

EmPy, a Python Templating System

**Presentation at EPC 2003
European Python and Zope Conference,
in Charleroi, Belgium, 25-27 June 2003**

Dinu Gherman
dinu @ mac dot com

Background

- General-purpose module/framework for template processing
- Single Python module, ca. 2700 LOC, 93 KB, version 3.0
- Written by Erik Max Francis (Why am I here, then?)
- This presentation uses *EmPy* and *PythonPoint* (pres.xml.em -> pres.xml -> pres.pdf)
- It contains ca. 45 auto-generated slides by accessing EmPy's source code (don't panic! ;-)

EmPy Characteristics

- **Universality:** any text files, not only HTML/XML (even binary files)
- **Simplicity:** one language fits all
- **Orthogonality:** use only features you need
- **Completeness:** provides a *real* language (works on Jython, too!)
- **Configurability:** options, environment, callbacks
- **Responsiveness:** excellent maintenance and support

EmPy Expansions

- **Prefix:** @ (can be changed)
- **Literals:** @, whitespace, closing parantheses, brackets and braces
- **Comments:** reduced rather than expanded
- **Escape codes:** C-/Python-like codes
- **Python code:** any legal Python code (expressions and statements)
- **Markup:** EmPy-Tags for callables, conditionals, iteratives, exceptions, definitions...

In fact: EmPy makes `@("YFML".replace("YFML", "Python"))` your markup language!

EmPy Features

- **Significators:** easy to parse file-level identifiers
- **Diversions:** streams to record and play-back
- **Filters:** dynamic, chainable filtering of streams
- **Hooks:** callbacks for EmPy events
- **Interpreters:** nestable directly or via "empty" pseudomodule

EmPy vs. Preppy

(Sloppy comparison...)

Feature	EmPy	Preppy
Statements	@{...}	{{script}}...{{endscript}}
Evaluations	@(...)	{{eval}}...{{eval}} or {{...}}
Conditionals	@[if ...]...@[end if]	{{if}}...{{endif}}
Loops	@[for ...]...@[end for]	{{for}}...{{endfor}}
	@[while ...]...@[end w.]	{{while}}...{{endwhile}}
Exceptions	@[try ...]...@[except]...	-
Escapes	@\...	-
Definitions	@[def ...]...@[end def]	-
Bytecode output	no	yes

EmPy only: Significators, Diversions, Filters, Hooks, Interpreters

Future

- Improved support for Unicode (whatever that means)
- Many other items on Erik's wishlist...
- More comparisons with "competition" like Preppy, Cheetah, DTML, TAL, YAPTU, Velocity, GNU m4, etc.
- *Wickie*, a minimal, ultra-flexible WikiWiki system on top of *EmPy* and *Docutils* by Dinu Gherman
- An article about *EmPy* and *Wickie* in the August-2003 edition of *Linux-Magazin* (German) by Dinu Gherman

Links

- **EmPy:** <http://www.alcyone.com/pyos/empy>
- **Preppy, PythonPoint:**
<http://www.reportlab.com/download.html>
- **YAPTU:** <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52305>
- **Velocity:** <http://jakarta.apache.org/velocity/>
- **Cheetah:** <http://www.cheetahtemplate.org/>
- **TAL:** <http://www.zope.org/Wikis/DevSite/Projects/ZPT/TAL%20Specification%201.4>
- **GNU m4:**
<http://www.gnu.org/software/m4/m4.html>
- **Wickie:** <http://python.net/~gherman#wickie>

Samples

All following slides were automatically generated using EmPy from code snippets describing test cases like this one for filters:

```
{
    'title': "Filters",

    'input': """\
@{import string}

This line should be in mixed case.
@empy.setFilter(string.upper)@
This line should be all uppercase.
@empy.resetFilter()""",

    'output': """\
This line should be in mixed case.
THIS LINE SHOULD BE ALL UPPERCASE.""",
}
```

Samples Template (Python)

```
@{
for path in ['test_sig0.py', 'test_stmt0.py']: # ...
    title, input, output = tquote(path)

    print '<slide title="%s" outlinelevel="1">\n' % title
    print '<frame %s>\n' % toAttr(frame)
    print '<para style="Heading1">%s</para>' % title

    print '<para style="Heading2">Input</para>'
    tag = ('prefmt', 'para')[input.count('\n')==0]
    format = '<%s style="BigCode">%s</%s>'
    print format % (tag, input, tag)

    print '<para style="Heading2">Output</para>'
    format = '<%s style="BigCode">%s</%s>'
    print format % (tag, output, tag)
    print '</frame>'
    print '</slide>'
}
```

