mwparserfromhell Documentation

Release 0.4.1

Ben Kurtovic

July 30, 2015

Contents

`mwparserfromhell` (the *MediaWiki Parser from Hell*) is a Python package that provides an easy-to-use and outrageously powerful parser for MediaWiki wikicode. It supports Python 2 and Python 3.

Developed by Earwig with contributions from Σ, Legoktm, and others. Development occurs on GitHub.

CHAPTER 1

Installation

The easiest way to install the parser is through the Python Package Index; you can install the latest release with `pip install mwparserfromhell` (get pip). On Windows, make sure you have the latest version of pip installed by running `pip install --upgrade pip`.

Alternatively, get the latest development version:

```
git clone https://github.com/earwig/mwparserfromhell.git
cd mwparserfromhell
python setup.py install
```

You can run the comprehensive unit testing suite with `python setup.py test -q`.

CHAPTER 2

Contents

2.1 Usage

Normal usage is rather straightforward (where `text` is page text):

```
>>> import mwparserfromhell
>>> wikicode = mwparserfromhell.parse(text)
```

`wikicode` is a *mwparserfromhell.Wikicode* object, which acts like an ordinary `str` object (or `unicode` in Python 2) with some extra methods. For example:

```
>>> text = "I has a template! {{foo|bar|baz|eggs=spam}} See it?"
>>> wikicode = mwparserfromhell.parse(text)
>>> print(wikicode)
I has a template! {{foo|bar|baz|eggs=spam}} See it?
>>> templates = wikicode.filter_templates()
>>> print(templates)
['{{foo|bar|baz|eggs=spam}}']
>>> template = templates[0]
>>> print(template.name)
foo
>>> print(template.params)
['bar', 'baz', 'eggs=spam']
>>> print(template.get(1).value)
bar
>>> print(template.get("eggs").value)
spam
```

Since nodes can contain other nodes, getting nested templates is trivial:

```
>>> text = "{{foo|{{bar}}={{baz|{{spam}}}}}}"
>>> mwparserfromhell.parse(text).filter_templates()
['{{foo|{{bar}}={{baz|{{spam}}}}}}', '{{bar}}', '{{baz|{{spam}}}}', '{{spam}}']
```

You can also pass *recursive=False* to *filter_templates()* and explore templates manually. This is possible because nodes can contain additional *Wikicode* objects:

```
>>> code = mwparserfromhell.parse("{{foo|this {{includes a|template}}}}")
>>> print(code.filter_templates(recursive=False))
['{{foo|this {{includes a|template}}}}']
>>> foo = code.filter_templates(recursive=False)[0]
>>> print(foo.get(1).value)
this {{includes a|template}}
```

```
>>> print(foo.get(1).value.filter_templates()[0])
{{includes a|template}}
>>> print(foo.get(1).value.filter_templates()[0].get(1).value)
template
```

Templates can be easily modified to add, remove, or alter params. *Wikicode* objects can be treated like lists, with *append()*, *insert()*, *remove()*, *replace()*, and more. They also have a *matches()* method for comparing page or template names, which takes care of capitalization and whitespace:

```
>>> text = "{{cleanup}} '''Foo''' is a [[bar]]. {{uncategorized}}"
>>> code = mwparserfromhell.parse(text)
>>> for template in code.filter_templates():
...     if template.name.matches("Cleanup") and not template.has("date"):
...         template.add("date", "July 2012")
...
>>> print(code)
{{cleanup|date=July 2012}} '''Foo''' is a [[bar]]. {{uncategorized}}
>>> code.replace("{{uncategorized}}", "{{bar-stub}}")
>>> print(code)
{{cleanup|date=July 2012}} '''Foo''' is a [[bar]]. {{bar-stub}}
>>> print(code.filter_templates())
['{{cleanup|date=July 2012}}', '{{bar-stub}}']
```

You can then convert `code` back into a regular `str` object (for saving the page!) by calling `str()` on it:

```
>>> text = str(code)
>>> print(text)
{{cleanup|date=July 2012}} '''Foo''' is a [[bar]]. {{bar-stub}}
>>> text == code
True
```

(Likewise, use `unicode(code)` in Python 2.)

For more tips, check out *Wikicode's full method list* and the *list of Nodes*.

2.2 Integration

`mwparserfromhell` is used by and originally developed for EarwigBot; `Page` objects have a `parse()` method that essentially calls `mwparserfromhell.parse()` on `get()`.

If you're using Pywikibot, your code might look like this:

```
import mwparserfromhell
import pywikibot

def parse(title):
    site = pywikibot.Site()
    page = pywikibot.Page(site, title)
    text = page.get()
    return mwparserfromhell.parse(text)
```

If you're not using a library, you can parse any page using the following code (via the API):

```
import json
from urllib.parse import urlencode
from urllib.request import urlopen
import mwparserfromhell
API_URL = "https://en.wikipedia.org/w/api.php"
```

```
def parse(title):
    data = {"action": "query", "prop": "revisions", "rvlimit": 1,
            "rvprop": "content", "format": "json", "titles": title}
    raw = urlopen(API_URL, urlencode(data).encode()).read()
    res = json.loads(raw)
    text = res["query"]["pages"].values()[0]["revisions"][0]["*"]
    return mwparserfromhell.parse(text)
```

2.3 Changelog

2.3.1 v0.4.1

Released July 30, 2015 (changes):

- The process for building Windows binaries has been fixed, and these should be distributed along with new releases. Windows users can now take advantage of C speedups without having a compiler of their own.
- Added support for Python 3.5.
- < and > are now disallowed in wikilink titles and template names. This includes when denoting tags, but not comments.
- Fixed the behavior of *preserve_spacing* in *Template.add()* and *keep_field* in *Template.remove()* on parameters with hidden keys.
- Removed _ListProxy.detach(). *SmartList*s now use weak references and their children are garbage-collected properly.
- Fixed parser bugs involving:
  - templates with completely blank names;
  - templates with newlines and comments.
- Heavy refactoring and fixes to the C tokenizer, including:
  - corrected a design flaw in text handling, allowing for substantial speed improvements when parsing long strings of plain text;
  - implemented new Python 3.3 PEP 393 Unicode APIs.
- Fixed various bugs in *SmartList*, including one that was causing memory issues on 64-bit builds of Python 2 on Windows.
- Fixed some bugs in the release scripts.

2.3.2 v0.4

Released May 23, 2015 (changes):

- The parser now falls back on pure Python mode if C extensions cannot be built. This fixes an issue that prevented some Windows users from installing the parser.
- Added support for parsing wikicode tables (patches by David Winegar).
- Added a script to test for memory leaks in `scripts/memtest.py`.
- Added a script to do releases in `scripts/release.sh`.

