Simplistix

A Journey to a New Text Templating System

Chris Withers

Who am I?

- Chris Withers
- Independent Zope and Python Consultant
- Using Zope and Python since 1999

- 90% of what I do is web-based

- I like things as simple as possible...

  ...but no simpler

Simplistix

chris@simplistix.co.uk
http://www.simplistix.co.uk

Who are you?

- What languages do you develop in?
- What frameworks do you use?
- What systems do you build?

- What text templating systems have you bumped into?

chris@simplistix.co.uk
http://www.simplistix.co.uk

What am I going to cover?

- How I learned to build template-based output
- Why I ended up writing a new templating system
- The problems I still face

chris@simplistix.co.uk
http://www.simplistix.co.uk

A long time ago...

...in a galaxy far far away:

```
class Folder:

    def __init__(self):
        self.pages = []

    def addPage(self,p):
        self.pages.append(p)
        p.folder = self

    def getPages(self):
        return tuple(self.pages)
```

```
class Page:

    status,folder = 'private',None

    def __init__(self,title,content):
        self.title = title
        self.content = content

    def publish(self):
        self.status = 'published'

    def getURL(self):
```

```
folder = Folder()
page1 = Page('This is my page!','This is the content of the page.')
page2 = Page('Page 2','This is the content of page 2.')
page3 = Page('Page 3','This is the content of page 3.')
folder.addPage(page1);folder.addPage(page2);folder.addPage(page3)
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Then came the web!

```
<html>
  <head>
    <title>This is my page!</title>
  </head>
  <body>
  <h1>This is my page!</h1>
  <div id="body">
    <div id="content">
    This is the content of the page.
    </div>
    <div id="tagline">
    This content is <b>published</b>.
    </div>
  </div>
  <ul id="leftnav">
    <li><a href="page1.html" class="selected">This is my page!</a></li>
    <li><a href="page2.html">Page 2</a></li>
    <li><a href="page3.html">Page 3</a></li>
  </ul>
  </body>
</html>
```

dynamic content
common material
page specific

Simplistix

chris@simplistix.co.uk
http://www.simplistix.co.uk

Oh crap, we need to generate that stuff!

```
print '<html>'
print '  <head>'
print '    <title>%s</title>' % page.title
print '  </head>'
print '  <body>'
print '  <h1>%s</h1>' % page.title
print '  <div id="body">'
print '    <div id="content">%s</div>' % page.content
print '    <div id="tagline">'
print '    This content is <b>%s</b>.' % page.status
print '    </div>'
print '  </div>'
print '  <ul id="leftnav">'
for p in page.folder.getPages():
    print '    <li><a href="%s" class="%s">%s</a></li>' % (
            p.getURL(),
            p is page and 'selected' or 'normal',
            p.title
        )
print '  </ul>'
print '  </body>'
print '</html>'
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Python 2.4 string templating

- But we're not using CGI!

```
from string import Template

template = Template('''<html>
  <head>
    <title>$title</title>
  </head>
  <body>
  <h1>$title</h1>
  <div id="body">
    <div id="content">$content</div>
    <div id="tagline">
    This content is <b>$status</b>.
    </div>
  </div>
  <ul id="leftnav">
  $leftnav
  </ul>
  </body>
</html>''')
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Python 2.4 string templating

```
def page_view(page):

    nav = []
    for p in page.folder.getPages():
        nav.append('    <li><a href="%s" class="%s">%s</a></li>' % (
            p.getURL(),
            p is page and 'selected' or 'normal',
            p.title
            )

    return template.substitute(
        'title':page.title,
        'content':page.content,
        'status':page.status,
        'leftnav':'\n'.join(nav),
    )
```

- What about things other than pages?
- How do we get a common look 'n' feel

chris@simplistix.co.uk
http://www.simplistix.co.uk

Web Monkeys

- We are programmers
- We don't *really* like HTML
- We *really* don't like CSS

- Give the problem to someone who cares...

- ...but in way that we can still do our job
- ...and they can do theirs!

chris@simplistix.co.uk
http://www.simplistix.co.uk

What did I learn?

- Zope
- DTML

chris@simplistix.co.uk
http://www.simplistix.co.uk

What is DTML?

- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/DTML.stx

- Actually a generic scripting language
- Highly tied to Acquisition
- Single layered namespace

- “Tell the web monkeys to leave <dtml-*anything*> alone”

chris@simplistix.co.uk
http://www.simplistix.co.uk

Replace

<dtml-var something>

<dtml-var “something + somethingelse”>

chris@simplistix.co.uk
http://www.simplistix.co.uk

Repeat

<dtml-in something>
<dtml-var sequence-item>
</dtml-in>

chris@simplistix.co.uk
http://www.simplistix.co.uk

Conditionals

<dtml-if something>

...

<dtml-elif somethingelse>

...

<dtml-else>

...

</dtml-if>

chris@simplistix.co.uk
http://www.simplistix.co.uk

Variable Definition

