

tatuylonen /
wikitextprocessor<> **Code**

Issues 4

Pull requests 1

Actions

Projects

Security ...




Python package for WikiMedia dump processing (Wiktionary, Wikipedia etc). Wikitext parsing, template expansion, Lua module execution. For data extraction, bulk syntax checking, error detection, and offline formatting.



View license




☆ **67** stars **19** forks **5** watching Activity

Public repository


 main ▾

...

 Branches  Tags

 **kristian-clausal** Merge pull request [#87](#) from tatuylonen/wikinode... ...  2 days ago  **690**

[View code](#)

 **README.md**

Releases

wikitextprocessor

8 tags

This is a Python package for processing [WikiMedia dump files](#) for [Wiktionary](#), [Wikipedia](#), etc., for data extraction, error checking, offline conversion into HTML or other formats, and other uses. Key features include:

No packages published

- Parsing dump files, including built-in support for processing pages in parallel

Used by 8

- [Wikitext](#) syntax parser that converts the whole page into a parse tree

- Extracting template definitions and [Scribunto](#) Lua module definitions from dump



- Expanding selected templates or all templates, and heuristically identifying templates that need to be expanded before parsing is reasonably possible (e.g., [Scribunto](#)'s `table` tags that emit table start and end tags)

Contributors

- Processing and expanding [wikitext](#) parser functions
- Processing, executing, and expanding [Scribunto](#) Lua modules (they are very widely used in, e.g., [Wiktionary](#), for example for generating [IPA](#) strings for many languages)

Deployments

- Controlled expansion of parts of pages for applications that parse overall page structure before parsing but then expand templates on certain sections of the page
- Capturing information from template arguments while expanding them, as template arguments often contain useful information not available in the expanded content.

Languages

This module is primarily intended as a building block for other packages that process Wiktionary or Wikipedia data, particularly for data extraction. You will need to write code

to use this.

● **Lua** 46.5% ● **Python** 32.9% ● **PHP** 19.8% ● **JavaScript** 0.7% ● **CSS** 0.1% ● **Shell** 0.0%

For pre-existing extraction modules that use this package, please see:

- [Wiktextextract](#) for extracting rich machine-readable dictionaries from Wiktionary. You can also find pre-extracted machine-readable Wiktionary data in JSON format at

kaikki.org.

Getting started

Installing

Install from source:

```
git clone https://github.com/tatuylonen/wikitextprocessor.git
cd wikitextprocessor
python -m venv .venv
source .venv/bin/activate
python -m pip install -U pip
python -m pip install --use-pep517 .
```



Alternatively, you can install from pypi.org:

```
python -m pip install wikitextprocessor
```



Running tests

This package includes tests written using the `unittest` framework. They can be run using, for example, `nose2`, which can be installed using `python -m pip install -e --use-pep517 ".[dev]"`.

To run the tests, use the following command in the top-level directory:

```
make test
```



Obtaining WikiMedia dump files

This package is primarily intended for processing Wiktionary and Wikipedia dump files (though you can also use it for processing individual pages or other files that are in wikitext format). To download WikiMedia dump files, go to the [dump download page](#). We recommend using the `<name>-<date>-pages-articles.xml.bz2` files.

API documentation

Usage example:

```
from wikitextprocessor import Wtp, WikiNode, NodeKind, Page
ctx = Wtp()

def page_handler(page: Page) -> None:
    if page.model != "wikitext" or page.title.startswith("Template:"):
        return None
    tree = ctx.parse(page.body, pre_expand=True)
    ... process parse tree
    ... value = ctx.node_to_wikitext(node)

ctx.process("enwiktionary-20201201-pages-articles.xml.bz2", page_handler)
# for testing, you can iterate over ctx.process(...) with
# list(ctx.process...) to actually have the iterator run.
```



The basic operation of `wtp.process()` is as follows:

- Extract templates, modules, and other pages from the dump file and save them in a temporary file
- Heuristically analyze which templates need to be pre-expanded before parsing to make sense of the page structure (this cannot detect templates that call Lua code that outputs wikitext that affects parsed structure). These first steps together are called the "first phase".
- Process the pages again, calling a page handler function for each page. The page handler can extract, parse, and otherwise process the page, and has full access to templates and Lua macros defined in the dump. This may call the page handler in multiple processes in parallel. Return values from the page handler calls are returned to the caller (this function acts as an iterator). This is called the second phase.
- Optionally, the `wtp.reprocess()` function may be used for processing the same data several times (it basically repeats the second phase).

