup attribute), 71
XMLTAG (folia.main.Cell attribute), 86
XMLTAG (folia.main.Chunk attribute), 537
XMLTAG (folia.main.ChunkingLayer attribute), 678
XMLTAG (folia.main.CoreferenceChain attribute), 550
XMLTAG (folia.main.CoreferenceLayer attribute), 691
XMLTAG (folia.main.CoreferenceLink attribute), 804
XMLTAG (folia.main.Definition attribute), 100
XMLTAG (folia.main.DependenciesLayer attribute), 704
XMLTAG (folia.main.Dependency attribute), 563
XMLTAG (folia.main.DependencyDependent attribute), 817
XMLTAG (folia.main.Division attribute), 114
XMLTAG (folia.main.DomainAnnotation attribute), 440
XMLTAG (folia.main.EntitiesLayer attribute), 716
XMLTAG (folia.main.Entity attribute), 576
XMLTAG (folia.main.Entry attribute), 128
XMLTAG (folia.main.Event attribute), 142
XMLTAG (folia.main.Example attribute), 156
XMLTAG (folia.main.Figure attribute), 170
XMLTAG (folia.main.Gap attribute), 184
XMLTAG (folia.main.Head attribute), 196
XMLTAG (folia.main.Headspan attribute), 830
XMLTAG (folia.main.LangAnnotation attribute), 464
XMLTAG (folia.main.LemmaAnnotation attribute), 476
XMLTAG (folia.main.Linebreak attribute), 210
XMLTAG (folia.main.List attribute), 224
XMLTAG (folia.main.ListItem attribute), 238
XMLTAG (folia.main.Note attribute), 252
XMLTAG (folia.main.Observation attribute), 589
XMLTAG (folia.main.ObservationLayer attribute), 729
XMLTAG (folia.main.Paragraph attribute), 266
XMLTAG (folia.main.Part attribute), 280
XMLTAG (folia.main.PhonContent attribute), 524
XMLTAG (folia.main.PosAnnotation attribute), 452
XMLTAG (folia.main.Predicate attribute), 601
XMLTAG (folia.main.Quote attribute), 294
XMLTAG (folia.main.Reference attribute), 308
XMLTAG (folia.main.Row attribute), 322
XMLTAG (folia.main.SemanticRole attribute), 652
XMLTAG (folia.main.SemanticRolesLayer attribute), 779
XMLTAG (folia.main.SenseAnnotation attribute), 488
XMLTAG (folia.main.Statement attribute), 337
XMLTAG (folia.main.Sentiment attribute), 614
XMLTAG (folia.main.SentimentLayer attribute), 741
XMLTAG (folia.main.Statement attribute), 627
XMLTAG (folia.main.StatementLayer attribute), 754
XMLTAG (folia.main.SubjectivityAnnotation attribute), 500

xpath() (folia.main.Document method), 14

XMLTAG (folia.main.SyntacticUnit attribute), 640
XMLTAG (folia.main.SyntaxLayer attribute), 766
XMLTAG (folia.main.Table attribute), 352
XMLTAG (folia.main.TableHead attribute), 380
XMLTAG (folia.main.Term attribute), 366
XMLTAG (folia.main.Text attribute), 394
XMLTAG (folia.main.TextContent attribute), 512
XMLTAG (folia.main.TimeSegment attribute), 665
XMLTAG (folia.main.TimingLayer attribute), 791
XMLTAG (folia.main.Whitespace attribute), 408
XMLTAG (folia.main.Word attribute), 423

942 Index