```
<dtml-with REQUEST>
  <dtml-var someformvar>
</dtml-with>

<dtml-let something="x + y" >
  <dtml-var something>
</dtml-let>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Common Elements

<dtml-var standard_html_header>

...

<dtml-var standard_html_footer>

chris@simplistix.co.uk
http://www.simplistix.co.uk

And all the rest...

- call
- comment
- mime
- try
- raise
- return
- unless
- sendmail
- sqlgroup
- sqltest
- sqlvar
- tree

chris@simplistix.co.uk
http://www.simplistix.co.uk

Hey Presto!

- We have a whole new programming language

:-(

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Page

```
<dtml-var standard_html_header>
    <div id="content">
    <dtml-var content>
    </div>
    <div id="tagline">
    This content is <b><dtml-var status></b>.
    </div>
<dtml-var standard_html_footer>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

The Template

- standard_html_header

```
<html>
  <head>
    <title><dtml-var title></title>
  </head>
  <body>
  <h1><dtml-var header></h1>
  <div id="body">
```

- standard_html_footer

```
  </div>
  <ul id="leftnav">
    <dtml-in getPages>
    <li><a href="<dtml-var getURL>"
                 <dtml-if "_.getitem('sequence-item') is this()">
                 class="selected"</dtml-if>>
        <dtml-var title></a></li>
    </dtml-in>
  </ul></body></html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Balance Sheet

- Positives
  - can often DWIM
  - simple templates
  - not just XML/HTML
- Negatives
  - Acquisition / One big namespace
  - Funky special variable names
  - Too many knobs to twiddle
  - What about HTML editors?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Zope Page Templaes (ZPT)

- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/ZPT.stx

- Zope templating mk II

- designed for visual html editors
- separation of logic and presentation

- “Tell the web monkeys to leave tal:*anything* alone”

chris@simplistix.co.uk
http://www.simplistix.co.uk

replace

<space tal:replace=”here/title”>a title

a title

<div tal:attributes=”class here/computeClass”>
Some text

chris@simplistix.co.uk
http://www.simplistix.co.uk

repeat

```
<ul>
<li tal:repeat=”item folder/getPages”
    tal:content=”item/title”>
 A menu item
</li>
</ul>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

condition

```
<div tal:condition=”something”>
Something is True!
</div>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

define

```
<ul tal:define=”value something_expensive”>
<li tal:content=”value/x”/>
<li tal:content=”value/y”/>
</ul>

<ul tal:condition=”something”>
<tal:d define=”value something_expensive”>
...
</tal:d>
</ul>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Expression types

- Path
  - tal:something=”x/y/z”
- String
  - tal:something=”string:blah ${x/y/z}”
- python
  - tal:something=”python:x['y'].z()”

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Page

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      metal:use-macro="context/main_template.pt/macros/page">
  <body>
  <metal:x fill-slot="body">
    <div id="content"
         tal:content="context/body">
    This is the content of the page.
    </div>
    <div id="tagline">
    This content is <b tal:content="context/status">published</b>.
    </div>
  </metal:x>
  </body>
</html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

The Template

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      metal:define-macro="page">
  <head>
    <title tal:content="context/title">This is my page!</title>
  </head>
  <body>
  <h1 tal:content="context/title">This is my page!</h1>
  <div id="body"
    <metal:x define-slot="body">
    Body content goes here!
    </metal:x>
  </div>
  <ul id="leftnav">
    <li tal:repeat="page context/folder/getPages">
         <a href="page1.html" class="selected"
            tal:attributes="href  page/getURL;
                            class python:context is page;"
            tal:content="page/title">This is my page!</a></li>
  </ul>
  </body>
</html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Balance Sheet

- Positives
  - clean namespaces
  - harder to introduce business logic
  - source is valid xml/html
- Negatives
  - tied to generating xml/html
  - 2 or 3 new languages
  - doesn't really work with visual editors
  - macros are confusing, especially nesting them!
  - still does way too much

chris@simplistix.co.uk
http://www.simplistix.co.uk

Types of Templating

- Preprocessor
  - DTML
  - Django
  - Cheetah
- Attribute Languages
  - ZPT
  - Nevow
- Class-based
  - PTL
  - Nevow
- DOM-based
  - PyMeld
  - meld2
  - meld3

chris@simplistix.co.uk
http://www.simplistix.co.uk

So what do we actually do?