Most of the functionality is hidden behind the `wtp` object. `WikiNode` objects are used for representing the parse tree that is returned by the `wtp.parse()` function.

`NodeKind` is an enumeration type used to encode the type of a `WikiNode`.

class Wtp

```
def __init__(self, num_threads: Optional[int] = None, db_path: Optional[Path] = None,
              quiet: bool = False, lang_code: str = "en", languages_by_code
```



```
template_override_funcs: Dict[str, Callable[List[str], str]] :
```

The initializer can usually be called without arguments, but recognizes the following arguments:

- `num_threads` - if set to an integer, use that many parallel processes for processing the dump. The default is to use as many processors as there are available cores/hyperthreads. You may need to limit the number of parallel processes if you are limited by available memory; we have found that processing Wiktionary (including templates and Lua macros) requires 3-4GB of memory per process. This **MUST** be set to 1 on Windows.
- `db_path` can be `None`, in which case a temporary database file will be created under `/tmp`, or a path for the database file which contains page texts and other data of the dump file. There are two reasons why you might want to set this:
 - i. you don't have enough space on `/tmp` (3.4G for English dump file), or 2) for testing. If you specify the path and an existing database file exists, that file will be used, eliminating the time needed for Phase 1 (this is very important for testing, allowing processing single pages reasonably fast). In this case, you should not call `wtp.process()` but instead use `wtp.reprocess()` or just call `wtp.expand()` or `wtp.parse()` on wikitext that you have obtained otherwise (e.g., from some file). If the file doesn't exist, you will need to call `wtp.process()` to parse a dump file, which will initialize the database file during the first phase. If you wish to re-create the database, you should remove the old file first.
- `quiet` - if set to `True`, suppress progress messages during processing
- `lang_code` - the language code of the dump file.
- `languages_by_code` - Languages data.
- `template_override_funcs` - Python functions for overriding expanded template text.

Windows and MacOS note: Setting `num_threads` to a value other than 1 doesn't work on Windows and MacOS. It now defaults to 1 on these platforms. This is because these platforms don't use `fork()` in the Python multiprocessing package, and the current parallelization implementation depends on this.

```
def process(self, path: str, page_handler, namespace_ids: Set[int], phase1_override_folders: Optional[List[Path]] = None, skip_extract_dur
```

This function processes a Wikimedia dump, uncompressing and extracing pages (including templates and Lua modules), and calling `wtp.add_page()` for each page (phase 1). This then calls `wtp.reprocess()` to execute the second phase.

This takes the following arguments:

- `path` (str) - path to the Wikimedia dump file to be processed (e.g., "enwiktionary-20201201-pages-articles.xml.bz2"). Note that the compressed file can be used. Dump files can be downloaded [here](#).
- `page_handler` (function) - this function will be called for each page in phase 2 (unless `phase1_only` is set to `True`). The call takes the form `page_handler(page: Page)`, where `page.model` is the `model` value for the page in the dump (`wikitext` for normal wikitext pages and templates, `Scribunto` for Lua modules; other values are also possible), `page.title` is page title (e.g., `sample` or `Template:foobar` or `Module:mystic`), and `page.body` is the contents of the page (usually wikitext).
- `namespace_ids` - a set of namespace ids, pages have namespace ids that not included in this set won't be processed.
- `phase1_only` (boolean) - if set to `True`, prevents phase 2 processing and the `page_handler` function will not be called. The `wtp.reprocess()` function can be used to run the second phase separately, or `wtp.expand()`, `wtp.parse()` and other functions can be used.
- `override_folders` - a list of folder paths, each folder contains files for overriding pages in the dump file.
- `skip_extract_dump` - Extract dump file can be skipped if the database was created before.

