

---

# wheezy.template documentation

*Release latest*

**Andriy Kornatskyy**

**Sep 13, 2023**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Contents</b>	<b>3</b>
2.1	Getting Started . . . . .	3
2.2	Examples . . . . .	3
2.3	User Guide . . . . .	5
2.4	Modules . . . . .	11
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



# CHAPTER 1

---

## Introduction

---

*wheezy.template* is a [python](#) package written in pure Python code. It is a lightweight template library. The design goals achieved with its development:

- **Compact, Expressive, Clean:** Minimizes the number of keystrokes required to build a template. Enables fast and well read coding. You do not need to explicitly denote statement blocks within HTML (unlike other template systems), the parser is smart enough to understand your code. This enables a compact and expressive syntax which is really clean and just pleasure to type.
- **Intuitive, No time to Learn:** Basic Python programming skills plus HTML markup. You are productive right from the start. Use the full power of Python with minimal markup required to denote python statements.
- **Do Not Repeat Yourself:** Master layout templates for inheritance; include and import directives for maximum reuse.
- **Blazingly Fast:** The most effective python code offers the maximum rendering performance; ultimate speed and context preprocessor features.

Simple template:

```
@require(user, items)
Welcome, @user.name!
@if items:
    @for i in items:
        @i.name: $i.price!s.
    @end
@else:
    No items found.
@end
```

It is optimized for performance, well tested and documented.

Resources:

- [source code](#), [examples](#) and [issues](#) tracker are available on [github](#)
- [documentation](#)



## 2.1 Getting Started

### 2.1.1 Install

*wheelzy.template* requires [python](#) version 3.6+. It is operating system independent. You can install it from the [pypi](#) site:

```
$ pip install wheelzy.template
```

## 2.2 Examples

Before we proceed let's setup a [virtualenv](#) environment, activate it and install:

```
$ pip install wheelzy.template
```

### 2.2.1 Big Table

The big table demo compares *wheelzy.template* with other template engines in terms of how fast a table with 10 columns x 1000 rows can be generated:

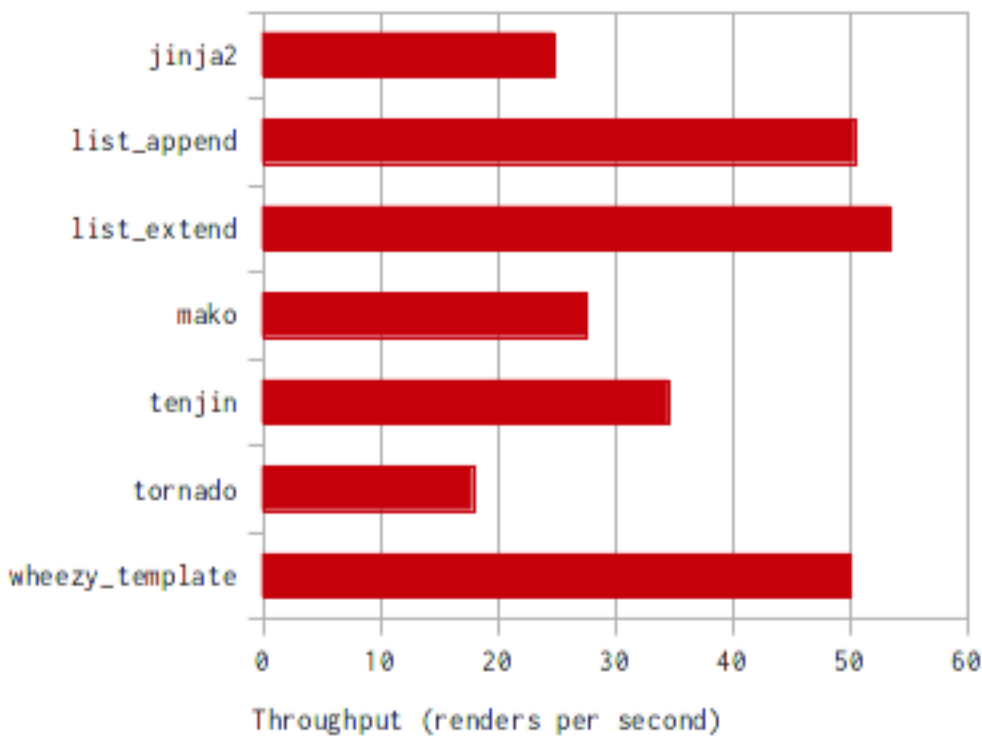
```
@require(table)
<table>
  @for row in table:
    <tr>
      @for key, value in row.items():
        <td>@key!h</td><td>@value!s</td>
      @end
    </tr>
  @end
</table>
```

