tinycss2

Release 0.6.1

Oct 02, 2017

Contents

tinycss2 is a rewrite of tinycss with a simpler API, based on the more recent CSS Syntax Level 3 specification.

- BSD licensed
- For Python 2.7 or 3.3+ (tested on CPython)
- Latest documentation: http://tinycss2.readthedocs.io/
- Source code and issue tracker: https://github.com/Kozea/tinycss2
- PyPI releases: https://pypi.python.org/pypi/tinycss2/
- Continuous integration:

CHAPTER 1

Installation

Installing tinycss2 with pip should Just Work:

```
pip install tinycss2
```

This will also automatically install tinycss2's only dependency, webencodings. tinycss2 and webencodings both only contain Python code and should work on any Python implementation, although they're only tested on CPython.

CHAPTER 2

Parsing

tinycss2 is "low-level" in that it doesn't parse all of CSS: it doesn't know about the syntax of any specific properties or at-rules. Instead, it provides a set of functions that can be composed to support exactly the parts of CSS you're interested in, including new or non-standard rules or properties, without modifying tinycss or having a complex hook/plugin system.

In many cases, parts of the parsed values (such as the `content` of a `AtRule`) is given as *component values* that can be parsed further with other functions.

`tinycss2.`**`parse_stylesheet_bytes`**(*css_bytes*, *protocol_encoding=None*, *environment_encoding=None*, *skip_comments=False*, *skip_whitespace=False*)

Parse stylesheet from bytes, determining the character encoding as web browsers do.

This is used when reading a file or fetching an URL. The character encoding is determined from the initial bytes (a BOM (Byte Order Mark) or an `@charset` rule) as well as the parameters. The ultimate fallback is UTF-8.

Parameters

- **`css_bytes`** – A byte string.
- **`protocol_encoding`** – A string. The encoding label, if any, defined by HTTP or equivalent protocol. (e.g. via the `charset` parameter of the `Content-Type` header.)
- **`environment_encoding`** – A `webencodings.Encoding` object for the environment encoding, if any.
- **`skip_comments`** – Ignore CSS comments at the top-level of the stylesheet. If the input is a string, ignore all comments.
- **`skip_whitespace`** – Ignore whitespace at the top-level of the stylesheet. Whitespace is still preserved in the `prelude` and the `content` of rules.

Returns

A `(rules, encoding)` tuple.

- `rules` is a list of `QualifiedRule`, `AtRule`, `Comment` (if `skip_comments` is false), `WhitespaceToken` (if `skip_whitespace` is false), and `ParseError` objects.

- `encoding` is the `webencodings.Encoding` object that was used. If `rules` contains an `@import` rule, this is the environment encoding for the imported stylesheet.

```
response = urlopen('http://example.net/foo.css')
rules, encoding = parse_stylesheet_bytes(
    css_bytes=response.read(),
    # Python 3.x
    protocol_encoding=response.info().get_content_type().get_param('charset'),
    # Python 2.x
    protocol_encoding=response.info().gettype().getparam('charset'),
)
for rule in rules:
    ...
```

`tinycss2.`**`parse_stylesheet`**(*input*, *skip_comments=False*, *skip_whitespace=False*)

Parse stylesheet from text.

This is used e.g. for a `<style>` HTML element.

This differs from *`parse_rule_list()`* in that top-level `<!--` and `-->` tokens are ignored. This is a legacy quirk for the `<style>` HTML element.

Parameters

- **`input`** – A string or an iterable of *component values*.
- **`skip_comments`** – Ignore CSS comments at the top-level of the stylesheet. If the input is a string, ignore all comments.
- **`skip_whitespace`** – Ignore whitespace at the top-level of the stylesheet. Whitespace is still preserved in the *`prelude`* and the *`content`* of rules.

Returns A list of *`QualifiedRule`*, *`AtRule`*, *`Comment`* (if `skip_comments` is false), *`WhitespaceToken`* (if `skip_whitespace` is false), and *`ParseError`* objects.

`tinycss2.`**`parse_rule_list`**(*input*, *skip_comments=False*, *skip_whitespace=False*)

Parse a non-top-level rule list.

