# FoLiA Tools

*Release 2.0.0*

**Maarten van Gompel**

**Jan 12, 2022**

# Contents


**1 Installation** **3**

**2 Installation Troubleshooting** **5**

**3 Usage** **7**

**4 More about FoLiA?** **9**

**5 Specific Tools** **11**
5.1 Validating FoLiA documents using foliavalidator . . . . . 11
5.2 TEI to FoLiA conversion . . . . . 12
5.3 FoLiA to Salt . . . . . 12

A number of command-line tools are readily available for working with FoLiA, to various ends. The following tools are currently available:

- `foliavalidator` – Tests if documents are valid FoLiA XML. **Always use this to test your documents if you produce your own FoLiA documents!**. See the extra documentation: *Validating FoLiA documents using foliavalidator*
- `foliaquery` – Advanced query tool that searches FoLiA documents for a specified pattern, or modifies a document according to the query. Supports FQL (FoLiA Query Language) and CQL (Corpus Query Language).
- `foliaeval` – Evaluation tool, can compute various evaluation metrics for selected annotation types, either against a gold standard reference or as a measure of inter-annotated agreement.
- `folia2txt` – Convert FoLiA XML to plain text (pure text, without any annotations). Use this to extract plain text from any FoLiA document.
- `folia2annotatedtxt` – Like above, but produces output simple token annotations inline, by appending them directly to the word using a specific delimiter.
- `folia2columns` – This conversion tool reads a FoLiA XML document and produces a simple columned output format (including CSV) in which each token appears on one line. Note that only simple token annotations are supported and a lot of FoLiA data can not be intuitively expressed in a simple columned format!
- `folia2html` – Converts a FoLiA document to a semi-interactive HTML document, with limited support for certain token annotations.
- `folia2dcoi` – Convert FoLiA XML to D-Coi XML (only for annotations supported by D-Coi)
- `foliatree` – Outputs the hierarchy of a FoLiA document.
- `foliacat` – Concatenate multiple FoLiA documents.
- `foliacount` – This script reads a FoLiA XML document and counts certain structure elements.
- `foliacorrect` – A tool to deal with corrections in FoLiA, can automatically accept suggestions or strip all corrections so parsers that don't know how to handle corrections can process it.
- `foliaerase` – Erases one or more specified annotation types from the FoLiA document.
- `folialangid` – Does language detection on FoLiA documents, assigns language identifiers to different substructures
- `foliaid` – Assigns IDs to elements in FoLiA documents. Use this to automatically generate identifiers on certain (or all) elements.
- `foliafreqlist` – Output a frequency list on tokenised FoLiA documents.
- `foliamerge` – Merges annotations from two or more FoLiA documents.
- `foliatextcontent` – A tool for adding or stripping text redundancy (i.e. text associated with multiple structural levels), supports computing and adding offset information. Use this if you want to have text available on a different level (e.g. the global text level).
- `foliaupgrade` – Upgrades a document to the latest FoLiA version.
- `alpino2folia` – Convert Alpino-DS XML to FoLiA XML
- `dcoi2folia` – Convert D-Coi XML to FoLiA XML
- `conllu2folia` – Convert files in the CONLL-U format to FoLiA XML.
- `rst2folia` – Convert ReStructuredText, a lightweight non-intrusive text markup language, to FoLiA, using docutils.
- `tei2folia` – Convert a subset of TEI to FoLiA. See the extra documentation: *TEI to FoLiA conversion*
- `folia2salt` – Convert FoLiA XML to Salt, which in turn enables further conversions (annis, paula, TCF, TigerXML, and others) through Pepper. See the extra documentation: *FoLiA to Salt*

All of these tools are written in Python, and thus require a Python 3 installation to run. More tools are added as time progresses.

CHAPTER 1

# Installation

The FoLiA tools are published to the Python Package Index and can be installed effortlessly using `pip`, from the command-line, type:

```
$ pip install folia-tools
```

You may need to use `pip3` to ensure you have the Python 3 version. Add `sudo` to install it globally on your system, but we strongly recommend you use virtualenv to make a self-contained Python environment.

The FoLiA tools are also included in our LaMachine distribution .

CHAPTER 2

# Installation Troubleshooting

If `pip` is not yet available, install it as follows:

On Debian/Ubuntu-based systems:

```
$ sudo apt-get install python3-pip
```

On RedHat-based systems:

```
$ yum install python3-pip
```

On Arch Linux systems:

```
$ pacman -Syu python-pip
```

# CHAPTER 3

# Usage

To obtain help regarding the usage of any of the available FoLiA tools, please pass the `-h` option on the command line to the tool you intend to use. This will provide a summary on available options and usage examples. Most of the tools can run on both a single FoLiA document, as well as a whole directory of documents, allowing also for recursion. The tools generally take one or more file names or directory names as parameters.

CHAPTER 4

# More about FoLiA?

Please consult the FoLiA website at https://proycon.github.io/folia for more!

CHAPTER 5

# Specific Tools

This section contains some extra important information for a few of the included tools.

## 5.1 Validating FoLiA documents using foliavalidator

The FoLiA validator is an essential tool for anybody working with FoLiA. It is very important that FoLiA documents are properly validated before they are published, this ensures that tools know what to expect when they get a FoLiA document as input for processing and are not confronted with any nasty surprises that are far too common in the field. The degree of formal validation offered by FoLiA is something that sets it apart from many alternative annotation formats. The key tool to perform validation is `foliavalidator` (or its alternative C++ implementation `folialint` as part of FoLiA-utils).