Install packages used in benchmark test:

```
pip install install -O2 jinja2 mako tenjin \
    tornado wheelzy.html wheelzy.template
```

Download [bigtable.py](#) source code and run it (Intel Core 2 Quad CPU Q6600 @ 2.40GHz × 4; Kernel Linux 3.2.0-2-686-pae; Debian Testing; Python 2.7.3):

```
$ python bigtable.py
jinja2                40.22ms  24.86rps
list_append           19.85ms  50.39rps
list_extend           18.71ms  53.46rps
mako                  36.19ms  27.63rps
tenjin                28.97ms  34.52rps
tornado              55.91ms  17.89rps
wheelzy_template      19.99ms  50.02rps
```



## 2.2.2 Real World

There is real world example available in the [wheelzy.web](#) package. It can be found in the [demo.template](#) application. The application has a few screens: home, sign up, sign in, etc. The content implementation is available for jinja2, mako, tenjin, wheelzy.template and for wheelzy.template with preprocessor.

The throughput was captured using apache benchmark (concurrency level 500, number of request 100K):

	/	/en/signin	/en/signup
jinja2	9339	6422	6196
mako	9681	6720	6567
tenjin	11138	7233	7203
wheelzy.template	15023	8898	8900

(continues on next page)

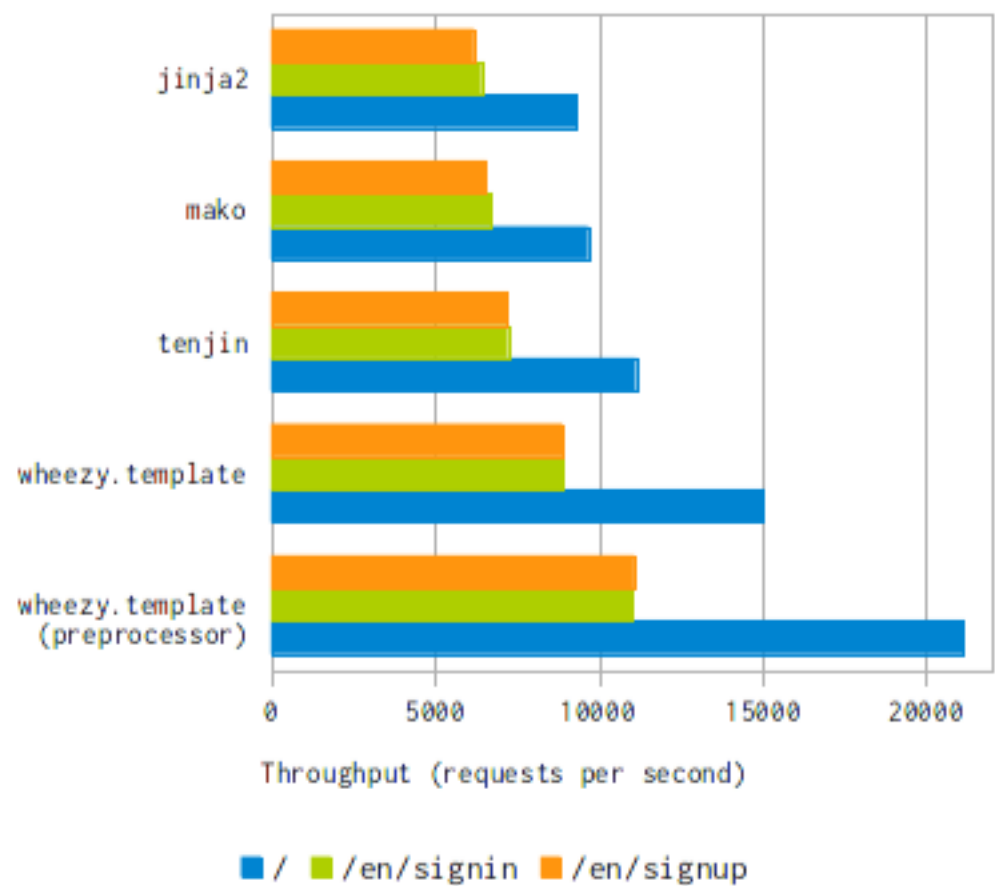


(continued from previous page)