Comments and Literals

Input

```
Sample text @# might need a review
```

```
" " == "@ "  
"@chr(64)" == "@@"  
Closing brackets: "@)", "@]", "@}"
```

Output

```
Sample text  
" " == "  
"@ " == "@ "  
Closing brackets: ")", "]", "}"
```

Escape Codes

Input

Should all be spaces:

```
"  ", "@\s", "@\x20",  
"@\o040", "@\q0200"
```

Output

Should all be spaces:

```
"  ", "  ", "  ",  
"  ", "  "
```

Expressions 1

Input

```
1 == @(1)
'x' == @repr("x")
#globals is @len(globals()).
```

Output

```
1 == 1
'x' == 'x'
#globals is 17.
```

Expressions 2

Input

```
@{  
x = 4; s = 'alpha'; l = [3, 2, 1]  
def square(n):  
    return n**2  
}  
x is @x, l is @l, s is "@s",  
and @x squared is @square(x).
```

Output

```
x is 4, l is [3, 2, 1], s is "alpha",  
and 4 squared is 16.
```

Statements

Input

```
@{
import time, urllib
url = "http://www.alcyone.com/"
url += "pyos/empy/version.txt"
try: ver = urllib.urlopen(url).read()
except IOError: ver = '?'
}
Current EmPy version:
@ver.strip() (@time.ctime())
```

Output

```
Current EmPy version:
3.0.2 (Sun Jun 29 20:45:16 2003)
```

Significators

Input

```
@%a
```

```
@%b "x"
```

```
a should be None: @`__a__`,  
b should be 'x': @`__b__`
```

Output

```
a should be None: None,  
b should be 'x': 'x'
```

Diversions

Input

```
foo @
@empy.startDiversion(1)
This text is diverted. @
@empy.stopDiverting()
bar @
@empy.playDiversion(1)
```

Output

```
foo
bar
This text is diverted.
```

Filters

Input

```
@{import string}
```

```
This line should be in mixed case.
```

```
@empty.setFilter(string.upper)@
```

```
This line should be all uppercase.
```

```
@empty.resetFilter()
```

Output

```
This line should be in mixed case.
```

```
THIS LINE SHOULD BE ALL UPPERCASE.
```

Hooks

Input

```
@{
def h(intp, kwds):
    expr = kwds['expression']
    intp.write("# %s\n" % expr)
}
@empy.addHook('before_evaluate', h)
foo @(0==1) bar
@empy.clearAllHooks()
```

Output

```
foo # 0==1
0 bar
# empy.clearAllHooks()
```

Embedded Interpreters

Input

```
@{  
import em  
intp = em.Interpreter()  
intp.string("@{x = 123}")  
x = intp.globals['x']  
intp.shutdown()  
}
```

@x

Output

123

Loops

Input

```
@{list = [0, 1, 2, 3]}
```

```
@[for i in list]@i @[end for]
```

```
@[while list]@
```

```
@list[0] @{del list[0]}@
```

```
@[end while]
```

Output

```
0 1 2 3
```

```
0 1 2 3
```

Exceptions

Input

```
@{list = [2, 1, 0]}
```

```
@[try]@list[0]
```

```
@[except IndexError]NaN
```

```
@[end try] @
```

```
@[try]@list[5]@[except]NaN@[end try]
```

Output

2

NaN

Appendix

All following slides were created automatically from code introspection into `em.py` module!

```
ESCAPE_INFO = [  
    ("@\\0", "NUL, null"),  
    ("@\\a", "BEL, bell"),  
    ("@\\b", "BS, backspace"),  
    ("@\\dDDD", "three-digital decimal code DDD"),  
    ("@\\e", "ESC, escape"),  
    ("@\\f", "FF, form feed"),  
    ("@\\h", "DEL, delete"),  
    ("@\\n", "LF, linefeed, newline"),  
    ("@\\oOOO", "three-digit octal code OOO"),  
    ("@\\qQQQQ", "four-digit quaternary code QQQQ"),  
    ("@\\r", "CR, carriage return"),  
    ("@\\s", "SP, space"),  
    ("@\\t", "HT, horizontal tab"),  
    ("@\\v", "VT, vertical tab"),  
    ("@\\xHH", "two-digit hexadecimal code HH"),  
    ("@\\z", "EOT, end of transmission"),  
]
```