This is used for parsing the *`content`* of nested rules like `@media`. This differs from *`parse_stylesheet()`* in that top-level `<!--` and `-->` tokens are not ignored.

Parameters

- **`input`** – A string or an iterable of *component values*.
- **`skip_comments`** – Ignore CSS comments at the top-level of the list. If the input is a string, ignore all comments.
- **`skip_whitespace`** – Ignore whitespace at the top-level of the list. Whitespace is still preserved in the *`prelude`* and the *`content`* of rules.

Returns A list of *`QualifiedRule`*, *`AtRule`*, *`Comment`* (if `skip_comments` is false), *`WhitespaceToken`* (if `skip_whitespace` is false), and *`ParseError`* objects.

`tinycss2.`**`parse_one_rule`**(*input*, *skip_comments=False*)

Parse a single qualified rule or at-rule.

This would be used e.g. by insertRule() in an implementation of CSSOM.

Parameters

- **`input`** – A string or an iterable of *component values*.
- **`skip_comments`** – If the input is a string, ignore all CSS comments.

Returns A *QualifiedRule*, *AtRule*, or *ParseError* objects.

Any whitespace or comment before or after the rule is dropped.

tinycss2.**parse_declaration_list**(*input*, *skip_comments=False*, *skip_whitespace=False*)
Parse a declaration list (which may also contain at-rules).

This is used e.g. for the *content* of a style rule or `@page` rule, or for the `style` attribute of an HTML element.

In contexts that don't expect any at-rule, all *AtRule* objects should simply be rejected as invalid.

Parameters

- **input** – A string or an iterable of *component values*.
- **skip_comments** – Ignore CSS comments at the top-level of the list. If the input is a string, ignore all comments.
- **skip_whitespace** – Ignore whitespace at the top-level of the list. Whitespace is still preserved in the *value* of declarations and the *prelude* and *content* of at-rules.

Returns A list of *Declaration*, *AtRule*, *Comment* (if `skip_comments` is false), *WhitespaceToken* (if `skip_whitespace` is false), and *ParseError* objects

tinycss2.**parse_one_declaration**(*input*, *skip_comments=False*)
Parse a single declaration.

This is used e.g. for a declaration in an @supports test.

Parameters

- **input** – A *string*, or an iterable of *component values*.
- **skip_comments** – If the input is a string, ignore all CSS comments.

Returns A *Declaration* or *ParseError*.

Any whitespace or comment before the `:` colon is dropped.

tinycss2.**parse_component_value_list**(*css*, *skip_comments=False*)
Parse a list of component values.

Parameters

- **css** – A *string*.
- **skip_comments** – Ignore CSS comments. The return values (and recursively its blocks and functions) will not contain any *Comment* object.

Returns A list of *component values*.

tinycss2.**parse_one_component_value**(*input*, *skip_comments=False*)
Parse a single component value.

This is used e.g. for an attribute value referred to by `attr(foo length)`.

Parameters

- **input** – A *string*, or an iterable of *component values*.
- **skip_comments** – If the input is a string, ignore all CSS comments.

Returns A *component value* (that is neither whitespace or comment), or a *ParseError*.

CHAPTER 3

Serialization

In addition to each node's a *serialize()* method, some serialization-related functions are available:

tinycss2.**serialize**(*nodes*)
Serialize nodes to CSS syntax.

This should be used for *component values* instead of just *serialize()* on each node as it takes care of corner cases such as **;** between declarations, and consecutive identifiers that would otherwise parse back as the same token.

Parameters **nodes** – an iterable of *Node* objects

Returns an Unicode string

tinycss2.**serialize_identifier**(*value*)
Serialize any string as a CSS identifier

Parameters **value** – a *string*

Returns an Unicode string that would parse as an *IdentToken* whose *value* attribute equals the passed value argument.

CHAPTER 4

Color

tinycss2.color3.**parse_color**(*input*)

Parse a color value as defined in CSS Color Level 3.

Parameters **input** – A *string*, or a single *component value*.