wheezy.template (preprocessor)	21144	11027	11087
-----------------------------------	-------	-------	-------

Environment specification:

- \* Client: Intel Core 2 Quad CPU Q6600 @ 2.40GHz x 4, Kernel 3.2.0-3-686-pae
- \* Server: Intel Xeon CPU X3430 @ 2.40GHz x 4, Kernel 3.2.0-3-amd64, uwsgi 1.2.4
- \* Debian Testing, Python 2.7.3, LAN 1 Gb.



## 2.3 User Guide

*wheezy.template* uses *Engine* to store configuration information and load templates. Here is a typical example that loads templates from the file system:

```
from wheezy.template.engine import Engine
from wheezy.template.ext.core import CoreExtension
from wheezy.template.loader import FileLoader

searchpath = ['content/templates-wheezy']
engine = Engine(
    loader=FileLoader(searchpath),
    extensions=[CoreExtension()]
)
```

(continues on next page)

(continued from previous page)

```
)  
template = engine.get_template('template.html')
```

You can render content from console:

```
wheezy.template demos/helloworld/hello.txt demos/helloworld/hello.json
```

another example:

```
wheezy.template -s demos/master index.html
```

## 2.3.1 Loaders

Loader is used to provide template content to *Engine* by some name requested by an application. What exactly constitutes a name and how each loader interprets it, is up to the loader implementation.

*wheezy.template* comes with the following loaders:

- *FileLoader* - loads templates from file system (*directories* - search path of directories to scan for template, *encoding* - template content encoding).
- *DictLoader* - loads templates from python dictionary (*templates* - a dict where key corresponds to template name and value to template content).
- *ChainLoader* - loads templates from loaders in turn until one succeeds.

## 2.3.2 Core Extension

The *CoreExtension* includes support for basic python statements, variables processing and markup.

### Context

In order to use variables passed to a template you use `require` statement and list names you need to pick from the context. These names becomes visible from the `require` statement to the end of the template scope (imagine a single template is a python function).

Context access syntax:

```
@require(var1, var2, ...)
```

### Variables

The application passes variables to the template renderer via context. Variable access syntax:

```
@variable_name  
@{variable_name}  
@{ variable_name }
```

Example:

```

from wheezy.template.engine import Engine
from wheezy.template.ext.core import CoreExtension
from wheezy.template.loader import DictLoader

template = """\
@require(name)
Hello, @name"""

engine = Engine(
    loader=DictLoader({'x': template}),
    extensions=[CoreExtension()]
)
template = engine.get_template('x')

print(template.render({'name': 'John'}))

```

Variable syntax is not limited to a single name access. You are able to use full power of python to access items in a dict, attributes, function calls, etc.

## Filters

Variables can be formatted by filters. Filters are separated from the variable by the ! symbol. Filter syntax:

```

@variable_name!filter1!filter2
@{variable_name!!filter1!filter2}
@{ variable_name !! filter1!filter2 }

```

The filters are applied from left to right, so the above syntax is equivalent to the following call:

```
@filter1(filter2(variable_name))
```

Example:

```

@user.age!s
@{user.age!!s}
@{ user.age !!s }

```

Assuming the age property of user is integer we apply a string filter.

You are able to use custom filters, here is an example on how to use html escape filter:

```

try:
    from wheezy.html.utils import escape_html as escape
except ImportError:
    import cgi
    escape = cgi.escape

# ... initialize Engine.
engine.global_vars.update({'e': escape})

```

First we try import an optimized version of html escape from `wheezy.html` package and if it is not available fallback to the one from the `cgi` package. Next we update the engine's global variables with the escape function, which is accessible as the `e` filter name in template:

```

@user.name!e
@{ user.name !! e }

```

You are able use engine `global_vars` dictionary in order to simplify your template access to some commonly used variables.