Appendix Template (EmPy)

```
@{import em; metaInfo = (("Options", em.OPTION_INFO), )}

@[for title, info in metaInfo]
  @[for i in range(len(info))]
    @{j = i%maxEn} @
    @[if j==0] @
      @{k=[len(info)-1,i+maxEn][(i+maxEn)<len(info)]}
      @{nav = map(lambda n:info[n][0], range(i,k+1))}
      @{nav = ' / '.join(nav)}
      <slide title="@title @(i/maxEn): @(nav)" outlinelevel="1">
        <frame @(toAttr(frame))>
          <para style="Heading1">@
            @title Info @[if i>0] (cont@\x27ed) @[end if]</para>
          @[end if]
          <para style="Heading3">@format(info[i][0])</para>
          <para style="Normal">@format(info[i][1])</para>
          @[if j==maxEn-1 or (i==len(info)-1 and 0<j<maxEn)] @
            </frame>
          </slide>
        @[end if]
      @[end for]
    @[end for]
```

Options Info

-V --version

Print version and exit

-h --help

Print usage and exit

-H --extended-help

Print extended usage and exit

-k --suppress-errors

Do not exit on errors; continue interactively

Options Info (cont'ed)

-p --prefix=

Change prefix to something other than @

-m --module=

Change the internal pseudomodule name

-f --flatten

Flatten the members of pseudomodule to start

-r --raw-errors

Show raw Python errors

Options Info (cont'ed)

-i --interactive

Go into interactive mode after processing

-n --no-override-stdout

Do not override sys.stdout with proxy

-o --output=

Specify file for output as write

-a --append=

Specify file for output as append

Options Info (cont'ed)

-b --buffered-output

Fully buffer output (even open), before -o or -a

--binary

Treat the file as a binary

--chunk-size=

Use this chunk size for reading binaries

-P --preprocess=

Interpret EmPy file before main processing

Options Info (cont'ed)

-I --import=

Import Python modules before processing

-D --define=

Execute Python assignment statement

-E --execute=

Execute Python statement before processing

-F --execute-file=

Execute Python file before processing

Environment Info

EMPY_OPTIONS

Specified options will be included

EMPY_PREFIX

Specify the default prefix: -p

EMPY_PSEUDO

Specify name of pseudomodule: -m

EMPY_FLATTEN

Flatten empy pseudomodule if defined: -f

Environment Info (cont'ed)

EMPY_RAW_ERRORS

Show raw errors if defined: -r

EMPY_INTERACTIVE

Enter interactive mode if defined: -i

EMPY_BUFFERED_OUTPUT

Fully buffered output if defined: -b

EMPY_NO_OVERRIDE

Do not override sys.stdout if defined: -n

Markup Info

@# ... NL

Comment; remove everything up to newline

@ WHITESPACE

Remove following whitespace; line continuation

@ \ ESCAPE_CODE

A C-style escape sequence

@ @

Literal @; @ is escaped (duplicated prefix)

Markup Info (cont'ed)

@), @], @}

Literal close parentheses, brackets, braces

@(*EXPRESSION*)

Evaluate expression and substitute with str

@(*TEST* [*? THEN* [*! ELSE*]])

If test is true, evaluate then, otherwise else

@(*TRY* \$ *CATCH*)

Expand try expression, or catch if it raises

Markup Info (cont'ed)

@ *SIMPLE_EXPRESSION*

Evaluate simple expression and substitute; e.g., @x, @x.y, @f(a, b), @l[i], etc.

@` *EXPRESSION* `

Evaluate expression and substitute with repr

@: *EXPRESSION* : [*DUMMY*] :

Evaluates to @:....:expansion:

@{ *STATEMENTS* }

Statements are executed for side effects

Markup Info (cont'ed)

@[CONTROL]

Control markups: if E; elif E; for N in E; while E; try; except E, N; finally; continue; break; end X

@%% KEY WHITESPACE VALUE NL

Significator form of `__KEY__ = VALUE`

Escapes Info

@\0

NUL, null

@\a

BEL, bell

@\b

BS, backspace

@\dDDD

three-digital decimal code DDD

Escapes Info (cont'ed)

@le

ESC, escape

@lf

FF, form feed

@lh

DEL, delete

@ln

LF, linefeed, newline

Escapes Info (cont'ed)

@l000

three-digit octal code 000

@lqQQQQ

four-digit quaternary code QQQQ

@v

CR, carriage return

@ls

SP, space

Escapes Info (cont'ed)