Returns

- None if the input is not a valid color value. (No exception is raised.)
- The string 'currentColor' for the *currentColor* keyword
- Or a *RGBA* object for every other values (including keywords, HSL and HSLA.) The alpha channel is clipped to [0, 1] but red, green, or blue can be out of range (eg. rgb(-10%, 120%, 0%) is represented as (-0.1, 1.2, 0, 1).)

class tinycss2.color3.**RGBA**

An RGBA color.

A tuple of four floats in the 0..1 range: (red, green, blue, alpha).

red

Convenience access to the red channel. Same as rgba[0].

green

Convenience access to the green channel. Same as rgba[1].

blue

Convenience access to the blue channel. Same as rgba[2].

alpha

Convenience access to the alpha channel. Same as rgba[3].

CHAPTER 5

<An+B>

tinycss2.nth.**parse_nth**(*input*)

Parse <An+B>, as found in :nth-child() and related Selector pseudo-classes.

Although tinycss2 does not include a full Selector parser, this bit of syntax is included as it is particularly tricky to define on top of a CSS tokenizer.

Parameters **input** – A *string*, or an iterable yielding *component values* (eg. the `arguments` of a functional pseudo-class.)

Returns A `(a, b)` tuple of integers, or `None` if the input is invalid.

CHAPTER 6

AST nodes

Various parsing functions return a **node** or a list of nodes. Some types of nodes contain nested nodes which may in turn contain more nodes, forming together an **abstract syntax tree**.

Although you typically don't need to import it, the *tinycss2.ast* module defines a class for every type of node.

class tinycss2.ast.**Node**
Every node type inherits from this class, which is never instantiated directly.

type
Each child class has a *type* class attribute with an unique string value. This allows checking for the node type with code like:

```
if node.type == 'whitespace':
```

instead of the more verbose:

```
from tinycss2.ast import WhitespaceToken
if isinstance(node, WhitespaceToken):
```

Every node also has these attributes and methods, which are not repeated for brevity:

source_line
The line number of the start of the node in the CSS source. Starts at 1.

source_column
The column number within *source_line* of the start of the node in the CSS source. Starts at 1.

serialize()
Serialize this node to CSS syntax and return an Unicode string.

class tinycss2.ast.**QualifiedRule**
A qualified rule.

```
<prelude> '{' <content> '}'
```

The interpretation of qualified rules depend on their context. At the top-level of a stylesheet or in a conditional rule such as `@media`, they are **style rules** where the *`prelude`* is Selectors list and the *`content`* is a list of property declarations.

`type` = 'qualified-rule'

`prelude`
The rule's prelude, the part before the {} block, as a list of *component values*.

`content`
The rule's content, the part inside the {} block, as a list of *component values*.

class `tinycss2.ast.`**`AtRule`**
An at-rule.

```
@<at_keyword> <prelude> '{' <content> '}'
@<at_keyword> <prelude> ';'
```

The interpretation of at-rules depend on their at-keyword as well as their context. Most types of at-rules (ie. at-keyword values) are only allowed in some context, and must either end with a {} block or a semicolon.

`type` = 'at-rule'

`at_keyword`
The unescaped value of the rule's at-keyword, without the `@` symbol, as an Unicode string.

`lower_at_keyword`
Same as *`at_keyword`* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS at-keyword.

```
if node.type == 'at-rule' and node.lower_at_keyword == 'import':
```

`prelude`
The rule's prelude, the part before the {} block or semicolon, as a list of *component values*.

`content`
The rule's content, if any. The block's content as a list of *component values* for at-rules with a {} block, or `None` for at-rules ending with a semicolon.

class `tinycss2.ast.`**`Declaration`**
A (property or descriptor) declaration.

```
<name> ':' <value>
<name> ':' <value> '!important'
```

`type` = 'declaration'

`name`
The unescaped name, as an Unicode string.

`type` = 'declaration'

`lower_name`
Same as *`name`* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS property or descriptor name.

```
if node.type == 'declaration' and node.lower_name == 'color':
```

`value`
The declaration value as a list of *component values*: anything between `:` and the end of the declaration, or `!important`.

important
A boolean, true if the declaration had an `!important` marker. It is up to the consumer to reject declarations that do not accept this flag, such as non-property descriptor declarations.

