EmPy Template for Do-It-Yourself Wiki

Happy-go-Wiki

Python fans will appreciate EmPy, a well designed template processor for web-based or other development. Its powerful features allowed the author to program a Wiki software project that had only a four KByte footprint. The small size allows us more easily to follow just what is going on and also aids in quickly experimenting with it.

BY DINU GHERMAN

In the world of Open Source software, some things seem to happen more-or-less of their own volition. Or how else would you explain the fact that an unsuspecting developer, sitting in front of his computer, might download the latest version of a software library, and suddenly discover that he had developed yet another version of a common tool, before even thinking about its possible usefulness?.

This is exactly what happened in the case of a WikiWiki program, that automagically appeared on my computer while I was pondering the issue of a useful application for the "EmPy" template processor.

Now, you may well ask whether the world really needs yet another WikiWiki program, or even another template processor for that matter. Wouldn't it be more useful to join the fight against ignorance and famine all over the world?

Admittedly, WikiWikis are simpler to manage, but a specimen with a four KByte footprint does make you wonder. Of course, this is only the core of the WikiWiki, but it might at least give you a chance to understand how a Wiki works; and that can't be said of most existing Wikis, not to mention ignorance, intolerance

and famine that we find in the world at large.

Anyway, the program we will be looking at in this article, HeyHeyWickie [2], which incidentally was named after a friendly Scandinavian cartoon character (Vicky the Viking), is remarkable in one respect: Its capability to execute arbitrary

source code demonstrates unprecedented flexibility, which can be an issue in itself, but we will get back to that subject later.

Templates

The reason for this flexibility is the underlying template processor. Nowadays, processors of this kind are two-a-penny. What they all have in common is the fact that they accept text input, recognize the templates in that input, and replace them with different content before outputting a typically larger file generated in the process. Replacements typically involve expanding simple strings or iterating against some kind of data described in the template.

Processors not only use different APIs, but often define a kind of mini-language for their templates, as if there weren't already an over-abundance of artificial, rather than natural, languages today. Some of these template processors are so closely knit-in to the (typically Web-based) application for which they were originally developed that they will not work without it.

This can't be said of EmPy [3], however, as it uses the same universal programming language in which it was written,

Listing 1: "RecentChanges" Page

```
01 @{
02 import wickie4K
03 from os.path import getmtime
04 files = wickie4K.getWikiPaths()
05 files = map(lambda f:(getmtime(f), f), files)
06 files.sort()
07 files.reverse()
08 files = map(lambda f:f[1], files)
09 files = map(wickie4K.wikiPath2name, files)
10 files = map(lambda f:f[0], files)
11 }
12
13 This is currently a list of all pages with
14 the most recently changed ones at the top:
15
16 @[for f in files]
17   - @wickie4K.wikiName2link(f)
18 @[end for]
```

Listing 2: Code Snippet for Quotes:

```
01 @{
02 import random, wickie4K
03 path = wickie4K.WIKI_DIR + "quotes.text"
04 quotes = open(path).read().split('\n')
05 print "*%s*" % random.choice(quotes)
06 }
```
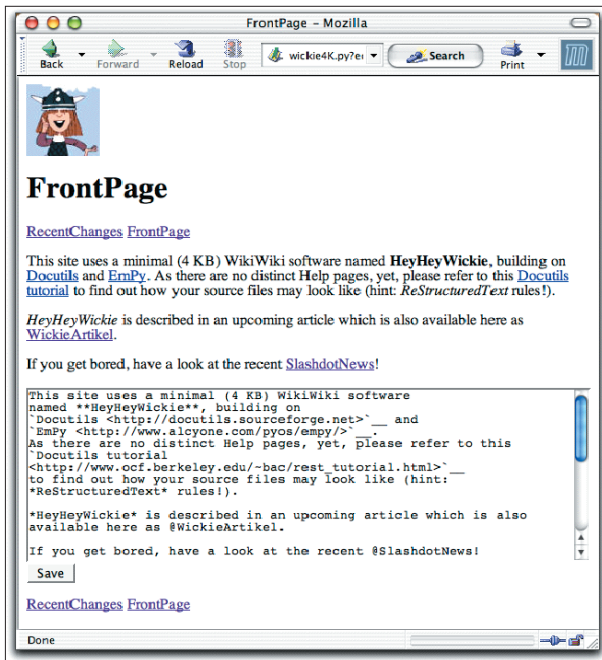
Figure 1: This is what the minimalist Wiki looks like – and you really can use it to create Wiki content. Figure 2 shows what "Slashdot News" is all about

Python [4]. The processor was designed without a specific application in mind and stands out from the crowd by virtue of its comprehensiveness and versatility. It can run arbitrary source code from text or even binary files. In addition to tags used to mark up raw code, EmPy also uses tags that map to individual language constructs, such as iterations or exception handling, the latter alone distinguishing it from most other template systems.