```
<html>
  <head>
    <title>This is my page!</title>
  </head>
  <body>
  <h1>This is my page!</h1>
  <div id="body">
    <div id="content">
    This is the content of the page.
    </div>
    <div id="tagline">
    This content is <b>published</b>.
    </div>
  </div>
  <ul id="leftnav">
    <li><a href="page1.html" class="selected">This is my page!</a></li>
    <li><a href="page2.html">Page 2</a></li>
    <li><a href="page3.html">Page 3</a></li>
  </ul>
  </body>
</html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

So what do we actually do?

- replace
  - attributes
  - values
  - tags

chris@simplistix.co.uk
http://www.simplistix.co.uk

So what do we actually do?

- repeat
  - tags
  - usually followed by a replace

chris@simplistix.co.uk
http://www.simplistix.co.uk

So what do we actually do?

- remove
  - tags
  - attributes
  - values
  - whole nodes

chris@simplistix.co.uk
http://www.simplistix.co.uk

Anything else?

?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Anything else?

- How do you get hold of the thing you want to work with?

chris@simplistix.co.uk
http://www.simplistix.co.uk

So what if we just did that?

- no new languages
- work with raw html
  - don't change it from html
- as simple as possible
  - but no simpler
- (everything in one file)

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- finding things...
- n = t.getById('something')
- n = t.getByName('something')

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- replace things...

- n.replace(value,tag,**attributes)
  - supplied argument is used
  - all arguments are optional
  - True means “leave as is”
  - False means “remove it”

  - value can be another node

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- repeat things...

- n.repeat(value,tag,**attributes)
  - return the newly inserted node
  - takes the same parameters as replace
    - for convenience

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- remove things...

- n.remove()
  - remove the node from the current twiddler

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- what if I want to keep it all in one place?

- code blocks
  - one per template
  - executed on render
  - executed when explicitly requested

chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- what about reusing common material?

- t.clone()
  - will be cheap
  - allows partial rendering

chris@simplistix.co.uk
http://www.simplistix.co.uk

Back to the original example...

- template

```
<html>
  <head>
    <title name="title">This is my page!</title>
  </head>
  <body>
  <h1 name="h1_title">This is my page!</h1>
  <div id="body">This is the body of this template</div>
  <ul id="leftnav">
    <li name="nav_item">
      <a name="nav_link"
         href="page1.html"
         class="selected">This is my page!</a>
    </li>
  </ul>
  </body>
</html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Back to the original example...

- template code

```
from twiddler.twiddler import Twiddler

source_t = Twiddler('template.html')

def get_template(page):
    t = source_t.clone()
    t.getById('title').replace(page.title)
    t.getById('h1_title').replace(page.title)
    i = t.getByName('nav_item')
    for p in page.folder.getPages():
        n = i.repeat(name=False)
        e = n.getByName('nav_link')
        e.replace(p.title,
                  href=p.getURL(),
                  class_=p is page,
                  name=False)
    return t
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Twiddler

- page

```
<html>
<!--twiddler
def render(t,context,page):
    template = context.get_template(page)
    template.getById('body').replace(t.getById('body'))
    t = template
    t.getById('content').replace(page.content)
    t.getByName('status').replace(page.status)
    return t
-->
  <body>
  <div id="body">
    <div id="content">
    This is the content of the page.
    </div>
    <div id="tagline">
    This content is <b name="status">published</b>.
    </div>
  </div>
  </body>
</html>
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Something cute...

- Repeating column-based table data

```
>>> t = Twiddler('''
... <html>
...   <body>
...     <table>
...       <tr>
...         <th>First Name</th>
...         <td name="firstname">John</td>
...       </tr>
...       <tr>
...         <th>Last Name</th>
...         <td name="lastname">Doe</td>
...       </tr>
...     </table>
...   </body>
... </html>
... ''')
>>> firstname = t.getByName('firstname')
>>> lastname = t.getByName('lastname')
>>> for person in people:
...   firstname.repeat(person.firstname)
...   lastname.repeat(person.surname)
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Balance Sheet

- Positives
  - works with real html
  - no new languages
  - simple as possible
  - explicit
- Negatives
  - not battle-proven
  - slow?

chris@simplistix.co.uk
http://www.simplistix.co.uk

How do I know I got it right?