Component values

class `tinycss2.ast.`**ParseError**
A syntax error of some sort. May occur anywhere in the tree.

Syntax errors are not fatal in the parser to allow for different error handling behaviors. For example, an error in a Selector list makes the whole rule invalid, but an error in a Media Query list only replaces one comma-separated query with `not all`.

type = 'error'

kind
Machine-readable string indicating the type of error. Example: `'bad-url'`.

message
Human-readable explanation of the error, as a string. Could be translated, expanded to include details, etc.

class `tinycss2.ast.`**Comment**
A CSS comment.

```
'/*' <value> '*/'
```

Comments can be ignored by passing `skip_comments=True` to functions such as *`parse_component_value_list()`*.

type = 'comment'

value
The content of the comment, between `/*` and `*/`, as a string.

class `tinycss2.ast.`**WhitespaceToken**
A <whitespace-token>.

type = 'whitespace'

value
The whitespace sequence, as a string, as in the original CSS source.

class `tinycss2.ast.`**LiteralToken**
Token that represents one or more characters as in the CSS source.

type = 'literal'

value
A string of one to four characters.

Instances compare equal to their *`value`*, so that these are equivalent:

```
if node == ';':
if node.type == 'literal' and node.value == ';':
```

This regroups what the specification defines as separate token types:

- *<colon-token>* `:`
- *<semicolon-token>* `;`

- *<comma-token>* `,`
- *<cdc-token>* `-->`
- *<cdo-token>* `<!--`
- *<include-match-token>* `~=`
- *<dash-match-token>* `|=`
- *<prefix-match-token>* `^=`
- *<suffix-match-token>* `$=`
- *<substring-match-token>* `*=`
- *<column-token>* `||`
- *<delim-token>* (a single ASCII character not part of any another token)

class `tinycss2.ast.`**`IdentToken`**
An <ident-token>.

`type` = 'ident'

`value`
The unescaped value, as an Unicode string.

`lower_value`
Same as *`value`* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS keyword.

class `tinycss2.ast.`**`AtKeywordToken`**
An <at-keyword-token>.

```
'@' <value>
```

`type` = 'at-keyword'

`value`
The unescaped value, as an Unicode string, without the preceding `@`.

`lower_value`
Same as *`value`* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS at-keyword.

```
if node.type == 'at-keyword' and node.lower_value == 'import':
```

class `tinycss2.ast.`**`HashToken`**
A <hash-token>.

```
'#' <value>
```

`type` = 'hash'

`value`
The unescaped value, as an Unicode string, without the preceding `#`.

`is_identifier`
A boolean, true if the CSS source for this token was `#` followed by a valid identifier. (Only such hash tokens are valid ID selectors.)

class `tinycss2.ast.`**`StringToken`**
A <string-token>.

```
'"' <value> '"'
```

type = 'string'

value
The unescaped value, as an Unicode string, without the quotes.

class tinycss2.ast.**URLToken**
An <url-token>.

```
'url("' <value> '")'
```

type = 'url'

value
The unescaped URL, as an Unicode string, without the `url(` and `)` markers or the optional quotes.

class tinycss2.ast.**UnicodeRangeToken**
An <unicode-range-token>.

type = 'unicode-range'

start
The start of the range, as an integer between 0 and 1114111.

end
The end of the range, as an integer between 0 and 1114111. Same as *start* if the source only specified one value.

class tinycss2.ast.**NumberToken**
A <numer-token>.

type = 'number'

value
The numeric value as a `float`.

int_value
The numeric value as an `int` if *is_integer* is true, `None` otherwise.

is_integer
Whether the token was syntactically an integer, as a boolean.

representation
The CSS representation of the value, as an Unicode string.

class tinycss2.ast.**PercentageToken**
A <percentage-token>.

```
<representation> '%'
```

type = 'percentage'

value
The value numeric as a `float`.

int_value
The numeric value as an `int` if the token was syntactically an integer, or `None`.

is_integer
Whether the token's value was syntactically an integer, as a boolean.

representation
The CSS representation of the value without the unit, as an Unicode string.