## R-value expressions

You can use single line r-value expressions that evaluates to a rendered value:

```
@{ accepted and 'YES' or 'NO' }
@{ (age > 20 and age < 120) and 'OK' or '?' }
@{ n > 0 and 1 or -1 !! s }
```

## Line Statements

The following python line statements are supported: *if*, *else*, *elif*, *for*. Here is simple example:

```
@require(items)
@if items:
    @for i in items:
        @i.name: $i.price!s.
    @end
@else:
    No items found.
@end
```

## Comments

Only single line comments are supported:

```
@# TODO:
```

## Line Join

In case you need continue a long line without breaking it with new line during rendering use line join (`\`):

```
@if menu_name == active:
    <li class='active'> \
@else:
    <li> \
@endif
```

## Inheritance

Template inheritance allows you to build a master template that contains common layout of your site and defines areas that a child templates can override.

## Master Template

Master template is used to provide common layout of your site. Let's define a master template (name `shared/master.html`):

```

<html>
  <head>
    <title>
      @def title():
      @end
      @title() - My Site</title>
    </head>
    <body>
      <div id="content">
        @def content():
        @end
        @content()
      </div>
      <div id="footer">
        @def footer():
        &copy; Copyright 2012 by Me.
        @end
        footer()
      </div>
    </body>
  </html>

```

In this example, the @def tags define python functions (substitution areas). These functions are inserted into a specific places (right after definition). These places become place holders for child templates. The @footer place holder defines default content while @title and @content are just empty.

## Child Template

Child templates are used to extend master templates via the defined place holders:

```

@extends("shared/master.html")

@def title():
  Welcome
@end

@def content():
  <h1>Home</h1>
  <p>
    Welcome to My Site!
  </p>
@end

```

In this example, the @title and @content place holders are overridden by the child template.

Note, @import and @require tokens are allowed at @extends token level.

## Include

The include is useful to insert a template content just in place of the statement:

```
@include("shared/snippet/script.html")
```

## Import

The import is used to reuse some code stored in other files. So you are able import all functions defined by that template:

```
@import "shared/forms.html" as forms

@forms.textbox('username')
```

or just certain name:

```
@from "shared/forms.html" import textbox

@textbox(name='username')
```

Once imported you use these names as variables in a template.

### 2.3.3 Code Extension

The *CodeExtension* includes support for embedded python code. Syntax:

```
@(
    # any python code
)
```

### 2.3.4 Preprocessor

The *Preprocessor* processes templates with syntax for the preprocessor engine and varying runtime templates (with runtime engine factory) by some key function that is context driven. Here is an example:

```
from wheezy.html.utils import html_escape
from wheezy.template.engine import Engine
from wheezy.template.ext.core import CoreExtension
from wheezy.template.ext.determined import DeterminedExtension
from wheezy.template.loader import FileLoader
from wheezy.template.preprocessor import Preprocessor

def runtime_engine_factory(loader):
    engine = Engine(
        loader=loader,
        extensions=[
            CoreExtension(),
        ])
    engine.global_vars.update({
        'h': html_escape,
    })
    return engine

searchpath = ['content/templates']
engine = Engine(
    loader=FileLoader(searchpath),
    extensions=[
        CoreExtension('#', line_join=None),
        DeterminedExtension(['path_for', '_']),
```

(continues on next page)

(continued from previous page)

```

    })
    engine.global_vars.update({
    })
    engine = Preprocessor(runtime_engine_factory, engine,
                          key_factory=lambda ctx: ctx['locale'])

```

In this example, the *Preprocessor* is defined to use engine with the start token defined as '#'. Any directives starting with # are processed once only by the preprocessor engine. The *key\_factory* is dependent on runtime context and particularly on 'locale'. This way runtime engine factory is varied by locale so locale dependent functions (*\_* and *path\_for*) are processed only once by the preprocessor. See complete example in [wheezy.web demo.template applicaiton](#).