- *skip_style_tags* can now be passed to `mwparserfromhell.parse()` (previously, only `Parser.parse()` allowed it).
- The *recursive* argument to `Wikicode's filter()` methods now accepts a third option, `RECURSE_OTHERS`, which recurses over all children except instances of *forcetype* (for example, `code.filter_templates(code.RECURSE_OTHERS)` returns all un-nested templates).
- The parser now understands HTML tag attributes quoted with single quotes. When setting a tag attribute's value, quotes will be added if necessary. As part of this, `Attribute`'s `quoted` attribute has been changed to `quotes`, and is now either a string or `None`.
- Calling `Template.remove()` with a `Parameter` object that is not part of the template now raises `ValueError` instead of doing nothing.
- `Parameter`s with non-integer keys can no longer be created with *showkey=False*, nor have the value of this attribute be set to *False* later.
- `_ListProxy.destroy()` has been changed to `_ListProxy.detach()`, and now works in a more useful way.
- If something goes wrong while parsing, `ParserError` will now be raised. Previously, the parser would produce an unclear `BadRoute` exception or allow an incorrect node tree to be build.
- Fixed parser bugs involving:
  - nested tags;
  - comments in template names;
  - tags inside of `<nowiki>` tags.
- Added tests to ensure that parsed trees convert back to wikicode without unintentional modifications.
- Added support for a `NOWEB` environment variable, which disables a unit test that makes a web call.
- Test coverage has been improved, and some minor related bugs have been fixed.
- Updated and fixed some documentation.

2.3.3 v0.3.3

Released April 22, 2014 (changes):

- Added support for Python 2.6 and 3.4.
- `Template.has()` is now passed *ignore_empty=False* by default instead of *True*. This fixes a bug when adding parameters to templates with empty fields, **and is a breaking change if you rely on the default behavior.**
- The *matches* argument of `Wikicode's filter()` methods now accepts a function (taking one argument, a `Node`, and returning a bool) in addition to a regex.
- Re-added *flat* argument to `Wikicode.get_sections()`, fixed the order in which it returns sections, and made it faster.
- `Wikicode.matches()` now accepts a tuple or list of strings/`Wikicode` objects instead of just a single string or `Wikicode`.
- Given the frequency of issues with the (admittedly insufficient) tag parser, there's a temporary *skip_style_tags* argument to `parse()` that ignores `''` and `'''` until these issues are corrected.
- Fixed a parser bug involving nested wikilinks and external links.
- C code cleanup and speed improvements.

2.3.4 v0.3.2

Released September 1, 2013 (changes):

- Added support for Python 3.2 (along with current support for 3.3 and 2.7).
- Renamed *Template.remove()*'s first argument from *name* to *param*, which now accepts *Parameter* objects in addition to parameter name strings.

2.3.5 v0.3.1

Released August 29, 2013 (changes):

- Fixed a parser bug involving URLs nested inside other markup.
- Fixed some typos.

2.3.6 v0.3

Released August 24, 2013 (changes):

- Added complete support for HTML *Tags*, including forms like `<ref>foo</ref>`, `<ref name="bar"/>`, and wiki-markup tags like bold (`'''`), italics (`''`), and lists (`*`, `#`, `;` and `:`).
- Added support for *ExternalLink*s (`http://example.com/` and `[http://example.com/ Example]`).
- *Wikicode's filter()* methods are now passed *recursive=True* by default instead of *False*. **This is a breaking change if you rely on any filter() methods being non-recursive by default.**
- Added a *matches()* method to *Wikicode* for page/template name comparisons.
- The *obj* param of *Wikicode.insert_before()*, *insert_after()*, *replace()*, and *remove()* now accepts *Wikicode* objects and strings representing parts of wikitext, instead of just nodes. These methods also make all possible substitutions instead of just one.
- Renamed *Template.has_param()* to *has()* for consistency with *Template*'s other methods; *has_param()* is now an alias.
- The C tokenizer extension now works on Python 3 in addition to Python 2.7.
- Various bugfixes, internal changes, and cleanup.

2.3.7 v0.2

Released June 20, 2013 (changes):

- The parser now fully supports Python 3 in addition to Python 2.7.
- Added a C tokenizer extension that is significantly faster than its Python equivalent. It is enabled by default (if available) and can be toggled by setting `mwparserfromhell.parser.use_c` to a boolean value.
- Added a complete set of unit tests covering parsing and wikicode manipulation.
- Renamed `filter_links()` to *filter_wikilinks()* (applies to *ifilter()* as well).
- Added filter methods for *Arguments*, *Comments*, *Headings*, and *HTMLEntities*.
- Added *before* param to *Template.add()*; renamed *force_nonconformity* to *preserve_spacing*.
- Added *include_lead* param to *Wikicode.get_sections()*.

- Removed *flat* param from *get_sections()*.
- Removed *force_no_field* param from *Template.remove()*.
- Added support for Travis CI.
- Added note about Windows build issue in the README.
- The tokenizer will limit itself to a realistic recursion depth to prevent errors and unreasonably long parse times.
- Fixed how some nodes' attribute setters handle input.
- Fixed multiple bugs in the tokenizer's handling of invalid markup.
- Fixed bugs in the implementation of *SmartList* and *StringMixIn*.
- Fixed some broken example code in the README; other copyedits.
- Other bugfixes and code cleanup.

2.3.8 v0.1.1

Released September 21, 2012 (changes):

- Added support for *Comments* (`<!-- foo -->`) and *Wikilinks* (`[[foo]]`).
- Added corresponding `ifilter_links()` and `filter_links()` methods to *Wikicode*.
- Fixed a bug when parsing incomplete templates.
- Fixed *strip_code()* to affect the contents of headings.
- Various copyedits in documentation and comments.

2.3.9 v0.1

Released August 23, 2012:

- Initial release.

2.4 mwparserfromhell

2.4.1 mwparserfromhell Package

`mwparserfromhell` Package

mwparserfromhell (the MediaWiki Parser from Hell) is a Python package that provides an easy-to-use and outrageously powerful parser for MediaWiki wikicode.