Validation can proceed on two levels: 1. **shallow validation** - Validates the full FoLiA document, checks if all elements are valid FoLiA elements,

> properly used, and if the document structure is valid. Checks if all the proper annotation declarations are present and if there are no inconsistencies in the text if text is specified on multiple levels (text redundancy). Note that shallow validation already does way more than validation against the RelaxNG Schema does.

2. **deep validation** - Does all of the above, but in addition it also checks the actual tagsets used. It checks if all declarations refer to valid set definition and if all used classes (aka tags/labels) are valid according to the declared set definitions and if the combination of certain classes is valid according to the set definition.

Note that validation against merely the RelaxNG schema could be called naive validation and is **NOT** considered sufficient FoLiA validation for most intents and purposes.

Shallow validation is invoked as: `$ foliavalidator document.folia.xml`. Deep validation invoked as: `$ foliavalidator --deep document.folia.xml`.

In addition to validating, the foliavalidator tool is capable of automatically fixing certain validation problems when explicitly asked to do so, such as automatically declaring missing annotations.

Another feature of the validator is that it can get as a converter to convert FoLiA documents to explicit form (using the `--explicit` parameter). Explicit form is a more verbose form of XML serialisation that is easier to parse to certain tools as it makes explicit certain details that are left implicit in normal form.

## 5.2 TEI to FoLiA conversion

The TEI P5 guidelines (Text Encoding Initiative) specify a widely used encoding method for machine-readable texts. It is primarly a format for capture text structure and markup in great detail, but there are some facilities for linguistic annotation too. The sheer flexibility and complexity of TEI leads to many different TEI dialects, and subsequently implementing support for TEI (all-of-it) in a tool is an almost impossible task. FoLiA is more constrained than TEI with regard to structural and markup annotation, but places more focus on linguistic annotation.

The `tei2folia` tool performs conversion from a (sizable) subset of TEI to FoLiA, but provides no guarantee that all TEI P5 documents can be processed. Some notable things that are supported:

- Conversion of text structure including divisions, paragraphs, headers & titles, lists, figures, tables (limited), front matter, back matter
- Verse text (limited, no metrical analysis etc), line groups (`<lg>`)
- Gaps
- **Text markup (highlighting, `<hi>`), emphasis, foreign, term, mentioned, names and places**
  - Limited corrections
- Conversion of lightweigth linguistic annotation.
- **Linguistic segments: sentences (`<s>`) & words (`w`), but not `<cl>` nor `<phr>`.**
  - Basic tokenisation (spacing) information (TEI's `@join` attribute)
- Limited metadata

Specifically not supported (yet), non-exhaustive list:

- Graphs and trees
- Milestones
- Span groups, interpretration groups, link groups (`<spanGrp>`, `<interpGrp>`, `<linkGrp>`)
- Speech
- Contextual information
- Feature structures (`<fs>`, `<f>`)

## 5.3 FoLiA to Salt

Salt is a graph based annotation model that is designed to act as an intermediate format in the conversion between various annotation formats. It is used by the conversion tool Pepper. Our FoLiA to Salt converter, however, is a standalone tool as part of these FoLiA tools, rather than integrated into pepper. You can use `folia2salt` to convert FoLiA XML to Salt XML and subsequently use Pepper to do conversions to other formats such as TCF, PAULA, TigerXML, GraF, Annis, etc… (there is no guarantee though that everything can be preserved accurately in each conversion).

The current state of this conversion is summarised below:

- Conversion of FoLiA tokens to salt SToken nodes * The converter only supports tokenised FoLiA documents
- Text extraction (from tokens) to STextualDS node and conversion to STextualRelation edges * preserves untokenised text only to a certain degree (using FoLiA's token spacing information only) * **not yet supported**: multiple text classes
- Conversion of FoLiA Inline Annotation (pos, lemma etc) to salt SAnnotation labels
- Conversion of FoLiA Structure Annotation (sentences,paragraph, etc) to salt SSpan nodes and SSpanRelation edges * converted structures will directly relate to the underlying token nodes rather than to a structural hierarchy like in FoLiA

- **Conversion of simple FoLiA Span Annotation (entities etc) to salt SSpan nodes and SSpanRelation edges**
  - Conversion of nested Span Annotation (syntax etc) to SSpan nodes and SDominanceRelation edges
  - **not yet supported**: Span Annotation including span roles (dependencies etc) to SSpan nodes and SDominanceRelation edges
- Grouping of annotation types/sets in salt SLayer nodes
- **Conversion of FoLiA higher order elements:**
  - Features
  - Comments
  - Descriptions
  - **not yet supported:**
    - Relations
    - Metrics
    - Span Relations
    - String annotation
    - Alternative annotation
    - Corrections
- Conversion of FoLiA phonetic content (as an extra STextualDS node and STextualRelation edges)
- Convert FoLiA native metadata
- **not yet supported:**
  - Conversion of FoLiA subtoken annotation (morphology/phonology)
  - Conversion of FoLiA references to audio/video sources and timing information

Our Salt conversion tries to preserve as much of the FoLiA as possible, we extensively use salt's capacity for specifying namespaces to hold and group the annotation type and set of an annotation. SLabel elements with the same namespace should often be considered together.