## 2.4 Modules

### 2.4.1 wheezy.template

```

class wheezy.template.Engine(loader: wheezy.template.typing.Loader, extensions: List[Any], tem-
                             plate_class: Optional[Callable[[str, Callable[[Mapping[str,
Any], Mapping[str, Any], Mapping[str, Any]], str]],
                             wheezy.template.typing.SupportsRender]] = None)

```

The core component of template engine.

**get\_template** (name: str) → wheezy.template.typing.SupportsRender  
Returns compiled template.

**remove** (name: str) → None  
Removes given name from internal cache.

**render** (name: str, ctx: Mapping[str, Any], local\_defs: Mapping[str, Any], super\_defs: Mapping[str, Any]) → str  
Renders template by name in given context.

```

class wheezy.template.CodeExtension(token_start: str = '@')
    Includes support for embedded python code.

```

```

class wheezy.template.CoreExtension(token_start: str = '@', line_join: str = '\')
    Includes basic statements, variables processing and markup.

```

```

class wheezy.template.DictLoader(templates: Mapping[str, str])
    Loads templates from python dictionary.

```

*templates* - a dict where key corresponds to template name and value to template content.

**list\_names** () → Tuple[str, ...]  
List all keys from internal dict.

**load** (name: str) → Optional[str]  
Returns template by name.

```

class wheezy.template.FileLoader(directories: List[str], encoding: str = 'UTF-8')
    Loads templates from file system.

```

*directories* - search path of directories to scan for template. *encoding* - decode template content per encoding.

**get\_fullname** (name: str) → Optional[str]  
Returns a full path by a template name.

**list\_names** () → Tuple[str, ...]

Return a list of names relative to directories. Ignores any files and directories that start with dot.

**load** (name: str) → Optional[str]

Loads a template by name from file system.

**class** wheezy.template.PreprocessLoader (engine: wheezy.template.engine.Engine, ctx: Optional[Mapping[str, Any]] = None)

Performs preprocessing of loaded template.

**class** wheezy.template.Preprocessor (runtime\_engine\_factory: Callable[[wheezy.template.typing.Loader], wheezy.template.engine.Engine], engine: wheezy.template.engine.Engine, key\_factory: Callable[[Mapping[str, Any], str]])

Preprocess templates with engine and vary runtime templates by key\_factory function using runtime\_engine\_factory.

## 2.4.2 wheezy.template.builder

## 2.4.3 wheezy.template.compiler

## 2.4.4 wheezy.template.console

## 2.4.5 wheezy.template.engine

**class** wheezy.template.engine.Engine (loader: wheezy.template.typing.Loader, extensions: List[Any], template\_class: Optional[Callable[[str, Callable[[Mapping[str, Any], Mapping[str, Any], Mapping[str, Any]], str]], wheezy.template.typing.SupportsRender]] = None)

The core component of template engine.

**get\_template** (name: str) → wheezy.template.typing.SupportsRender

Returns compiled template.

**remove** (name: str) → None

Removes given name from internal cache.

**render** (name: str, ctx: Mapping[str, Any], local\_defs: Mapping[str, Any], super\_defs: Mapping[str, Any]) → str

Renders template by name in given context.

**class** wheezy.template.engine.Template (name: str, render\_template: Callable[[Mapping[str, Any], Mapping[str, Any], Mapping[str, Any]], str])

Simple template class.

**wheezy.template.engine.complement\_syntax\_error** (err: SyntaxError, template\_source: str, source: str) → SyntaxError

Complements SyntaxError with template and source snippets, like one below:

```
File "shared/snippet/widget.html", line 4
    if :

template snippet:
02 <h1>
03     @msg!h
04     @if :
05         sd
```

(continues on next page)



(continued from previous page)

```

06      @end

generated snippet:
02      _b = []; w = _b.append; w('<h1>\n      ')
03      w(h(msg)); w('\n')
04      if :
05          w('          sd\n')
06

      if :
      ^
SyntaxError: invalid syntax

```

## 2.4.6 wheezy.template.lexer

**class** `wheezy.template.lexer.Lexer` (*lexer\_rules: List[Tuple[Pattern[AnyStr], Callable[[Match[AnyStr], Tuple[int, str, str]]], preprocessors: Optional[List[Callable[[str, str]]] = None, postprocessors: Optional[List[Callable[[List[Tuple[int, str, str]]], str]] = None, \*\*ignore)*

Tokenizes input source per rules supplied.