`compat` Module

Implements support for both Python 2 and Python 3 by defining common types in terms of their Python 2/3 variants. For example, `str` is set to `unicode` on Python 2 but `str` on Python 3; likewise, `bytes` is `str` on 2 but `bytes` on 3. These types are meant to be imported directly from within the parser's modules.

definitions Module

Contains data about certain markup, like HTML tags and external links.

mwparserfromhell.definitions.**get_html_tag**(*markup*)
Return the HTML tag associated with the given wiki-markup.

mwparserfromhell.definitions.**is_parsable**(*tag*)
Return if the given *tag*'s contents should be passed to the parser.

mwparserfromhell.definitions.**is_visible**(*tag*)
Return whether or not the given *tag* contains visible text.

mwparserfromhell.definitions.**is_single**(*tag*)
Return whether or not the given *tag* can exist without a close tag.

mwparserfromhell.definitions.**is_single_only**(*tag*)
Return whether or not the given *tag* must exist without a close tag.

mwparserfromhell.definitions.**is_scheme**(*scheme*, *slashes=True*)
Return whether *scheme* is valid for external links.

smart_list Module

This module contains the *SmartList* type, as well as its *_ListProxy* child, which together implement a list whose sublists reflect changes made to the main list, and vice-versa.

class mwparserfromhell.smart_list.**SmartList**(*iterable=None*)
Bases: mwparserfromhell.smart_list._SliceNormalizerMixIn, list

Implements the list interface with special handling of sublists.

When a sublist is created (by list[i:j]), any changes made to this list (such as the addition, removal, or replacement of elements) will be reflected in the sublist, or vice-versa, to the greatest degree possible. This is implemented by having sublists - instances of the *_ListProxy* type - dynamically determine their elements by storing their slice info and retrieving that slice from the parent. Methods that change the size of the list also change the slice info. For example:

```
>>> parent = SmartList([0, 1, 2, 3])
>>> parent
[0, 1, 2, 3]
>>> child = parent[2:]
>>> child
[2, 3]
>>> child.append(4)
>>> child
[2, 3, 4]
>>> parent
[0, 1, 2, 3, 4]
```

append(*item*)
L.append(object) – append object to end

extend(*item*)
L.extend(iterable) – extend list by appending elements from the iterable

insert(*index*, *item*)
L.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

remove(*item*)
L.remove(value) – remove first occurrence of value. Raises ValueError if the value is not present.

reverse()
L.reverse() – reverse *IN PLACE*

sort(*cmp=None*, *key=None*, *reverse=None*)
L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

class mwparserfromhell.smart_list.**_ListProxy**(*parent*, *sliceinfo*)
Bases: mwparserfromhell.smart_list._SliceNormalizerMixIn, list

Implement the list interface by getting elements from a parent.

This is created by a *SmartList* object when slicing. It does not actually store the list at any time; instead, whenever the list is needed, it builds it dynamically using the _render() method.

append(*item*)
L.append(object) – append object to end

count(*value*) → integer – return number of occurrences of value

extend(*item*)
L.extend(iterable) – extend list by appending elements from the iterable

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert(*index*, *item*)
L.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

remove(*item*)
L.remove(value) – remove first occurrence of value. Raises ValueError if the value is not present.

reverse()
L.reverse() – reverse *IN PLACE*

sort(*cmp=None*, *key=None*, *reverse=None*)
L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

string_mixin Module

This module contains the *StringMixIn* type, which implements the interface for the unicode type (str on py3k) in a dynamic manner.

class mwparserfromhell.string_mixin.**StringMixIn**
Implement the interface for unicode/str in a dynamic manner.

To use this class, inherit from it and override the __unicode__() method (same on py3k) to return the string representation of the object. The various string methods will operate on the value of __unicode__() instead of the immutable self like the regular str type.

utils Module

This module contains accessory functions for other parts of the library. Parser users generally won't need stuff from here.

mwparserfromhell.utils.**parse_anything**(*value*, *context=0*, *skip_style_tags=False*)
Return a *Wikicode* for *value*, allowing multiple types.

This differs from *Parser.parse()* in that we accept more than just a string to be parsed. Unicode objects (strings in py3k), strings (bytes in py3k), integers (converted to strings), None, existing *Node* or *Wikicode* objects, as well as an iterable of these types, are supported. This is used to parse input on-the-fly by various methods of *Wikicode* and others like *Template*, such as *wikicode.insert()* or setting template.name.

Additional arguments are passed directly to *Parser.parse()*.

wikicode Module

class mwparserfromhell.wikicode.**Wikicode**(*nodes*)
Bases: *mwparserfromhell.string_mixin.StringMixIn*

A Wikicode is a container for nodes that operates like a string.

Additionally, it contains methods that can be used to extract data from or modify the nodes, implemented in an interface similar to a list. For example, *index()* can get the index of a node in the list, and *insert()* can add a new node at that index. The *filter()* series of functions is very useful for extracting and iterating over, for example, all of the templates in the object.

RECURSE_OTHERS = 2

append(*value*)
Insert *value* at the end of the list of nodes.

value can be anything parsable by *parse_anything()*.

filter(**args*, ***kwargs*)
Return a list of nodes within our list matching certain conditions.

This is equivalent to calling list() on *ifilter()*.

filter_arguments(**a*, ***kw*)
Iterate over arguments.

This is equivalent to *filter()* with *forcetype* set to *Argument*.

filter_comments(**a*, ***kw*)
Iterate over comments.

This is equivalent to *filter()* with *forcetype* set to *Comment*.

filter_external_links(**a*, ***kw*)
Iterate over external_links.

This is equivalent to *filter()* with *forcetype* set to *ExternalLink*.

filter_headings(**a*, ***kw*)
Iterate over headings.

This is equivalent to *filter()* with *forcetype* set to *Heading*.

filter_html_entities(**a*, ***kw*)
Iterate over html_entities.

This is equivalent to *filter()* with *forcetype* set to *HTMLEntity*.

filter_tags(**a*, ***kw*)
Iterate over tags.

This is equivalent to *filter()* with *forcetype* set to *Tag*.

filter_templates(**a*, ***kw*)
Iterate over templates.

This is equivalent to *filter()* with *forcetype* set to *Template*.

filter_text(**a*, ***kw*)
Iterate over text.

This is equivalent to *filter()* with *forcetype* set to *Text*.

filter_wikilinks(**a*, ***kw*)
Iterate over wikilinks.

This is equivalent to *filter()* with *forcetype* set to *Wikilink*.

get(*index*)
Return the *index*th node within the list of nodes.

get_sections(*levels=None*, *matches=None*, *flags=50*, *flat=False*, *include_lead=None*, *include_headings=True*)
Return a list of sections within the page.

Sections are returned as *Wikicode* objects with a shared node list (implemented using *SmartList*) so that changes to sections are reflected in the parent Wikicode object.

Each section contains all of its subsections, unless *flat* is `True`. If *levels* is given, it should be a iterable of integers; only sections whose heading levels are within it will be returned. If *matches* is given, it should be either a function or a regex; only sections whose headings match it (without the surrounding equal signs) will be included. *flags* can be used to override the default regex flags (see *ifilter()*) if a regex *matches* is used.