EmPy is extremely well documented and easy to install, so we will not be looking into the installation, or features, at any great length. A presentation on these topics, which was created using (guess what) EmPy, was given at the European Python and Zope Conference, EPC2003, and is available online from [5].

Instead, we will be looking at EmPy using the Wiki application mentioned at the start of this article as an example. A word of caution before we continue: The project is still under development and future versions can be expected to have a slightly larger footprint than the current, less than 4000 Bytes.

At the Heart of the Wiki

HeyHeyWickie itself is a simple CGI program. It accesses a special directory (normally below the local Apache installation) that contains all the files that are used to build the WikiWiki Websites. HeyHeyWickie uses file extensions to recognize appropriate file types, for example *FrontPage. txt*, and then loads the file, using EmPy to process it and Docutils [6] to convert it to HTML. Filenames are mapped to WikiWiki names (that is, simply the filename without the extension in this case) which can then be placed on arbitrary pages.

EmPy and Docutils fulfill different roles here, although both of them process the source file. First, EmPy parses the file content, and this causes any dynamic code elements the file may contain to be executed, including expanding WikiWiki names to clickable hyperlinks. HeyHeyWickie expects a prepended @ in this case.

Docutils has the task of converting the text (loaded statically from a file and/or dynamically modified by EmPy) to HTML; the text uses a kind of enhanced, but readable ASCII format called ReStructuredText (ReST), where elements such as headings, lists, hyperlinks or tables are tagged in a kind of two dimensional markup. ReST can do a lot more, as clearly indicated by the tutorial found at [7] and specification found at [8].

The *wickie4K.py* CGI program (which dropped the HeyHey prefix for brevity's sake) basically comprises a few constants, which specify the WikiWiki file directory, for example, and a few snippets of HTML which occur as header and footer lines on every page (it might be preferable to import these from a different module to provide more output flexibility).

Additionally, there are a few short functions that basically translate between filenames, WikiWiki names and hyperlinks, load files and use EmPy and Docutils to process their content, and finally pass the results to the browser after adding headers and footers. The main program takes care of various CGI actions, such as the displaying and editing of existing pages and the storing of new pages.

Total Flexibility

So far we have been looking at a neat, compact system that simply uses EmPy

Listing 3: Code for "SlashdotNews"

```
01 @{
02 import time, urllib, xml2obj
03
04 def printHeadlineLinks(root):
05   for story in root.children:
06     for c in story.children:
07       name = c.name
08       if name == "title":
09         title = c.getData()
10       elif name == "url":
11         url = c.getData()
12     print "- `%s <%s>`__ \n" %
  (title, url)
13
14 url = "http://slashdot.org/
  slashdot.xml"
15 try:
16   xmlData = urllib.urlopen
  (url).read()
17   parser = xml2obj.Xml2Obj()
18   root = parser.Parse
    String(xmlData)
19 except IOError:
20   root = None
21 }
22
23 @[if root]
24 Below you can see the list of
   the very current Slashdot news
25 (as of @time.ctime()).
26 (Don't reload this too often
   or Slashdot might temporarily
27 ban your IP!)
28
29 @printHeadlineLinks(root)
30 @[else]
31 Sorry, but there seems to be
   no Internet connectivity!
32 @[end if]
33
```

to link WikiWiki pages and Docutils to display and create them, as shown in Figure 1. But the program derives its flexibility from the fact that the page can contain arbitrary source code which need not be embedded in the main program. We will be looking at a few examples in this article.

One of the typical pages that every WikiWiki system uses, is the "RecentChanges" page, which is generated dynamically and informs users about recent page edits. In the case of HeyHeyWickie this is just a page, like any other. It creates a header and footer containing a link to the current page for any other pages, but is otherwise unaware of the current page's special function.

In order to create a list of recent changes, you first need to know what files exist in the WikiWiki directory; you can then sort them by timestamp and list them in the order of most recent edits. The code in Listing 1 does exactly that. First, a string of Python commands generates a sorted list of WikiWiki names from the source files stored in the Wiki-Wiki directory. This list includes the RecentChanges.txt file itself which is shown as RecentChanges.