**tokenize** (*source: str*) → List[Tuple[int, str, str]]

Translates source accoring to lexer rules into an iterable of tokens.

`wheezy.template.lexer.lexer_scan` (*extensions: List[Any]*) → Mapping[str, Any]

Scans extensions for `lexer_rules` and `preprocessors` attributes.

## 2.4.7 wheezy.template.loader

**class** `wheezy.template.loader.AutoReloadProxy` (*engine: wheezy.template.engine.Engine*)

**get\_template** (*name: str*) → `wheezy.template.typing.SupportsRender`

Returns compiled template.

**remove** (*name: str*) → None

Removes given name from internal cache.

**render** (*name: str, ctx: Mapping[str, Any], local\_defs: Mapping[str, Any], super\_defs: Mapping[str, Any]*) → str

Renders template by name in given context.

**class** `wheezy.template.loader.ChainLoader` (*loaders: List[wheezy.template.typing.Loader]*)

Loads templates from `loaders` until first succeed.

**list\_names** () → Tuple[str, ...]

Returns as list of names from all loaders.

**load** (*name: str*) → Optional[str]

Returns template by name from the first loader that succeed.

**class** `wheezy.template.loader.DictLoader` (*templates: Mapping[str, str]*)

Loads templates from python dictionary.

`templates` - a dict where key corresponds to template name and value to template content.

**list\_names** () → Tuple[str, ...]  
List all keys from internal dict.

**load** (name: str) → Optional[str]  
Returns template by name.

**class** wheezy.template.loader.**FileLoader** (directories: List[str], encoding: str = 'UTF-8')  
Loads templates from file system.

directories - search path of directories to scan for template. encoding - decode template content per encoding.

**get\_fullname** (name: str) → Optional[str]  
Returns a full path by a template name.

**list\_names** () → Tuple[str, ...]  
Return a list of names relative to directories. Ignores any files and directories that start with dot.

**load** (name: str) → Optional[str]  
Loads a template by name from file system.

**class** wheezy.template.loader.**PreprocessLoader** (engine: wheezy.template.engine.Engine, ctx: Optional[Mapping[str, Any]] = None)  
Performs preprocessing of loaded template.

wheezy.template.loader.**autoreload** (engine: wheezy.template.engine.Engine, enabled: bool = True) → wheezy.template.engine.Engine  
Auto reload template if changes are detected in file.

Limitation: master (inherited), imported and preprocessed templates.

It is recommended to use application server that supports file reload instead.

## 2.4.8 wheezy.template.parser

**class** wheezy.template.parser.**Parser** (parser\_rules: Dict[str, Callable[[str], Union[str, List[str]]]], parser\_configs: Optional[List[Callable[[wheezy.template.typing.ParserConfig], None]]] = None, \*\*ignore)

continue\_tokens are used to insert end node right before them to simulate a block end. Such nodes have token value None.

out\_tokens are combined together into a single node.

**end\_continue** (tokens: List[Tuple[int, str, str]]) → Iterator[Tuple[int, str, str]]  
If token is in continue\_tokens prepend it with end token so it simulate a closed block.

## 2.4.9 wheezy.template.preprocessor

**class** wheezy.template.preprocessor.**Preprocessor** (runtime\_engine\_factory: Callable[[wheezy.template.typing.Loader, wheezy.template.engine.Engine], wheezy.template.engine.Engine], key\_factory: Callable[[Mapping[str, Any], str]])

Preprocess templates with engine and vary runtime templates by key\_factory function using runtime\_engine\_factory.

### 2.4.10 wheezy.template.utils

`wheezy.template.utils.find_all_balanced` (*text: str, start: int = 0*) → int

Finds balanced ( [ with ] ) assuming that start is pointing to ( or [ in text.