If *include_lead* is `True`, the first, lead section (without a heading) will be included in the list; `False` will not include it; the default will include it only if no specific *levels* were given. If *include_headings* is `True`, the section's beginning *Heading* object will be included; otherwise, this is skipped.

get_tree()
Return a hierarchical tree representation of the object.

The representation is a string makes the most sense printed. It is built by calling `_get_tree()` on the *Wikicode* object and its children recursively. The end result may look something like the following:

```
>>> text = "Lorem ipsum {{foo|bar|{{baz}}|spam=eggs}}"
>>> print(mwparserfromhell.parse(text).get_tree())
Lorem ipsum
{{
      foo
    | 1
    = bar
    | 2
    = {{
            baz
      }}
    | spam
    = eggs
}}
```

ifilter(*recursive=True*, *matches=None*, *flags=50*, *forcetype=None*)
Iterate over nodes in our list matching certain conditions.

If *forcetype* is given, only nodes that are instances of this type (or tuple of types) are yielded. Setting *recursive* to `True` will iterate over all children and their descendants. `RECURSE_OTHERS` will only

iterate over children that are not the instances of *forcetype*. `False` will only iterate over immediate children.

`RECURSE_OTHERS` can be used to iterate over all un-nested templates, even if they are inside of HTML tags, like so:

```
>>> code = mwparserfromhell.parse("{{foo}}<b>{{foo|{{bar}}}}</b>")
>>> code.filter_templates(code.RECURSE_OTHERS)
["{{foo}}", "{{foo|{{bar}}}}"]
```

matches can be used to further restrict the nodes, either as a function (taking a single *Node* and returning a boolean) or a regular expression (matched against the node's string representation with `re.search()`). If *matches* is a regex, the flags passed to `re.search()` are `re.IGNORECASE`, `re.DOTALL`, and `re.UNICODE`, but custom flags can be specified by passing *flags*.

ifilter_arguments(**a*, ***kw*)
Iterate over arguments.

This is equivalent to *ifilter()* with *forcetype* set to *Argument*.

ifilter_comments(**a*, ***kw*)
Iterate over comments.

This is equivalent to *ifilter()* with *forcetype* set to *Comment*.

ifilter_external_links(**a*, ***kw*)
Iterate over external_links.

This is equivalent to *ifilter()* with *forcetype* set to *ExternalLink*.

ifilter_headings(**a*, ***kw*)
Iterate over headings.

This is equivalent to *ifilter()* with *forcetype* set to *Heading*.

ifilter_html_entities(**a*, ***kw*)
Iterate over html_entities.

This is equivalent to *ifilter()* with *forcetype* set to *HTMLEntity*.

ifilter_tags(**a*, ***kw*)
Iterate over tags.

This is equivalent to *ifilter()* with *forcetype* set to *Tag*.

ifilter_templates(**a*, ***kw*)
Iterate over templates.

This is equivalent to *ifilter()* with *forcetype* set to *Template*.

ifilter_text(**a*, ***kw*)
Iterate over text.

This is equivalent to *ifilter()* with *forcetype* set to *Text*.

ifilter_wikilinks(**a*, ***kw*)
Iterate over wikilinks.

This is equivalent to *ifilter()* with *forcetype* set to *Wikilink*.

index(*obj*, *recursive=False*)
Return the index of *obj* in the list of nodes.

Raises `ValueError` if *obj* is not found. If *recursive* is `True`, we will look in all nodes of ours and their descendants, and return the index of our direct descendant node within *our* list of nodes. Otherwise, the lookup is done only on direct descendants.

insert(*index*, *value*)
Insert *value* at *index* in the list of nodes.

value can be anything parsable by *parse_anything()*, which includes strings or other *Wikicode* or *Node* objects.

insert_after(*obj*, *value*, *recursive=True*)
Insert *value* immediately after *obj*.

obj can be either a string, a *Node*, or another *Wikicode* object (as created by *get_sections()*, for example). If *obj* is a string, we will operate on all instances of that string within the code, otherwise only on the specific instance given. *value* can be anything parsable by *parse_anything()*. If *recursive* is `True`, we will try to find *obj* within our child nodes even if it is not a direct descendant of this *Wikicode* object. If *obj* is not found, `ValueError` is raised.

insert_before(*obj*, *value*, *recursive=True*)
Insert *value* immediately before *obj*.

obj can be either a string, a *Node*, or another *Wikicode* object (as created by *get_sections()*, for example). If *obj* is a string, we will operate on all instances of that string within the code, otherwise only on the specific instance given. *value* can be anything parsable by *parse_anything()*. If *recursive* is `True`, we will try to find *obj* within our child nodes even if it is not a direct descendant of this *Wikicode* object. If *obj* is not found, `ValueError` is raised.

matches(*other*)
Do a loose equivalency test suitable for comparing page names.

other can be any string-like object, including *Wikicode*, or a tuple of these. This operation is symmetric; both sides are adjusted. Specifically, whitespace and markup is stripped and the first letter's case is normalized. Typical usage is `if template.name.matches("stub"): ...`.

nodes
A list of *Node* objects.

This is the internal data actually stored within a *Wikicode* object.

remove(*obj*, *recursive=True*)
Remove *obj* from the list of nodes.

obj can be either a string, a *Node*, or another *Wikicode* object (as created by *get_sections()*, for example). If *obj* is a string, we will operate on all instances of that string within the code, otherwise only on the specific instance given. If *recursive* is `True`, we will try to find *obj* within our child nodes even if it is not a direct descendant of this *Wikicode* object. If *obj* is not found, `ValueError` is raised.

replace(*obj*, *value*, *recursive=True*)
Replace *obj* with *value*.

obj can be either a string, a *Node*, or another *Wikicode* object (as created by *get_sections()*, for example). If *obj* is a string, we will operate on all instances of that string within the code, otherwise only on the specific instance given. *value* can be anything parsable by *parse_anything()*. If *recursive* is `True`, we will try to find *obj* within our child nodes even if it is not a direct descendant of this *Wikicode* object. If *obj* is not found, `ValueError` is raised.

set(*index*, *value*)
Set the `Node` at *index* to *value*.

Raises `IndexError` if *index* is out of range, or `ValueError` if *value* cannot be coerced into one *Node*. To insert multiple nodes at an index, use *get()* with either *remove()* and *insert()* or *replace()*.

strip_code(*normalize=True*, *collapse=True*)
Return a rendered string without unprintable code such as templates.

The way a node is stripped is handled by the `__strip__()` method of *Node* objects, which generally return a subset of their nodes or `None`. For example, templates and tags are removed completely, links are stripped to just their display part, headings are stripped to just their title. If *normalize* is `True`, various things may be done to strip code further, such as converting HTML entities like `&Sigma;`, `&#931;`, and `&#x3a3;` to Σ. If *collapse* is `True`, we will try to remove excess whitespace as well (three or more newlines are converted to two, for example).