The next line is an explanation which occurs at the start of the page before a few EmPy tags are iterated against the

list to display each WikiWiki name as a URL hyperlink. The ReST format is used to generate the list; Docutils then looks for pipe signs in the results, which it uses to delimit the column entries in a neat HTML-based list.

Quotes

Quotes are another potential enhancement; some people not only add them to email messages, but also to presentation slides (why not, if it stops people dropping off during your talk?). A mere four lines of code are required to implement this feature (of course you could improve efficiency by using a lot more code, but this does work). Enclosed in an EmPy

Listing 4: wickie4K.py

```
001 #!/usr/bin/env python
002
003 from os.path import join,
    basename, dirname, \
004     splitext, exists
005 import glob, sys, re
006 import cgi, cgitb
007
008 from docutils.core import
    publish_string
009 import em
010
011
012 __version__ = '0.1.0'
013 __license__ = 'GPL'
014 __author__ = 'Dinu Gherman'
015
016 PROG = basename(sys.argv[0])
017 WIKI_DIR = "/Library/Web
    Server/Documents/wickiedata/"
018 WIKI_EXT = ".txt"
019 PAGES = {}
020
021
022 # HTML snippets (should go
    into some styles module)
023
024 HEADER = """\
025 <html>
026 <head>
027 <title>%%(name)s</title>
028 </head>
029 <body>
030 <img src="%(image)s">
031 <h1>%%(name)s</h1>
032 """ % {'image':
    'file://'+join(WIKI_DIR,
    "wickie.jpg")}
033
034 EDIT_THIS = '\n<a
    href="%(PROG)s?edit=%(name)s"
    >Edit</a>\n'
035
036 BUTTONS = """\
037 <a
    href="%(PROG)s?show=RecentCha
    nges">RecentChanges</a>
038 <a
    href="%(PROG)s?show=FrontPage
    ">FrontPage</a>
039 """
040
041 EDIT_AREA = """\
042 <form action="%(PROG)s"
    method="post">
043 <textarea name="area"
    cols="80" rows="20" \
044 wrap="virtual"
    style="width:100%%">
045 %(content)s
046 </textarea>
047 <br/>
048 <input type="hidden"
    name="save" value="%(name)s">
049 <input type="submit"
    value="Save">
050 </form>
051 """
052
053 FOOTER =
    "\n</body>\n</html>\n"
054
055 NEW_PAGE = "Please edit some
    content for this new page \
056 in order to save it!"
057
058
059 def getWikiPaths():
060     pat = join(WIKI_DIR, "*"
    + WIKI_EXT)
061     return glob.glob(pat)
062
063 def wikiPath2name(path):
064     base = basename(path)
065     return splitext(base)[0],
    exists(path)
066
067 def wikiName2path(name):
068     base = name + WIKI_EXT
069     path = join(WIKI_DIR,
    base)
070     return path, exists(path)
071
072 def wikiName2link(name,
    action="show"):
073     anchor = name
074     if action == "new":
    anchor = anchor + '?'
075     return '`%s
    <%s?%s=%s>`__' % (anchor,
    PROG, action, name)
076
077 def makeButtonsLine(path,
    edit):
078     name, ex =
    wikiPath2name(path)
079     d = {'name': name,
    'PROG': PROG}
080     s = "\n<p>\n"
081     if not edit: s = s +
    EDIT_THIS % d
082     s = s + BUTTONS % d +
    "</p>\n"
083     return s
084
085 def loadPage(path):
086     name, ex =
```

tag for arbitrary instructions, @{?},… this takes the form shown in Listing 2:

If this code occurs somewhere on a source page (and the matching *quotes.text* file with one quote per line is also supplied), you can look forward to undying perceptions such as: "Whenever you find yourself on the side of the majority, it's time to pause and reflect. (Mark Twain)"

Let's leave other typical extensions, such as including the last modification time for a page, or even displaying a hit statistic, or email notification in case of pages modifications, to more adventurous readers (the standard Python library provides everything you need).

The main thing to look into when planning an extension is where to place it: on the page itself or in the main program?

News for Net Junkies

If what you have read so far has not exactly had you on the edge of your seat, read on. There are far wackier applications just around the corner. For example, you could create a page called "SlashdotNews" that contains a code snippet designed to download an RDF file and display the article as an up-to-date link list, using a similar technique to that used by the "RecentPages" page.