@lt

HT, horizontal tab

@lv

VT, vertical tab

@\xHH

two-digit hexadecimal code HH

@\z

EOT, end of transmission

Hooks Info

at_shutdown

Interpreter is shutting down

at_handle

Exception is being handled (not thrown)

before_include

empy.include is starting to execute

after_include

empy.include is finished executing

Hooks Info (cont'ed)

before_expand

empy.expand is starting to execute

after_expand

empy.expand is finished executing

at_quote

empy.quote is executing

before_file

A file-like object is will be processed

Hooks Info (cont'ed)

after_file

A file-like object is finished processing

before_binary

A binary file-like object is will processed

after_binary

A binary file-like object is finished processing

before_string

A standalone string is will be processed

Hooks Info (cont'ed)

after_string

A standalone string is finished processing

at_parse

A parsing pass is being performed

before_evaluate

A low-level evaluation is will be done

after_evaluate

A low-level evaluation has just finished

Hooks Info (cont'ed)

before_execute

A low-level execution is will be done

after_execute

A low-level execution has just finished

before_significate

A significator is will be processed

after_significate

A significator has just finished processing

Pseudomodule Info

VERSION

String representing EmPy version

SIGNIFICATOR_RE_STRING

Regular expression string matching signifiers

SIGNIFICATOR_RE_SUFFIX

The above stub, lacking the prefix

interpreter

Currently-executing interpreter instance

Pseudomodule Info (cont'ed)

argv

The EmPy script name and command line arguments

args

The command line arguments only

identify()

Identify top context as name, line

setName(name)

Set the name of the current context

Pseudomodule Info (cont'ed)

setLine(line)

Set the line number of the current context

atExit(callable)

Invoke no-argument function at shutdown

getGlobals()

Retrieve this interpreter's globals

setGlobals(dict)

Set this interpreter's globals

Pseudomodule Info (cont'ed)

updateGlobals(dict)

Merge dictionary into interpreter's globals

clearGlobals()

Start globals over anew

evaluate(expression, [loc])

Evaluate the expression

serialize(expression, [loc])

Evaluate and serialize the expression

Pseudomodule Info (cont'ed)

execute(statements, [loc])

Execute the statements

single(source, [loc])

Execute the 'single' object

atomic(name, value, [loc])

Perform an atomic assignment

assign(name, value, [loc])

Perform an assignment

Pseudomodule Info (cont'ed)

significate(key, [value])

Significate the given key, value pair

include(file, [loc])

Include filename or file-like object

expand(string, [loc])

Explicitly expand string and return

string(data, [name], [loc])

Process string-like object

Pseudomodule Info (cont'ed)

quote(string)

Quote prefixes in provided string and return

flatten([keys])

Flatten module contents into globals namespace

getPrefix()

Get current prefix

setPrefix(char)

Set new prefix

Pseudomodule Info (cont'ed)

stopDiverting()

Stop diverting; data now sent directly to output

createDiversion(name)

Create a diversion but do not divert to it

retrieveDiversion(name)

Retrieve the actual named diversion object

startDiversion(name)

Start diverting to given diversion

Pseudomodule Info (cont'ed)

playDiversion(name)

Recall diversion and then eliminate it

replayDiversion(name)

Recall diversion but retain it

purgeDiversion(name)

Erase diversion

playAllDiversions()

Stop diverting and play all diversions in order

Pseudomodule Info (cont'ed)

replayAllDiversions()

Stop diverting and replay all diversions

purgeAllDiversions()

Stop diverting and purge all diversions

getFilter()

Get current filter

resetFilter()

Reset filter; no filtering

Pseudomodule Info (cont'ed)

nullFilter()

Install null filter

setFilter(shortcut)

Install new filter or filter chain

attachFilter(shortcut)

Attach single filter to end of current chain

enableHooks()

Enable hooks (default)

Pseudomodule Info (cont'ed)

disableHooks()

Disable hook invocation

areHooksEnabled()

Return whether or not hooks are enabled

getHooks(name)

Get the list of hooks with name

clearHooks(name)

Clear all hooks with name

Pseudomodule Info (cont'ed)

clearAllHooks()

Clear absolutely all hooks

addHook(name, hook, [i])

Register hook with name (optionally insert)

removeHook(name, hook)

Remove hook from name

invokeHook(name_, ...)

Manually invoke hook with name

Pseudomodule Info (cont'ed)

Interpreter

The interpreter class

Filter

The base class for custom filters

NullFilter

A filter which never outputs anything

FunctionFilter

A filter which calls a function

Pseudomodule Info (cont'ed)

StringFilter

A filter which uses a translation mapping

BufferedFilter

A buffered filter (and base class)

SizeBufferedFilter

A filter which buffers data into fixed chunks

LineBufferedFilter

A filter which buffers by lines

Pseudomodule Info (cont'ed)

MaximallyBufferedFilter

A filter which buffers everything until close