Subpackages

nodes Package

nodes Package This package contains *Wikicode* "nodes", which represent a single unit of wikitext, such as a Template, an HTML tag, a Heading, or plain text. The node "tree" is far from flat, as most types can contain additional *Wikicode* types within them - and with that, more nodes. For example, the name of a *Template* is a *Wikicode* object that can contain text or more templates.

class mwparserfromhell.nodes.**Node**
Represents the base Node type, demonstrating the methods to override.

`__unicode__()` must be overridden. It should return a `unicode` or (`str` in py3k) representation of the node. If the node contains *Wikicode* objects inside of it, `__children__()` should be a generator that iterates over them. If the node is printable (shown when the page is rendered), `__strip__()` should return its printable version, stripping out any formatting marks. It does not have to return a string, but something that can be converted to a string with `str()`. Finally, `__showtree__()` can be overridden to build a nice tree representation of the node, if desired, for *get_tree()*.

argument Module
class mwparserfromhell.nodes.argument.**Argument**(*name*, *default=None*)
Bases: *mwparserfromhell.nodes.Node*

Represents a template argument substitution, like `{{{foo}}}`.

default
The default value to substitute if none is passed.

This will be `None` if the argument wasn't defined with one. The MediaWiki parser handles this by rendering the argument itself in the result, complete braces. To have the argument render as nothing, set default to `""` (`{{{arg}}}` vs. `{{{arg|}}}`).

name
The name of the argument to substitute.

comment Module
class mwparserfromhell.nodes.comment.**Comment**(*contents*)
Bases: *mwparserfromhell.nodes.Node*

Represents a hidden HTML comment, like `<!-- foobar -->`.

contents
The hidden text contained between `<!--` and `-->`.

external_link Module

class mwparserfromhell.nodes.external_link.**ExternalLink**(*url*, *title=None*, *brackets=True*)

Bases: *mwparserfromhell.nodes.Node*

Represents an external link, like `[http://example.com/ Example]`.

brackets
Whether to enclose the URL in brackets or display it straight.

title
The link title (if given), as a *Wikicode* object.

url
The URL of the link target, as a *Wikicode* object.

heading Module

class mwparserfromhell.nodes.heading.**Heading**(*title*, *level*)

Bases: *mwparserfromhell.nodes.Node*

Represents a section heading in wikicode, like `== Foo ==`.

level
The heading level, as an integer between 1 and 6, inclusive.

title
The title of the heading, as a *Wikicode* object.

html_entity Module

class mwparserfromhell.nodes.html_entity.**HTMLEntity**(*value*, *named=None*, *hexadecimal=False*, *hex_char=u'x'*)

Bases: *mwparserfromhell.nodes.Node*

Represents an HTML entity, like ` `, either named or unnamed.

hex_char
If the value is hexadecimal, this is the letter denoting that.

For example, the hex_char of `"&#x1234;"` is `"x"`, whereas the hex_char of `"&#X1234;"` is `"X"`. Lowercase and uppercase `x` are the only values supported.

hexadecimal
If unnamed, this is whether the value is hexadecimal or decimal.

named
Whether the entity is a string name for a codepoint or an integer.

For example, `&Sigma;`, `&#931;`, and `&#x3a3;` refer to the same character, but only the first is "named", while the others are integer representations of the codepoint.

normalize()
Return the unicode character represented by the HTML entity.

value
The string value of the HTML entity.

tag Module

class mwparserfromhell.nodes.tag.**Tag**(*tag*, *contents=None*, *attrs=None*, *wiki_markup=None*, *self_closing=False*, *invalid=False*, *implicit=False*, *padding=u''*, *closing_tag=None*, *wiki_style_separator=None*, *closing_wiki_markup=None*)

Bases: *mwparserfromhell.nodes.Node*

Represents an HTML-style tag in wikicode, like <ref>.

add(*name*, *value=None*, *quotes=u""*, *pad_first=u' '*, *pad_before_eq=u''*, *pad_after_eq=u''*)

Add an attribute with the given *name* and *value*.

name and *value* can be anything parsable by *utils.parse_anything()*; *value* can be omitted if the attribute is valueless. If *quotes* is not None, it should be a string (either " or ') that *value* will be wrapped in (this is recommended). None is only legal if *value* contains no spacing.

pad_first, *pad_before_eq*, and *pad_after_eq* are whitespace used as padding before the name, before the equal sign (or after the name if no value), and after the equal sign (ignored if no value), respectively.

attributes

The list of attributes affecting the tag.

Each attribute is an instance of *Attribute*.

closing_tag

The closing tag, as a *Wikicode* object.

This will usually equal *tag*, unless there is additional spacing, comments, or the like.

closing_wiki_markup

The wikified version of the closing tag to show instead of HTML.

If set to a value, this will be displayed instead of the close tag brackets. If tag is *self_closing* is True then this is not displayed. If *wiki_markup* is set and this has not been set, this is set to the value of *wiki_markup*. If this has been set and *wiki_markup* is set to a False value, this is set to None.

contents

The contents of the tag, as a *Wikicode* object.

get(*name*)

Get the attribute with the given *name*.

The returned object is a *Attribute* instance. Raises ValueError if no attribute has this name. Since multiple attributes can have the same name, we'll return the last match, since all but the last are ignored by the MediaWiki parser.

has(*name*)

Return whether any attribute in the tag has the given *name*.

Note that a tag may have multiple attributes with the same name, but only the last one is read by the MediaWiki parser.

implicit

Whether the tag is implicitly self-closing, with no ending slash.

This is only possible for specific "single" tags like <br> and <li>. See *definitions.is_single()*. This field only has an effect if *self_closing* is also True.

invalid

Whether the tag starts with a backslash after the opening bracket.

This makes the tag look like a lone close tag. It is technically invalid and is only parsable Wikicode when the tag itself is single-only, like <br> and <img>. See *definitions.is_single_only()*.

padding
Spacing to insert before the first closing >.

remove(*name*)
Remove all attributes with the given *name*.

self_closing
Whether the tag is self-closing with no content (like <br/>).

tag
The tag itself, as a *Wikicode* object.

wiki_markup
The wikified version of a tag to show instead of HTML.

If set to a value, this will be displayed instead of the brackets. For example, set to '' to replace <i> or ---- to replace <hr>.

wiki_style_separator
The separator between the padding and content in a wiki markup tag.