Listing 3 contains the complete listing

for the "SlashdotNews" page. It uses an extremely lean module, called xml2obj, which John Bair created to handle simple conversions from XML to Python objects, and kindly deposited in the ActiveState collection, which can be found at [9].

Figure 2 shows what the resulting page will look like (assuming you have a connection to the Internet, of course).

Security Issues

All this flexibility can be too much of a good thing, particularly if other people that you cannot trust have access to it, and especially not if it happens to be on your own machine. As HeyHeyWickie

Listing 4: wickie4K.py

```
        wikiPath2name(path)
 087     if ex: return
        open(path).read()
 088     return NEW_PAGE
 089
 090 def empyfy(text):
 091     wikiNames =
        re.findall("@[a-zA-z0-9_]+",
        text)
 092     wikiNames = map(lambda
        name:name[1:], wikiNames)
 093     wikiNames = filter(lambda
        name:
 094                         name
        not in PAGES.keys(),
        wikiNames)
 095     for name in wikiNames:
 096         PAGES[name] =
        wikiName2link(name,
        action="new")
 097     return em.expand(text,
        PAGES)
 098
 099 def reSTify(content):
 100     settings =
        {'input_encoding': 'latin-1',
 101
        'output_encoding': 'latin-1'}
 102     rest =
        publish_string(content,
        writer_name='html',
 103
        settings_overrides=settings)
 104     start =
        rest.find("<body>")
 105     p =
        rest[start+len("<body>"):].fi
        nd(">")
 106     snip =
```
```
        rest[start+len("<body>")+p+2:
        -23] # magic
 107     snip =
        snip.replace("&#64;", "@") #
        hack
 108     return snip
 109
 110 def addPageNames():
 111     for path in
        getWikiPaths():
 112         name, ex =
        wikiPath2name(path)
 113         if ex:
 114             link =
        wikiName2link(name)
 115             PAGES[name] =
        link
 116
 117 def outputHTML(name, path,
        edit, html, textarea=""):
 118     out = sys.stdout.write
 119     out("Content-type:
        text/html\n\n")
 120     out(HEADER % {'name':
        name})
 121     out(makeButtonsLine(path,
        edit))
 122     out(html)
 123     if textarea:
        out(textarea)
 124     out(makeButtonsLine(path,
        edit))
 125     out(FOOTER)
 126
 127 def mainCGI():
 128     cgitb.enable()
 129
 130     query =
        cgi.FieldStorage()
```
```
 131     qgetval = lambda n:
        query.getvalue(n, '')
 132     show, edit, save, new = \
 133         map(qgetval, "show
        edit save new".split())
 134     name = show or edit or
        save or new or "FrontPage"
 135     path, ex =
        wikiName2path(name)
 136     textarea = ''
 137
 138     page = loadPage(path)
 139
 140     if edit:
 141         args = {'PROG': PROG,
 142             'content':
        loadPage(path),
 143             'name':
        wikiPath2name(path)[0]}
 144         textarea = EDIT_AREA
        % args
 145     elif save:
 146         area =
        qgetval('area')
 147         open(path,
        'w').write(area)
 148
 149     addPageNames()
 150     mpPage = empyfy(page)
 151     html = reSTify(mpPage)
 152     outputHTML(name, path,
        edit, html, textarea)
 153
 154 if __name__ == '__main__':
 155     mainCGI()
```

can execute arbitrary source code on any page, the possible consequences should come as no surprise if you simply dump the program on the Internet. This leaves you with the typical sandbox issue: How can I create a "secure" environment to restrict the capabilities of the tool?

The easiest way to go, is to require authentication for the program (as provided by Apache, for example), allowing access only to a select group of users. But this puts you at the mercy of the individual members, or to be more precise, their capability to use their access credentials sensibly. Individual users could delete files inadvertently. And in many cases, enforcing registration for an application or the users does not make sense

So what exactly could happen? For one thing, malevolent code might be executed causing file deletions or installing viruses. Or it might be a case of file spying and forwarding. It is quite trivial to prevent write or read access, by running a program like HeyHeyWickie with a non-privileged user account and assigning rights only to specific directories.