`wheezy.template.utils.find_balanced` (*text: str, start: int = 0, start\_sep: str = '(', end\_sep: str = ')'*) → int

Finds balanced start\_sep with end\_sep assuming that start is pointing to start\_sep in text.

### 2.4.11 wheezy.template.ext.code

**class** `wheezy.template.ext.code.CodeExtension` (*token\_start: str = '@'*)

Includes support for embedded python code.

### 2.4.12 wheezy.template.ext.core

**class** `wheezy.template.ext.core.CoreExtension` (*token\_start: str = '@', line\_join: str = '\'*)

Includes basic statements, variables processing and markup.

`wheezy.template.ext.core.rvalue_token` (*m: Match[str]*) → Tuple[int, str, str]

Produces variable token as r-value expression.

`wheezy.template.ext.core.stmt_token` (*m: Match[str]*) → Tuple[int, str, str]

Produces statement token.

`wheezy.template.ext.core.var_token` (*m: Match[str]*) → Tuple[int, str, str]

Produces variable token.

### 2.4.13 wheezy.template.ext.determined

**class** `wheezy.template.ext.determined.DeterminedExtension` (*known\_calls: List[str], runtime\_token\_start: str = '@', token\_start: str = '#'*)

Tranlates function calls between template engines.

Strictly determined known calls are converted to preprocessor calls, e.g.:

```
@_('Name:')
@path_for('default')
@path_for('static', path='/static/css/site.css')
```

Those that are not strictly determined are ignored and processed by runtime engine.

`wheezy.template.ext.determined.determined` (*expression: str*) → bool

Checks if expresion is strictly determined.

```
>>> determined("'default'")
True
>>> determined('name')
False
>>> determined("'default', id=id")
False
>>> determined("'default', lang=100")
True
```

(continues on next page)

(continued from previous page)

```
>>> determined('')
True
```

wheezy.template.ext.determined.**parse\_args** (*text: str*) → List[str]  
Parses argument type of parameters.

```
>>> parse_args('')
[]
>>> parse_args('10, "x"')
['10', '"x"']
>>> parse_args("'x', 100")
['"x"', '100']
>>> parse_args('"default"')
['default']
```

wheezy.template.ext.determined.**parse\_kwargs** (*text: str*) → Mapping[str, str]  
Parses key-value type of parameters.

```
>>> parse_kwargs('id=item.id')
{'id': 'item.id'}
>>> sorted(parse_kwargs('lang="en", id=12').items())
[('id', '12'), ('lang', '"en"')]
```

wheezy.template.ext.determined.**parse\_params** (*text: str*) → Tuple[List[str], Mapping[str, str]]  
Parses function parameters.

```
>>> parse_params('')
([], {})
>>> parse_params('id=item.id')
([], {'id': 'item.id'})
>>> parse_params('"default"')
(['default'], {})
>>> parse_params('"default", lang="en"')
(['default'], {'lang': '"en"'})
```

wheezy.template.ext.determined.**str\_or\_int** (*text: str*) → bool  
Ensures text as string or int expression.

```
>>> str_or_int('"default"')
True
>>> str_or_int('"Hello"')
True
>>> str_or_int('100')
True
>>> str_or_int('item.id')
False
```

### W

- `wheezy.template`, [11](#)
- `wheezy.template.builder`, [12](#)
- `wheezy.template.compiler`, [12](#)
- `wheezy.template.console`, [12](#)
- `wheezy.template.engine`, [12](#)
- `wheezy.template.ext.code`, [15](#)
- `wheezy.template.ext.core`, [15](#)
- `wheezy.template.ext.determined`, [15](#)
- `wheezy.template.lexer`, [13](#)
- `wheezy.template.loader`, [13](#)
- `wheezy.template.parser`, [14](#)
- `wheezy.template.preprocessor`, [14](#)
- `wheezy.template.utils`, [15](#)