Essentially the wiki equivalent of the TagCloseOpen.

template Module

class mwparserfromhell.nodes.template.**Template**(*name*, *params=None*)
Bases: *mwparserfromhell.nodes.Node*

Represents a template in wikicode, like {{foo}}.

add(*name*, *value*, *showkey=None*, *before=None*, *preserve_spacing=True*)
Add a parameter to the template with a given *name* and *value*.

name and *value* can be anything parsable by *utils.parse_anything()*; pipes and equal signs are automatically escaped from *value* when appropriate.

If *name* is already a parameter in the template, we'll replace its value.

If *showkey* is given, this will determine whether or not to show the parameter's name (e.g., {{foo|bar}}'s parameter has a name of "1" but it is hidden); otherwise, we'll make a safe and intelligent guess.

If *before* is given (either a *Parameter* object or a name), then we will place the parameter immediately before this one. Otherwise, it will be added at the end. If *before* is a name and exists multiple times in the template, we will place it before the last occurrence. If *before* is not in the template, ValueError is raised. The argument is ignored if *name* is an existing parameter.

If *preserve_spacing* is True, we will try to preserve whitespace conventions around the parameter, whether it is new or we are updating an existing value. It is disabled for parameters with hidden keys, since MediaWiki doesn't strip whitespace in this case.

get(*name*)
Get the parameter whose name is *name*.

The returned object is a *Parameter* instance. Raises ValueError if no parameter has this name. Since multiple parameters can have the same name, we'll return the last match, since the last parameter is the only one read by the MediaWiki parser.

has(*name*, *ignore_empty=False*)
Return True if any parameter in the template is named *name*.

With *ignore_empty*, `False` will be returned even if the template contains a parameter with the name *name*, if the parameter's value is empty. Note that a template may have multiple parameters with the same name, but only the last one is read by the MediaWiki parser.

has_param(*name*, *ignore_empty=False*)
Alias for *has()*.

name
The name of the template, as a *Wikicode* object.

params
The list of parameters contained within the template.

remove(*param*, *keep_field=False*)
Remove a parameter from the template, identified by *param*.

If *param* is a *Parameter* object, it will be matched exactly, otherwise it will be treated like the *name* argument to *has()* and *get()*.

If *keep_field* is `True`, we will keep the parameter's name, but blank its value. Otherwise, we will remove the parameter completely.

When removing a parameter with a hidden name, subsequent parameters with hidden names will be made visible. For example, removing `bar` from `{{foo|bar|baz}}` produces `{{foo|2=baz}}` because `{{foo|baz}}` is incorrect.

If the parameter shows up multiple times in the template and *param* is not a *Parameter* object, we will remove all instances of it (and keep only one if *keep_field* is `True` - either the one with a hidden name, if it exists, or the first instance).

text Module

class mwparserfromhell.nodes.text.**Text**(*value*)
Bases: *mwparserfromhell.nodes.Node*

Represents ordinary, unformatted text with no special properties.

value
The actual text itself.

wikilink Module

class mwparserfromhell.nodes.wikilink.**Wikilink**(*title*, *text=None*)
Bases: *mwparserfromhell.nodes.Node*

Represents an internal wikilink, like `[[Foo|Bar]]`.

text
The text to display (if any), as a *Wikicode* object.

title
The title of the linked page, as a *Wikicode* object.

Subpackages

extras Package

extras Package This package contains objects used by *Node*s, but that are not nodes themselves. This includes template parameters and HTML tag attributes.

attribute Module

class mwparserfromhell.nodes.extras.attribute.**Attribute**(*name*, *value=None*, *quotes=u""*, *pad_first=u' '*, *pad_before_eq=u''*, *pad_after_eq=u''*, *check_quotes=True*)

Bases: *mwparserfromhell.string_mixin.StringMixIn*

Represents an attribute of an HTML tag.

This is used by *Tag* objects. For example, the tag `<ref name="foo">` contains an Attribute whose name is `"name"` and whose value is `"foo"`.

static coerce_quotes(*quotes*)
Coerce a quote type into an acceptable value, or raise an error.

name
The name of the attribute as a *Wikicode* object.

pad_after_eq
Spacing to insert right after the equal sign.

pad_before_eq
Spacing to insert right before the equal sign.

pad_first
Spacing to insert right before the attribute.

quotes
How to enclose the attribute value. `"`, `'`, or `None`.

value
The value of the attribute as a *Wikicode* object.

parameter Module

class mwparserfromhell.nodes.extras.parameter.**Parameter**(*name*, *value*, *showkey=True*)

Bases: *mwparserfromhell.string_mixin.StringMixIn*

Represents a paramater of a template.

For example, the template `{{foo|bar|spam=eggs}}` contains two Parameters: one whose name is `"1"`, value is `"bar"`, and `showkey` is `False`, and one whose name is `"spam"`, value is `"eggs"`, and `showkey` is `True`.

static can_hide_key(*key*)
Return whether or not the given key can be hidden.

name
The name of the parameter as a *Wikicode* object.

showkey
Whether to show the parameter's key (i.e., its "name").

value
The value of the parameter as a *Wikicode* object.

parser Package

parser Package This package contains the actual wikicode parser, split up into two main modules: the *tokenizer* and the *builder*. This module joins them together into one interface.

class mwparserfromhell.parser.**Parser**
Represents a parser for wikicode.

Actual parsing is a two-step process: first, the text is split up into a series of tokens by the *Tokenizer*, and then the tokens are converted into trees of *Wikicode* objects and *Node*s by the *Builder*.

Instances of this class or its dependents (*Tokenizer* and *Builder*) should not be shared between threads. *parse()* can be called multiple times as long as it is not done concurrently. In general, there is no need to do this because parsing should be done through mwparserfromhell.parse(), which creates a new *Parser* object as necessary.

parse(*text*, *context=0*, *skip_style_tags=False*)
Parse *text*, returning a *Wikicode* object tree.

If given, *context* will be passed as a starting context to the parser. This is helpful when this function is used inside node attribute setters. For example, *ExternalLink*'s *url* setter sets *context* to *contexts.EXT_LINK_URI* to prevent the URL itself from becoming an *ExternalLink*.

If *skip_style_tags* is True, then '' and ''' will not be parsed, but instead will be treated as plain text.

If there is an internal error while parsing, *ParserError* will be raised.

exception mwparserfromhell.parser.**ParserError**(*extra*)
Exception raised when an internal error occurs while parsing.

This does not mean that the wikicode was invalid, because invalid markup should still be parsed correctly. This means that the parser caught itself with an impossible internal state and is bailing out before other problems can happen. Its appearance indicates a bug.

builder Module

class mwparserfromhell.parser.builder.**Builder**
Builds a tree of nodes out of a sequence of tokens.

