Tonnikala Documentation

Release 0.17

Antti Haapala, Ilja Everilä, Pete Sevander, Hiu Nguyn

July 22, 2015

Contents

Tonnikala is the latest reincarnation among the Python templating languages that feed on Kid-inspired XML syntax. It rejects the Kid and Genshi notions of tagstreams and trees, and follows in footsteps of Chameleon and Kajiki in making the template to compile into Python bytecode directly. The syntax is very close to that of Kajiki, but the internals are very different: Tonnikala writes code as Abstract Syntax Trees and optimizes the resulting trees extensively. In addition, there is an optional speed-up module (currently Python 3 only), that provides a specialized class used for output buffering.

CHAPTER 1

Examples

```
from tonnikala.loader import Loader

template_source = u"""
<table>
    <tr py:for="row in table">
        <py:for each="key, value in row.items()"
            ><td>${key}</td><td>${literal(value)}</td></py:for>
    </tr>
</table>
"""

template = Loader().load_string(template_source)

ctx = {
    'table': [dict(a=1,b=2,c=3,d=4,e=5,f=6,g=7,h=8,i=9,j=10)
        for x in range(1000)]
}

print(template.render(ctx))
```

CHAPTER 2

Template inheritance

2.1 base.tk

```
<html>
<title><py:block name="title_block">I am ${title}</py:block></title>
<py:def function="foo()">I can be overridden too!</py:def>
<h1>${title_block()}</h1>
${foo()}
</html>
```

2.2 child.tk

```
<py:extends href="base.tk">
<py:block name="title_block">But I am ${title} instead</py:block>
<py:def function="foo()">I have overridden the function in parent template</py:def>
</py:extends>
```

CHAPTER 3

Template imports

3.1 importable.tk

```
<html>
<py:def function="foo()">I am an importable function</py:def>
</html>
```

3.2 importer.tk

```
<html>
<py:import href="importable.tk" alias="imp" />
${imp.foo()}
</html>
```

CHAPTER 4

FileLoader

To load templates from files, use the `tonnikala.FileLoader` class:

```
loader = FileLoader(paths=['/path/to/templates'])
template = loader.load('child.tk')
```

A `FileLoader` currently implicitly caches **all** loaded templates in memory.

CHAPTER 5

Template

To render the template:

```
result = template.render(ctx)
```

You can specify a block, or no-argument def to render explicitly:

```
result = template.render(ctx, funcname='title_block')
```

CHAPTER 6

Pyramid integration

Include *'tonnikala.pyramid'* into your config to enable Tonnikala. When included, tonnikala adds the following configuration directives:

`add_tonnikala_extensions(*extensions)` Registers tonnikala renderer for these template extensions. By default Tonnikala is not registered as a renderer for any extension. For example: `config.add_tonnikala_extensions('.html', '.tk')` would enable Tonnikala renderer for templates with either of these extensions.

`add_tonnikala_search_paths(*paths)` Adds the given paths to the end of Tonnikala search paths that are searched for templates. These can be absolute paths, or `package.module:directory/subdirectory`-style asset specs. By default no search path is set (though of course you can use an asset spec for template).

`set_tonnikala_reload(reload)` If `True`, makes Tonnikala not cache templates. Default is `False`.

`set_debug_templates(debug)` If `True`, makes Tonnikala skip some optimizations that make debugging harder.

These 3 can also be controlled by `tonnikala.extensions`, `tonnikala.search_paths` and `tonnikala.reload` respectively in the deployment settings (the `.ini` files). If `tonnikala.reload` is not set, Tonnikala shall follow the `pyramid.reload_templatea` setting.

CHAPTER 7

Status

Beta, working features are

- Structural elements `py:if`, `py:unless`, `py:def`, `py:for`, `py:replace`, `py:content`
- Basic template inheritance: `py:extends` and `py:block`; the child template also inherits top level function declarations from the parent template, and the child can override global functions that the parent defines and uses.
- Expression interpolation using `$simple_identifier` and `${complex + python + "expression"}`
- Boolean attributes: `<tag attr="${False}">`, `<tag attr="$True">`
- Implicit escaping
- Disabling implicit escaping (`literal()`)
- C speedups for both Python 2 and Python 3
- Importing def blocks from another template: `py:import`
- Basic I18N using gettext.
- Pyramid integration
- Javascript as the target language (using `js:` prefix)
- Overriding attributes, setting attrs from dictionary: `py:attrs`
- Understandable exceptions and readable tracebacks on CPython

Upcoming features:

- Structural elements: `py:vars`, `py:switch`, `py:case`; `py:else` for `for`, `if` and `switch`.
- Custom tags mapping to `py:def`
- I18N with optional in-parse-tree localization (partially done)
- Pluggable frontend syntax engines (partially done)
- METAL-like macros
- Pluggable expression languages akin to Chameleon
- Even better template inheritance
- Better documentation

CHAPTER 8

Contents

CHAPTER 9

Indices and tables

- genindex
- modindex
- search