- meld3
- never saw it while Twiddler was being developed
- scarily similar!

chris@simplistix.co.uk
http://www.simplistix.co.uk

Meld3

- uses 'meld:id' attribute

- clone()
- findmeld(name)
- repeat(childname)
- replace(text,structure=False)
- content(text,structure=False)
- attributes(**kw)

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Real World

- Everything is perfect, right?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Finding Things

- finding things...

- n = t['something']
- n = t.getBy(id='something')

- n = n['something']
- n = n.getBy(name='something')

chris@simplistix.co.uk
http://www.simplistix.co.uk

Replacing

- replace things...

- n.replace(content,
  attributes={'class':'something})

  - True means “leave as is”
  - False means “remove it”
  - value can be another node

chris@simplistix.co.uk
http://www.simplistix.co.uk

Repeating

- repeat things...
- r = n.repeater()
  - returns a repeater that can be called many times
  - remove source element from tree
- n = n.repeat(content,...)
  - return the newly inserted node
  - calls replace with any parameters passed

chris@simplistix.co.uk
http://www.simplistix.co.uk

Code Blocks in Templates

- what if I want to keep it all in one place?
- code blocks are a bad idea...
- now factored out into the input parser..

chris@simplistix.co.uk
http://www.simplistix.co.uk

Nested Data Structures

- n.clone()

```
<a>
   <b />
   <c>
      <e />
      <f />
   </c>
   <d />
</a>
```

```
>>> n = t['parent']
>>> p = n.clone()
>>> stack = [(root,n)]
>>> while stack:
...     d,n = stack.pop(0)
...     n.replace(p.clone(),tag=d.title,id=False)
...     r = n['child'].repeater()
...     for child in d.getChildren():
...         stack.append((child,r.repeat(n)))
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Back to the original example...

- The page:

```
from templates import Master

class PageView:

    template = FileTwiddler('page.html')

    def __init__(self,context,request):
        self.page = context
        self.request = request

    def __call__(self):
        t = Master(page,request)()
        t['body'].replace(self.template['body'])
        t['content'].replace(self.page.content)
        t['status'].replace(self.page.status)
        return t.render()
```

...now wire into your publication process

chris@simplistix.co.uk
http://www.simplistix.co.uk

The Template

```
class Template:

    template = FileTwiddler('template.html')

    def __init__(self,context,request):
        self.context = context
        self.request = request

    def __call__(self):
        t = self.template.clone()
        t['title'].replace(self.context.title)
        t['h1_title'].replace(self.context.title)
        r = t['nav_item'].repeater()
        for p in self.context.folder.getPages():
            n = r.repeat(name=False)
            e = n['nav_link']
            e.replace(p.title,href=p.getURL(),name=False,
                      attributes={'class':p is page})
        return t
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Filtering

- i18n
- html quoting
- ...

```
from i18n import translate
from filters import html_quote

t.setFilters(html_quote)

x.replace(
  someText,
  filters=(translate,html_quote)
  )

y.replace(
  id='<b> something </b>',
  filters=False
  )
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Plugability

- Input Parser
  - plain text
- DOM manipulation
- Output Renderer
  - email

```
To: $to
From: webmaster@example.com
Subject: Order No $order_do

Dear $to,

The following order is on it's way:

<items>$sku $description $quantity $code
</items>

Sincerely,

ExampleCo Order Team
```


chris@simplistix.co.uk
http://www.simplistix.co.uk

Performance

- Why does templating need to be fast?
- What will be fast?
- What will not?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Introspection

- Node introspection
  - attribute values
  - content
    - what renderer to use?
- all indexed values for an attribute
  - possible ids to substitute

chris@simplistix.co.uk
http://www.simplistix.co.uk

Future tweaks...

- Attribute exclusion
  - t:id
- Better packaging
  - eggs
  - cheeseshop
- Advertise?

  ...or just build a framework that uses it?

chris@simplistix.co.uk
http://www.simplistix.co.uk

Thankyou!

- Chris Withers
- chris@simplistix.co.uk
- http://www.simplistix.co.uk/software
- http://www.simplistix.co.uk/presentations