This function returns an iterator over the values returned by the `page_handler` function (if `page_handler` returns `None` or no value, the iterator does not return those values). Note that `page_handler` will usually be run in a separate process, and cannot pass any values back in global variables. It can, however, access global variables assigned before calling `wtp.process()` (in Linux only).

```
def reprocess(self, page_handler, namespace_ids: Optional[List[int]] = None,
              include_redirects: bool = True):
```

Iterates over all pages in the cache file and calls `page_handler` for each page. This basically implements phase 2 of processing a dump file (see `wtp.process()`). This can be called more than once if desired.

The arguments are:

- `page_handler` (function) - as same as the argument in `wtp.process()` .
- `namespace_ids` - as same as the argument in `wtp.process()` .
- `include_redirects` - redirect pages will be processed if set to `True` .

This function returns an iterator that iterates over the return values of `page_handler` . If the return value is `None` or `page_handler` returns no value, no value is returned by the iterator for such calls.

This calls the `page_handler` using subprocesses (unless `num_threads` was set to 1 in the initializer). It may be necessary to set it to 1 on Windows and MacOS due to operating system/python limitations on those platforms.

```
def read_by_title(self, title: str, namespace_id: Optional[int] = None) ->
```

Reads the contents of the page with the specified title from the cache file. There is usually no need to call this function explicitly, as `wtp.process()` and `wtp.reprocess()` normally load the page automatically. This function does not automatically call `wtp.start_page()` .

Arguments are:

- `title` - the title of the page to read
- `namespace_id` - namespace id number, this argument is required if `title` doesn't have namespace prefix like `Template:` .

This returns the page contents as a string, or `None` if the page does not exist.

```
def parse(self, text, pre_expand=False, expand_all=False,
          additional_expand=None)
```

Parses wikitext into a parse tree (`wikiNode`), optionally expanding some or all the templates and Lua macros in the wikitext (using the definitions for the templates and macros in the cache files, as added by `wtp.process()` or calls to `wtp.add_page()` .

The `wtp.start_page()` function must be called before this function to set the page title (which may be used by templates, Lua macros, and error messages). The `wtp.process()` and `wtp.reprocess()` functions will call it automatically.

This accepts the following arguments:

- `text` (str) - the wikitext to be parsed
- `pre_expand` (boolean) - if set to `True`, the templates that were heuristically detected as affecting parsing (e.g., expanding to table start or end tags or list items) will be automatically expanded before parsing. Any Lua macros those templates use may also be called.
- `expand_all` - if set to `True`, expands all templates and Lua macros in the wikitext before parsing.
- `additional_expand` (set or `None`) - if this argument is provided, it should be a set of template names that should be expanded in addition to those specified by the other options (i.e., in addition to the heuristically detected templates if `pre_expand` is `True` or just these if it is false; this option is meaningless if `expand_all` is set to `True`).

This returns the parse tree. See below for a documentation of the `WikiNode` class used for representing the parse tree.

```
def node_to_wikitext(self, node)
```



Converts a part of a parse tree back to wikitext.

- `node` (`WikiNode`, str, list/tuple of these) - This is the part of the parse tree that is to be converted back to wikitext. We also allow strings and lists, so that `node.children` can be used directly as the argument.

```
def expand(self, text, template_fn=None, post_template_fn=None,  
           pre_expand=False, templates_to_expand=None,  
           expand_parserfns=True, expand_invoke=True)
```



Expands the selected templates, parser functions and Lua macros in the given Wikitext. This can selectively expand some or all templates. This can also capture the arguments and/or the expansion of any template as well as substitute custom expansions instead of the default expansions.

The `wtp.start_page()` function must be called before this function to set the page title (which may be used by templates and Lua macros). The `wtp.process()` and `wtp.reprocess()` will call it automatically. The page title is also used in error messages.