To use, pass a list of *Token*s to the *build()* method. The list will be exhausted as it is parsed and a *Wikicode* object containing the node tree will be returned.

_handle_argument(*token*)
Handle a case where an argument is at the head of the tokens.

_handle_attribute(*start*)
Handle a case where a tag attribute is at the head of the tokens.

_handle_comment(*token*)
Handle a case where an HTML comment is at the head of the tokens.

_handle_entity(*token*)
Handle a case where an HTML entity is at the head of the tokens.

_handle_external_link(*token*)
Handle when an external link is at the head of the tokens.

_handle_heading(*token*)
Handle a case where a heading is at the head of the tokens.

_handle_parameter(*default*)
Handle a case where a parameter is at the head of the tokens.

default is the value to use if no parameter name is defined.

_handle_tag(*token*)
Handle a case where a tag is at the head of the tokens.

_handle_template(*token*)
Handle a case where a template is at the head of the tokens.

_handle_token(*token*)
Handle a single token.

_handle_wikilink(*token*)
Handle a case where a wikilink is at the head of the tokens.

_pop()
Pop the current node list off of the stack.

The raw node list is wrapped in a *SmartList* and then in a *Wikicode* object.

_push()
Push a new node list onto the stack.

_write(*item*)
Append a node to the current node list.

build(*tokenlist*)
Build a Wikicode object from a list tokens and return it.

contexts Module This module contains various "context" definitions, which are essentially flags set during the tokenization process, either on the current parse stack (local contexts) or affecting all stacks (global contexts). They represent the context the tokenizer is in, such as inside a template's name definition, or inside a level-two heading. This is used to determine what tokens are valid at the current point and also if the current parsing route is invalid.

The tokenizer stores context as an integer, with these definitions bitwise OR'd to set them, AND'd to check if they're set, and XOR'd to unset them. The advantage of this is that contexts can have sub-contexts (as `FOO == 0b11` will cover `BAR == 0b10` and `BAZ == 0b01`).

Local (stack-specific) contexts:

- `TEMPLATE`
  - `TEMPLATE_NAME`
  - `TEMPLATE_PARAM_KEY`
  - `TEMPLATE_PARAM_VALUE`
- `ARGUMENT`
  - `ARGUMENT_NAME`
  - `ARGUMENT_DEFAULT`
- `WIKILINK`
  - `WIKILINK_TITLE`
  - `WIKILINK_TEXT`
- `EXT_LINK`
  - `EXT_LINK_URI`
  - `EXT_LINK_TITLE`
- `HEADING`
  - `HEADING_LEVEL_1`
  - `HEADING_LEVEL_2`

    - HEADING_LEVEL_3
    - HEADING_LEVEL_4
    - HEADING_LEVEL_5
    - HEADING_LEVEL_6
- TAG
    - TAG_OPEN
    - TAG_ATTR
    - TAG_BODY
    - TAG_CLOSE
- STYLE
    - STYLE_ITALICS
    - STYLE_BOLD
    - STYLE_PASS_AGAIN
    - STYLE_SECOND_PASS
- DL_TERM
- SAFETY_CHECK
    - HAS_TEXT
    - FAIL_ON_TEXT
    - FAIL_NEXT
    - FAIL_ON_LBRACE
    - FAIL_ON_RBRACE
    - FAIL_ON_EQUALS
    - HAS_TEMPLATE
- TABLE
    - TABLE_OPEN
    - TABLE_CELL_OPEN
    - TABLE_CELL_STYLE
    - TABLE_TD_LINE
    - TABLE_TH_LINE
    - TABLE_CELL_LINE_CONTEXTS

Global contexts:

- GL_HEADING

Aggregate contexts:

- FAIL
- UNSAFE
- DOUBLE

- NO_WIKILINKS
- NO_EXT_LINKS

tokenizer Module

class mwparserfromhell.parser.tokenizer.**Tokenizer**

Creates a list of tokens from a string of wikicode.

END = <object object>

MARKERS = [u'{', u'}', u'[', u']', u'<', u'>', u'|', u'=', u'&', u'"', u'#', u'*', u';', u':', u'/', u'-', u'!', u'\n', <object object a

MAX_CYCLES = 100000

MAX_DEPTH = 40

START = <object object>

USES_C = False

_can_recurse()
Return whether or not our max recursion depth has been exceeded.

_context
The current token context.

_emit(*token*)
Write a token to the end of the current token stack.

_emit_all(*tokenlist*)
Write a series of tokens to the current stack at once.

_emit_first(*token*)
Write a token to the beginning of the current token stack.

_emit_style_tag(*tag*, *markup*, *body*)
Write the body of a tag and the tokens that should surround it.

_emit_table_tag(*open_open_markup*, *tag*, *style*, *padding*, *close_open_markup*, *contents*, *open_close_markup*)
Emit a table tag.

_emit_text(*text*)
Write text to the current textbuffer.

_emit_text_then_stack(*text*)
Pop the current stack, write *text*, and then write the stack.

_fail_route()
Fail the current tokenization route.

Discards the current stack/context/textbuffer and raises *BadRoute*.

_handle_argument_end()
Handle the end of an argument at the head of the string.

_handle_argument_separator()
Handle the separator between an argument's name and default.

_handle_blacklisted_tag()
Handle the body of an HTML tag that is parser-blacklisted.

_handle_dl_term()
Handle the term in a description list (`foo` in `;foo:bar`).

_handle_end()
Handle the end of the stream of wikitext.

_handle_free_link_text(*punct*, *tail*, *this*)
Handle text in a free ext link, including trailing punctuation.

_handle_heading_end()
Handle the end of a section heading at the head of the string.

_handle_hr()
Handle a wiki-style horizontal rule (----) in the string.

_handle_invalid_tag_start()
Handle the (possible) start of an implicitly closing single tag.

_handle_list()
Handle a wiki-style list (#, *, ;, :).

_handle_list_marker()
Handle a list marker at the head (#, *, ;, :).

_handle_single_only_tag_end()
Handle the end of an implicitly closing single-only HTML tag.

_handle_single_tag_end()
Handle the stream end when inside a single-supporting HTML tag.

_handle_table_cell(*markup*, *tag*, *line_context*)
Parse as normal syntax unless we hit a style marker, then parse style as HTML attributes and the remainder as normal syntax.

_handle_table_cell_end(*reset_for_style=False*)
Returns the current context, with the TABLE_CELL_STYLE flag set if it is necessary to reset and parse style attributes.

_handle_table_end()
Return the stack in order to handle the table end.

_handle_table_row()
Parse as style until end of the line, then continue.

_handle_table_row_end()
Return the stack in order to handle the table row end.