class tinycss2.ast.**DimensionToken**
A <dimension-token>.

```
<representation> <unit>
```

type = 'dimension'

value
The value numeric as a `float`.

int_value
The numeric value as an `int` if the token was syntactically an integer, or `None`.

is_integer
Whether the token's value was syntactically an integer, as a boolean.

representation
The CSS representation of the value without the unit, as an Unicode string.

unit
The unescaped unit, as an Unicode string.

lower_unit
Same as *`unit`* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS unit.

```
if node.type == 'dimension' and node.lower_unit == 'px':
```

class tinycss2.ast.**ParenthesesBlock**
A <()-block>.

```
'(' <content> ')'
```

type = '() block'

content
The content of the block, as list of *component values*. The `(` and `)` markers themselves are not represented in the list.

class tinycss2.ast.**SquareBracketsBlock**
A <[]-block>.

```
'[' <content> ']'
```

type = '[] block'

content
The content of the block, as list of *component values*. The `[` and `]` markers themselves are not represented in the list.

class tinycss2.ast.**CurlyBracketsBlock**
A <{}-block>.

```
'{' <content> '}'
```

type = '{} block'

content
The content of the block, as list of *component values*. The `[` and `]` markers themselves are not represented in the list.

class `tinycss2.ast.`**FunctionBlock**
A <function-block>.

```
<name> '(' <arguments> ')'
```

type = 'function'

name
The unescaped name of the function, as an Unicode string.

lower_name
Same as *name* but normalized to *ASCII lower case*, see `ascii_lower()`. This is the value to use when comparing to a CSS function name.

arguments
The arguments of the function, as list of *component values*. The `(` and `)` markers themselves are not represented in the list. Commas are not special, but represented as *LiteralToken* objects in the list.

CHAPTER 7

Glossary

String In this documentation "a string" means an Unicode string: `unicode` on Python 2.x and `str` on Python 3.x. On 2.x, a byte string (`str`) that only contains ASCII bytes is also accepted and implicitly decoded.

Component value

Component values A *ParseError*, *WhitespaceToken*, *LiteralToken*, *IdentToken*, *AtKeywordToken*, *HashToken*, *StringToken*, *URLToken*, *NumberToken*, *PercentageToken*, *DimensionToken*, *UnicodeRangeToken*, *ParenthesesBlock*, *SquareBracketsBlock*, *CurlyBracketsBlock*, *FunctionBlock*, or *Comment* object.

CHAPTER 8

tinycss2 changelog

Version 0.6.1

Released on 2017-10-02.

- Update documentation.

Version 0.6.0

Released on 2017-08-16.

- Don't allow identifiers starting with two dashes.
- Don't use Tox for tests.
- Follow semantic versioning.

Version 0.5

Released on 2014-08-19.

- Update for spec changes.
- Add a *value* attribute to *WhitespaceToken*.
- **Breaking change**: CSS comments are now preserved as *Comment* objects by default. Pass `skip_comments=True` to parsing functions to get the old behavior.
- **Breaking change**: Top-level comments and whitespace are now preserved when parsing a stylesheet, rule list, or declaration list. Pass `skip_comments=True` and `skip_whitespace=True` to get the old behavior.
- Test on Python 3.4 and PyPy3.
- Set up continous integration on Travis-CI.

Version 0.4

Released on 2014-01-04.

- Fix *HashToken* starting with a non-ASCII character.
- Fix `repr()` on AST nodes.

Version 0.3

Released on 2013-12-27.

- Document all the things!
- Add *Serialization*
- Merge `tinycss2.color3.parse_color_string()` behavior into *parse_color()*.
- Fix and test parsing form bytes and tokenization of <unicode-range>.

Version 0.2

Released on 2013-09-02.

Add parsing for <An+B>, as in `:nth-child()` and related Selectors pseudo-classes.

Version 0.1

Released on 2013-08-31.

First PyPI release. Contains:

- Decoding from bytes (`@charset`, etc.)
- Tokenization
- Parsing for "generic" rules and declarations
- Parsing for CSS Color Level 3
- Tests for all of the above, except for decoding from bytes.

Python Module Index

t

Index

Q

R

S

T

U

V

W