The arguments are as follows:

- `text` (str) - the wikitext to be expanded
- `template_fn` (function) - if set, this will be called as `template_fn(name, args)`, where `name` (str) is the name of the template and `args` is a dictionary containing arguments to the template. Positional arguments (and named arguments with numeric names) will have integer keys in the dictionary, whereas other named arguments will have their names as keys. All values corresponding to arguments are strings (after they have been expanded). This function can return `None` to cause the template to be expanded in the normal way, or a string that will be used instead of the expansion of the template. This can return `""` (empty string) to expand the template to nothing. This can also capture the template name and its arguments.
- `post_template_fn` (function) - if set, this will be called as `post_template_fn(name, ht, expansion)` after the template has been expanded in the normal way. This can return `None` to use the default expansion, or a string to use a that string as the expansion. This can also be used to capture the template, its arguments, and/or its expansion.
- `pre_expand` (boolean) - if set to `True`, all templates that were heuristically determined as needing to be expanded before parsing will be expanded.
- `templates_to_expand` (`None` or set or dictionary) - if this is set, these templates will be expanded in addition to any other templates that have been specified to be expanded. If a dictionary is provided, its keys will be taken as the names of the templates to be expanded. If this has not been set or is `None`, all templates will be expanded.
- `expand_parserfns` (boolean) - Normally, wikitext parser functions will be expanded. This can be set to `False` to prevent parser function expansion.
- `expand_invoke` (boolean) - Normally, the `#invoke` parser function (which calls a Lua module) will be expanded along with other parser functions. This can be set to `False` to prevent expansion of the `#invoke` parser function.

```
def start_page(self, title)
```



This function should be called before starting the processing of a new page or file. This saves the page title (which is frequently accessed by templates, parser functions, and Lua macros). The page title is also used in error messages.

The `wtp.process()` and `wtp.reprocess()` functions will automatically call this before calling the page handler for each page. This needs to be called manually when processing wikitext obtained from other sources.

The arguments are as follows:

- `title` (str) - The page title. For normal pages, there is usually no prefix. Templates typically have `Template: prefix` and Lua modules `Module: prefix`, and other prefixes are also used (e.g., `Thesaurus:`). This does not care about the form of the name, but some parser functions do.

```
def start_section(self, title)
```



Sets the title of the current section on the page. This is automatically reset to `None` by `wtp.start_page()`. The section title is only used in error, warning, and debug messages.

The arguments are:

- `title` (str) - the title of the section, or `None` to clear it.

```
def start_subsection(self, title)
```



Sets the title of the current subsection of the current section on the page. This is automatically reset to `None` by `wtp.start_page()` and `wtp.start_section()`. The subsection title is only used in error, warning, and debug messages.

The arguments are:

- `title` (str) - the title of the subsection, or `None` to clear it.

```
def add_page(self, title: str, namespace_id: int, body: Optional[str],
             redirect_to: Optional[str] = None, need_pre_expand: bool = False,
             model: str = "wikitext") -> None:
```



This function is used to add pages, templates, and modules for processing. There is usually no need to use this if `wtp.process()` is used; however, this can be used to add templates and pages for testing or other special processing needs.

The arguments are:

- `title` - the title of the page to be added (normal pages typically have no prefix in the title, templates begin with `Template:` , and Lua modules begin with `Module:`)
- `namespace_id` - namespace id
- `body` - the content of the page, template, or module
- `redirect_to` - title of redirect page
- `need_pre_expand` - set to `True` if the page is a template that need to be expanded before parsing.
- `model` - the model value for the page (usually `wikitext` for normal pages and templates and `Scribunto` for Lua modules)

The `wtp.analyze_templates()` function needs to be called after calling `wtp.add_page()` before pages can be expanded or parsed (it should preferably only be called once after adding all pages and templates).

```
def analyze_templates(self)
```



Analyzes the template definitions in the cache file and determines which of them should be pre-expanded before parsing because they affect the document structure significantly. Some templates in, e.g., Wiktionary expand to table start tags, table end tags, or list items, and parsing results are generally much better if they are expanded before parsing. The actual expansion only happens if `pre_expand` or some other argument to `wtp.expand()` or `wtp.parse()` tells them to do so.

The analysis is heuristic and is not guaranteed to find every such template. In particular, it cannot detect templates that call Lua modules that output Wikitext control structures (there are several templates in Wiktionary that call Lua code that outputs list items, for example). Such templates may need to be identified manually and specified as additional templates to expand. Luckily, there seem to be relatively few such templates, at least in Wiktionary.