## A

`autoreload()` (in module `wheezy.template.loader`), 14

`AutoReloadProxy` (class in `wheezy.template.loader`), 13

## C

`ChainLoader` (class in `wheezy.template.loader`), 13

`CodeExtension` (class in `wheezy.template`), 11

`CodeExtension` (class in `wheezy.template.ext.code`), 15

`complement_syntax_error()` (in module `wheezy.template.engine`), 12

`CoreExtension` (class in `wheezy.template`), 11

`CoreExtension` (class in `wheezy.template.ext.core`), 15

## D

`determined()` (in module `wheezy.template.ext.determined`), 15

`DeterminedExtension` (class in `wheezy.template.ext.determined`), 15

`DictLoader` (class in `wheezy.template`), 11

`DictLoader` (class in `wheezy.template.loader`), 13

## E

`end_continue()` (`wheezy.template.parser.Parser` method), 14

`Engine` (class in `wheezy.template`), 11

`Engine` (class in `wheezy.template.engine`), 12

## F

`FileLoader` (class in `wheezy.template`), 11

`FileLoader` (class in `wheezy.template.loader`), 14

`find_all_balanced()` (in module `wheezy.template.utils`), 15

`find_balanced()` (in module `wheezy.template.utils`), 15

## G

`get_fullname()` (`wheezy.template.FileLoader` method), 11

`get_fullname()` (`wheezy.template.loader.FileLoader` method), 14

`get_template()` (`wheezy.template.Engine` method), 11

`get_template()` (`wheezy.template.engine.Engine` method), 12

`get_template()` (`wheezy.template.loader.AutoReloadProxy` method), 13

## L

`Lexer` (class in `wheezy.template.lexer`), 13

`lexer_scan()` (in module `wheezy.template.lexer`), 13

`list_names()` (`wheezy.template.DictLoader` method), 11

`list_names()` (`wheezy.template.FileLoader` method), 11

`list_names()` (`wheezy.template.loader.ChainLoader` method), 13

`list_names()` (`wheezy.template.loader.DictLoader` method), 13

`list_names()` (`wheezy.template.loader.FileLoader` method), 14

`load()` (`wheezy.template.DictLoader` method), 11

`load()` (`wheezy.template.FileLoader` method), 12

`load()` (`wheezy.template.loader.ChainLoader` method), 13

`load()` (`wheezy.template.loader.DictLoader` method), 14

`load()` (`wheezy.template.loader.FileLoader` method), 14

## P

`parse_args()` (in module `wheezy.template.ext.determined`), 16

`parse_kwargs()` (in module `wheezy.template.ext.determined`), 16

`parse_params()` (in module `wheezy.template.ext.determined`), 16  
`Parser` (class in `wheezy.template.parser`), 14  
`PreprocessLoader` (class in `wheezy.template`), 12  
`PreprocessLoader` (class in `wheezy.template.loader`), 14  
`Preprocessor` (class in `wheezy.template`), 12  
`Preprocessor` (class in `wheezy.template.preprocessor`), 14

## R

`remove()` (`wheezy.template.Engine` method), 11  
`remove()` (`wheezy.template.engine.Engine` method), 12  
`remove()` (`wheezy.template.loader.AutoReloadProxy` method), 13  
`render()` (`wheezy.template.Engine` method), 11  
`render()` (`wheezy.template.engine.Engine` method), 12  
`render()` (`wheezy.template.loader.AutoReloadProxy` method), 13  
`rvalue_token()` (in module `wheezy.template.ext.core`), 15

## S

`stmt_token()` (in module `wheezy.template.ext.core`), 15  
`str_or_int()` (in module `wheezy.template.ext.determined`), 16

## T

`Template` (class in `wheezy.template.engine`), 12  
`tokenize()` (`wheezy.template.lexer.Lexer` method), 13

## V

`var_token()` (in module `wheezy.template.ext.core`), 15

## W

`wheezy.template` (module), 11  
`wheezy.template.builder` (module), 12  
`wheezy.template.compiler` (module), 12  
`wheezy.template.console` (module), 12  
`wheezy.template.engine` (module), 12  
`wheezy.template.ext.code` (module), 15  
`wheezy.template.ext.core` (module), 15  
`wheezy.template.ext.determined` (module), 15  
`wheezy.template.lexer` (module), 13  
`wheezy.template.loader` (module), 13  
`wheezy.template.parser` (module), 14  
`wheezy.template.preprocessor` (module), 14  
`wheezy.template.utils` (module), 15