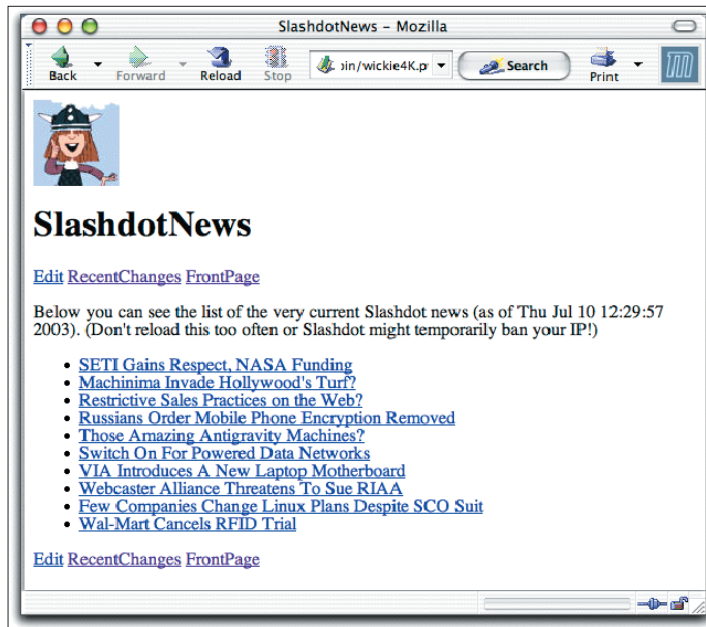The Python interpreter used the Bastion and Rexec (restricted execution)

Figure 2: Slashdot News as an "Add-on" to the minimalist Wiki. Listing 3 contains the complete code

modules to apply artificial restrictions of this kind, but these modules were dropped in Python 2.3, as they were considered insecure. At the moment, there is no way to run untrusted Python code in a sandbox using the standard interpreter.

Overly adventurous readers might consider running their own interpreter in a hostile context of this kind, allowing them to restrict the privileges of specific modules as required. Of course, you could choose this option, but there is another source of potential issues, such as infinite loops or other resource consuming processes that are not easily recognized from the "inside". In cases like these, it makes more sense to ask the operating system, rather than the interpreter, to take care of the issue – after all it is the operating system's job to manage process resources, such as CPU cycles and memory usage.

Hey, Wickie, where you goin'?

What started off more-or-less as a just-for-fun programming exercise has now turned into an interesting experiment in minimalist WikiWiki development. A neat combination of the assets of Python, as an all-round programming language, and EmPy and Docutils, as template processors for various tasks, ensures extremely flexible applications. Thanks to the minimal code base, it is quite easy to understand the basic workings of this

kind of system, and this in turn facilitates customization and extensions.

However, a high degree of flexibility does imply some risk, particularly in an environment that provides unrestricted access to users on the Internet. In addition to the security risks, there may be some scalability and efficiency issues which, however, are beyond the scope of this article. You might well be asking yourself where you could usefully deploy a tool like HeyHeyWickie, despite all of its flexibility.

Besides its primary role as an experimental platform for WikiWiki systems and the EmPy processor, there is a great deal of scope for developers to put their ideas into practice. Another small group of developers will be using the tool to develop the documentation for another program library in the near future. The continuing success story of "ReStructuredText" as a near-standard format in other Wiki systems, such as ZWiki [10], and the fact that HTML, LaTeX and PDF converters exist, is one practical aspect that speaks in favor of the tool. Due to security concerns, the current extended version of HeyHeyWickie provides facilities for capturing static snapshots of WikiWiki pages, and storing them as HTML pages, which can in turn be served up on the Internet, to mitigate the risk of defacement attacks.

You might even consider using this extended version as a kind of personal, lightweight content management system, a URL manager, or scrapbook with an interface to remote services on the Internet, just like the Slashdot example. ∎

INFO
[1] WikiPedia: *http://www.wikipedia.org*
[2] HeyHeyWickie: *http://python.net/ ~gherman/#heyheywickie*
[3] EmPy: *http://www.alcyone.com/pyos/empy*
[4] Python: *http://www.python.org*
[5] EmPy Presentation: *http://europython. org/Talks/Slides/empy.pdf*
[6] Docutils: *http://docutils.sourceforge.net*
[7] Docutils Tutorial: *http://www.ocf.berkeley. edu/~bac/rest_tutorial.html*
[8] ReST Specification: *http://docutils.sourceforge.net/spec/rst/ reStructuredText.html*
[9] Xml2obj: *http://aspn.activestate.com/ ASPN/Cookbook/Python/Recipe/149368*
[10] ZWiki: *http://www.zwiki.org*

THE AUTHOR

Dinu Gherman is a freelance IT developer, consultant, author, and translator. He has translated four books on the subject of Python. His Python Website [2] offers a whole bunch of interesting projects besides Wickie.