This function is automatically called by `wtp.process()` at the end of phase 1. An explicit call is only necessary if `wtp.add_page()` has been used by the application.

Error handling

Various functions in this module, including `wtp.parse()` and `wtp.expand()` may generate errors and warnings. Those will be displayed on `stdout` as well as collected in `wtp.errors`, `wtp.warnings`, and `wtp.debugs`. These fields will contain lists of dictionaries, where each dictionary describes an error/warning/debug message. The dictionary can have the following keys (not all of them are always present):

- `msg` (str) - the error message
- `trace` (str or `None`) - optional stacktrace where the error occurred
- `title` (str) - the page title on which the error occurred
- `section` (str or `None`) - the section where the error occurred
- `subsection` (str or `None`) - the subsection where the error occurred
- `path` (tuple of str) - a path of title, template names, parser function names, or Lua module/function names, giving information about where the error occurred during expansion or parsing.

The fields containing the error messages will be cleared by every call to `wtp.start_page()` (including the implicit calls during `wtp.process()` and `wtp.reprocess()`). Thus, the `page_handler` function often returns these lists together with any information extracted from the page, and they can be collected together from the values returned by the iterators returned by these functions. The `wtp.to_return()` function maybe useful for this.

The following functions can be used for reporting errors. These can also be called by application code from within the `page_handler` function as well as `template_fn` and `post_template_fn` functions to report errors, warnings, and debug messages in a uniform way.

```
def error(self, msg, trace=None)
```



Reports an error message. The error will be added to `wtp.errors` list and printed to `stdout`. The arguments are:

- `msg` (str) - the error message (need not include page title or section)
- `trace` (str or `None`) - an optional stack trace giving more information about where the error occurred

```
def warning(self, msg, trace=None)
```



Reports a warning message. The warning will be added to `wtp.warnings` list and printed to stdout. The arguments are the same as for `wtp.error()` .

```
def debug(self, msg, trace=None)
```



Reports a debug message. The message will be added to `wtp.debugs` list and printed to stdout. The arguments are the same as for `wtp.error()` .

```
def to_return(self)
```



Produces a dictionary containing the error, warning, and debug messages from `wtp` . This would typically be called at the end of a `page_handler` function and the value returned along with whatever data was extracted from that page. The error lists are reset by `wtp.start_page()` (including the implicit calls from `wtp.process()` and `wtp.reprocess()`), so they should be saved (e.g., by this call) for each page. (Given the parallelism in the processing of the pages, they cannot just be accumulated in the subprocesses.)

The returned dictionary contains the following keys:

- `errors` - a list of dictionaries describing any error messages
- `warnings` - a list of dictionaries describing any warning messages
- `debugs` - a list of dictionaries describing any debug messages.

class WikiNode

The `WikiNode` class represents a parse tree node and is returned by `wtp.parse()` . This object can be printed or converted to a string and will display a human-readable format that is suitable for debugging purposes (at least for small parse trees).

The `WikiNode` objects have the following fields:

- `kind` (`NodeKind`, see below) - The type of the node. This determines how to interpret the other fields.
- `children` (list) - Contents of the node. This is generally used when the node has arbitrary size content, such as subsections, list items/sublists, other HTML tags, etc.
- `args` (list or str, depending on `kind`) - Direct arguments to the node. This is used, for example, for templates, template arguments, parser function arguments, and link arguments, in which case this is a list. For some node types (e.g., list, list item,

and HTML tag), this is directly a string.