_handle_table_style(*end_token*)
Handle style attributes for a table until `end_token`.

_handle_tag_close_close()
Handle the ending of a closing tag (</foo>).

_handle_tag_close_open(*data*, *token*)
Handle the closing of a open tag (<foo>).

_handle_tag_data(*data*, *text*)
Handle all sorts of *text* data inside of an HTML open tag.

_handle_tag_open_close()
Handle the opening of a closing tag (</foo>).

_handle_tag_space(*data*, *text*)
Handle whitespace (*text*) inside of an HTML open tag.

_handle_tag_text(*text*)
Handle regular *text* inside of an HTML open tag.

_handle_template_end()
Handle the end of a template at the head of the string.

_handle_template_param()
Handle a template parameter at the head of the string.

_handle_template_param_value()
Handle a template parameter's value at the head of the string.

_handle_wikilink_end()
Handle the end of a wikilink at the head of the string.

_handle_wikilink_separator()
Handle the separator between a wikilink's title and its text.

_is_free_link_end(*this*, *next*)
Return whether the current head is the end of a free link.

_parse(*context=0*, *push=True*)
Parse the wikicode string, using *context* for when to stop.

_parse_argument()
Parse an argument at the head of the wikicode string.

_parse_bold()
Parse wiki-style bold.

_parse_bracketed_uri_scheme()
Parse the URI scheme of a bracket-enclosed external link.

_parse_comment()
Parse an HTML comment at the head of the wikicode string.

_parse_entity()
Parse an HTML entity at the head of the wikicode string.

_parse_external_link(*brackets*)
Parse an external link at the head of the wikicode string.

_parse_free_uri_scheme()
Parse the URI scheme of a free (no brackets) external link.

_parse_heading()
Parse a section heading at the head of the wikicode string.

_parse_italics()
Parse wiki-style italics.

_parse_italics_and_bold()
Parse wiki-style italics and bold together (i.e., five ticks).

_parse_style()
Parse wiki-style formatting (`''`/`'''` for italics/bold).

_parse_table()
Parse a wikicode table by starting with the first line.

_parse_tag()
Parse an HTML tag at the head of the wikicode string.

_parse_template(*has_content*)
Parse a template at the head of the wikicode string.

_parse_template_or_argument()
Parse a template or argument at the head of the wikicode string.

_parse_wikilink()
Parse an internal wikilink at the head of the wikicode string.

_pop(*keep_context=False*)
Pop the current stack/context/textbuffer, returning the stack.

If *keep_context* is `True`, then we will replace the underlying stack's context with the current stack's.

_push(*context=0*)
Add a new token stack, context, and textbuffer to the list.

_push_tag_buffer(*data*)
Write a pending tag attribute from *data* to the stack.

_push_textbuffer()
Push the textbuffer onto the stack as a Text node and clear it.

_read(*delta=0*, *wrap=False*, *strict=False*)
Read the value at a relative point in the wikicode.

The value is read from `self._head` plus the value of *delta* (which can be negative). If *wrap* is `False`, we will not allow attempts to read from the end of the string if `self._head + delta` is negative. If *strict* is `True`, the route will be failed (with *_fail_route()*) if we try to read from past the end of the string; otherwise, *self.END* is returned. If we try to read from before the start of the string, *self.START* is returned.

_really_parse_entity()
Actually parse an HTML entity and ensure that it is valid.

_really_parse_external_link(*brackets*)
Really parse an external link.

_really_parse_tag()
Actually parse an HTML tag, starting with the open (`<foo>`).

_remove_uri_scheme_from_textbuffer(*scheme*)
Remove the URI scheme of a new external link from the textbuffer.

_stack
The current token stack.

_textbuffer
The current textbuffer.

_verify_safe(*this*)
Make sure we are not trying to write an invalid character.

regex = <_sre.SRE_Pattern object>

tag_splitter = <_sre.SRE_Pattern object>

tokenize(*text*, *context=0*, *skip_style_tags=False*)
Build a list of tokens from a string of wikicode and return it.

exception mwparserfromhell.parser.tokenizer.**BadRoute**(*context=0*)
Raised internally when the current tokenization route is invalid.

tokens Module This module contains the token definitions that are used as an intermediate parsing data type - they are stored in a flat list, with each token being identified by its type and optional attributes. The token list is generated

in a syntactically valid form by the *Tokenizer*, and then converted into the :class‘.Wikicode‘ tree by the *Builder*.

class mwparserfromhell.parser.tokens.**Token**
A token stores the semantic meaning of a unit of wikicode.

class mwparserfromhell.parser.tokens.**Text**

class mwparserfromhell.parser.tokens.**TemplateOpen**

class mwparserfromhell.parser.tokens.**TemplateParamSeparator**

class mwparserfromhell.parser.tokens.**TemplateParamEquals**

class mwparserfromhell.parser.tokens.**TemplateClose**

class mwparserfromhell.parser.tokens.**ArgumentOpen**

class mwparserfromhell.parser.tokens.**ArgumentSeparator**

class mwparserfromhell.parser.tokens.**ArgumentClose**

class mwparserfromhell.parser.tokens.**WikilinkOpen**

class mwparserfromhell.parser.tokens.**WikilinkSeparator**

class mwparserfromhell.parser.tokens.**WikilinkClose**

class mwparserfromhell.parser.tokens.**ExternalLinkOpen**

class mwparserfromhell.parser.tokens.**ExternalLinkSeparator**

class mwparserfromhell.parser.tokens.**ExternalLinkClose**

class mwparserfromhell.parser.tokens.**HTMLEntityStart**

class mwparserfromhell.parser.tokens.**HTMLEntityNumeric**

class mwparserfromhell.parser.tokens.**HTMLEntityHex**

class mwparserfromhell.parser.tokens.**HTMLEntityEnd**

class mwparserfromhell.parser.tokens.**HeadingStart**

class mwparserfromhell.parser.tokens.**HeadingEnd**

class mwparserfromhell.parser.tokens.**CommentStart**

class mwparserfromhell.parser.tokens.**CommentEnd**

class mwparserfromhell.parser.tokens.**TagOpenOpen**

class mwparserfromhell.parser.tokens.**TagAttrStart**

class mwparserfromhell.parser.tokens.**TagAttrEquals**

class mwparserfromhell.parser.tokens.**TagAttrQuote**

class mwparserfromhell.parser.tokens.**TagCloseOpen**

class mwparserfromhell.parser.tokens.**TagCloseSelfclose**

class mwparserfromhell.parser.tokens.**TagOpenClose**

class mwparserfromhell.parser.tokens.**TagCloseClose**

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

m

Symbols

A

B

C

L

M

N

P

Q

R

S

T

U

V

W