- `attrs` - A dictionary containing HTML attributes or a definition list definition (under the `def` key).

class NodeKind(enum.Enum)

The `NodeKind` type is an enumerated value for parse tree (`WikiNode`) node types. Currently the following values are used (typically these need to be prefixed by `NodeKind.` , e.g., `NodeKind.LEVEL2`):

- `ROOT` - The root node of the parse tree.
- `LEVEL2` - Level 2 subtitle (`==`). The `args` field contains the title and `children` field contains any contents that are within this section
- `LEVEL3` - Level 3 subtitle (`===`)
- `LEVEL4` - Level 4 subtitle (`====`)
- `LEVEL5` - Level 5 subtitle (`=====`)
- `LEVEL6` - Level 6 subtitle (`=====`)
- `ITALIC` - Italic, content is in `children`
- `BOLD` - Bold, content is in `children`
- `HLINE` - A horizontal line (no arguments or children)
- `LIST` - Indicates a list. Each list and sublist will start with this kind of node. `args` will contain the prefix used to open the list (e.g., `"##"` - note this is stored directly as a string in `args`). List items will be stored in `children` .
- `LIST_ITEM` - A list item in the children of a `LIST` node. `args` is the prefix used to open the list item (same as for the `LIST` node). The contents of the list item (including any possible sublists) are in `children` . If the list is a definition list (i.e., the prefix ends in `;"`), then `children` contains the item label to be defined and `definition` contains the definition.
- `PREFORMATTED` - Preformatted text where markup is interpreted. Content is in `children` . This is used for lines starting with a space in wikitext.
- `PRE` - Preformatted text where markup is not interpreted. Content is in `children` . This is indicated in wikitext by `<pre>...</pre>`.
- `LINK` - An internal wikimedia link (`[[...]]` in wikitext). The link arguments are in `args` . This tag is also used for media inclusion. Links with a trailing word end immediately after the link have the trailing part in `children` .
- `TEMPLATE` - A template call (transclusion). Template name is in the first argument

and template arguments in subsequent arguments in `args`. The `children` field is not used. In wikitext templates are marked up as `{{name|arg1|arg2|...}}`.

- `TEMPLATE_ARG` - A template argument. The argument name is in the first item in `args` followed by any subsequent arguments (normally at most two items, but I've seen arguments with more - probably an error in those template definitions). The `children` field is not used. In wikitext template arguments are marked up as `{{{name|defval}}}`.
- `PARSER_FN` - A parser function invocation. This is also used for built-in variables such as `{{PAGENAME}}`. The parser function name is in the first element of `args` and parser function arguments in subsequent elements.
- `URL` - An external URL. The first argument is the URL. The second optional argument (in `args`) is the display text. The `children` field is not used.
- `TABLE` - A table. Content is in `children`. In wikitext, a table is encoded as `{| ... |}`.
- `TABLE_CAPTION` - A table caption. This can only occur under `TABLE`. The content is in `children`. The `attrs` field contains a dictionary of any HTML attributes given to the table.
- `TABLE_ROW` - A table row. This can only occur under `TABLE`. The content is in `children` (normally the content would be `TABLE_CELL` or `TABLE_HEADER_CELL` nodes). The `attrs` field contains a dictionary of any HTML attributes given to the table row.
- `TABLE_HEADER_CELL` - A table header cell. This can only occur under `TABLE_ROW`. Content is in `children`. The `attrs` field contains a dictionary of any HTML attributes given to the table row.
- `TABLE_CELL` - A table cell. This can only occur under `TABLE_ROW`. Content is in `children`. The `attrs` field contains a dictionary of any HTML attributes given to the table row.
- `MAGIC_WORD` - A MediaWiki magic word. The magic word is assigned directly to `args` as a string (i.e., not in a list). `children` is not used. An example of a magic word would be `__NOTOC__`.
- `HTML` - A HTML tag (or a matched pair of HTML tags). `args` is the name of the HTML tag directly (not in a list and always without a slash). `attrs` is set to a dictionary of any HTML attributes from the tag. The contents of the HTML tag is in `children`.

Expected performance

This can generally process a few Wiktionary pages per second per processor core, including expansion of all templates, Lua macros, parsing the full page, and analyzing the parse. On a multi-core machine, this can generally process a few dozen to a few hundred pages per second, depending on the speed and the number of the cores.

Most of the processing effort goes to expanding Lua macros. You can elect not to expand Lua macros, but they are used extensively in Wiktionary and for important information. Expanding templates and Lua macros allows much more robust and complete data extraction, but does not come cheap.

Contributing and bug reports

Please create an issue on github to report bugs or to contribute!