TeX and Scripting Languages

William M. Richter
Texas Life Insurance Company
Bill Clinton: Eat your heart out...
This is MY LIFE!
Painting, Hacking and TeX --- Say what??

- Paul Graham - "Hackers and Painters"
- Painting and writing are "evolutionary"
- Hacking: Also an evolutionary process
- TeX/Authoring: Yet another evolutionary process
- TeX + Authoring + Hacking = ???
Definition of “Hacking”

- From Eric Raymond's “Hacking Folklore”: Hacking has several meanings:
  - Appropriate application of ingenuity
    - Could be in a quick-and-dirty patchwork job,
    - Or in a carefully crafted work of art
  - A creative practical joke
- Today we're interested in the first definition. The practical joke will have to wait for another day :(
The Recipe:

- Mix equal parts hacking and TeX'ing, stir well...
WANTED:

A programming language that allows for the “appropriate application” of ingenious algorithms.

--or--

A programming language that allows us to HACK.
Properties of our desired language:

- Simple syntax
- Standard control structures
- Embeddable in other systems
- High level data types
- Malleable
- Plays well with other entities (i.e. TeX)
Scripting Languages

- Depart from traditional edit / compile / link / test “cycle-of-pain”
- Just edit and run
- High level data types:
  - Lists, Tuples, Dictionaries, etc.
- Object-oriented
- Clean, readable syntax
- Dynamic variables / Loose type checking
Welcome to the scripting language Zoo

- Perl
- Python
- Tcl/Tk
- JavaScript
- Rebol
- Bash
- Awk
- PHP
- Ruby
- Small
- Groovy
- Lua
Scripting Languages go prime time

- Google search of “scripting language”: returns 1,570,000 hits
- Many have evolved past original origins to become general-purpose languages
- Only reason to continue calling them “scripting languages” is lack of a better term -- ESR
Popular Scripting Languages

- Perl - www.perl.org
- Python - www.python.org
- PHP - www.php.net
- Ruby - www.ruby-lang.org
We chose Python because:

- Simple syntax
- Standard control structures
- Embeddable
- Object-oriented to the bone
- Elegant internal design
- High level data types
- Malleable
- Plays well with external entities (i.e. TeX)
- Highly Extendable
Combining Python and TeX

- **Application Domains:** Thinking about Python / TeX integration:
  - Python does most of the work
  - TeX does most of the work
  - Cooperative effort
Four approaches to Python / TeX integration

- Imperative
- Form-based paradigm (TeXmerge)
- Tricks with TeX macros
- Hybrid: Imperative + TeXmerge
The imperative technique:

Example: A trivial script that creates a .tex file, runs it through TeX, and Dvips.

```python
!/usr/bin/env python
import sys
import os
f = open('MyDocument.tex', 'w')
f.write('\nopagenumbers\n')
f.write('This is my first \TeX\ document \n   produced from a script.\n')
f.write('\vfil\eject\bye\n')
f.close()
os.system('tex MyDocument.tex')
os.system('dvips MyDocument')
print 'Done.'
```
About the imperative way...

• Simplest approach to combining Python / TeX

• Development emphasis is on the Python side (but not exclusively so).

• Surprisingly effective: Python logic decides:
  - What text to assemble
  - How to assemble it
A small increase in sophistication: Using m4

Example: Assume we have a file, form.txt, with the following contents:

```
Hello, NAME, today is DATE.
```

Now consider the following command:

```
m4 -DNAME=Sally -DDATE='22-June-2004' form.txt
```

The command produces the following output:

```
Hello, Sally, today is 22-June-2004.
```
Now use Python to command m4 and TeX

Assemble TeX code from snippets of text.

Gather data for tag-replacement.

Build m4 command line with \-Dname=value arguments.

Execute the m4 command.

Present m4's output to TeX.
#!/usr/bin/env python
import sys
import os
import commands

# DATA TO BE 'MERGED' ONTO FORM.TXT:
data = {'NAME': 'Sally', 'DATE': '22-June-2040'}

# BUILD THE m4 COMMAND LINE, EXECUTE IT, AND GET RESULT:
cmd = 'm4 \
for name in data.keys():
    cmd += " -D%s='%s'" % (name, data[name])
cmd += ' form.txt'
snippet = commands.getoutput(cmd)

# ASSEMBLE TEX FILE, RUN TEX AND PRODUCE POSTSCRIPT FILE:
f = open('MyDocument.tex', 'w')
f.write('\nopagenumbers\n')
f.write(snippet)
f.write('\vfil\eject\bye\n')
f.close()
os.system('tex MyDocument.tex')
os.system('dvips MyDocument')
print 'Done.'
While m4 is an excellent macro preprocessor, there is another equally powerful tool that can do the job.

And we're already using it... TeX!

Consider the following file, form.tex:

```
\nopagenumbers
This is my first \TeX\ document produced from a script.
\par
Hello, \NAME, today is \DATE.
\vfil\eject
```

Alone, this file will result in undefined macro references because the macros \NAME and \DATE are not defined.
Imperative TeX code-writing script

```python
#!/usr/bin/env python
import sys
import os
f = open('temp.tex', 'w')
f.write('\\def\NAME{Sally}\n')
f.write('\\def\DATE{22-June-2004}\n')
f.write('\\input form.tex\n')
f.write('\\bye\n')
f.close()

os.system('tex temp.tex')
os.system('dvips temp')
print 'Done.'
```
Results of the script

Running the previous script creates \texttt{temp.tex}:

\begin{verbatim}
\def\NAME{Sally}
\def\DATE{22-June-2004}
\input form.tex
\bye
\end{verbatim}
Formalizing the process as an API: TeXmerge

- Need a concise formalism for interfacing with TeX in order to:
  - Escape tokens in merge data that would otherwise confuse TeX
  - Remove the tedium of running TeX and backend DVI programs
  - Help with error checking
- The API is called TeXmerge
Schematic overview of TeXmerge-based document production

1. **Template .tex files**
2. **Data Source**
3. **TeXmerge Application**
4. **Temporary .tex file**
5. **Run TeX and DVI backend processor**
6. **Finished Document**
A TeXmerge script:

```python
#!/usr/bin/env python
import sys
import os
import TeXmerge

f = TeXmerge.openOutput('temp.tex')
mergeVars = {'NAME': 'Sally',
             'DATE': '22-June-2004'}

TeXmerge.merge('form.tex', mergeVars)
TeXmerge.closeOutput(f)
TeXmerge.process('temp.tex', 'dvips')
print 'Done.'
```
Results of the script

Running the previous script creates `temp.tex`:

```
\batchmode
\nopagenumbers
\begingroup
\def\NAME{Sally}
\def\DATE{22-June-2004}
\input form.tex
\endgroup
\bye
```
Another Example
(this time object-oriented)

```python
#!/usr/bin/env python
import sys
import os
import TeXmerge

mergeObj = TeXmerge.TeXmerge('temp.tex')
mergeVars = {'NAME': 'Sally',
             'DATE': '22-June-2004'}

mergeObj.merge('form.tex', mergeVars)
mergeObj.process('dvips')
print 'Done.'
```
Work increasingly done by TeX

- TeXmerge-based Applications
- Sophisticated Script-TeX-Script Schemes
- Cooperative TeXmerge/Code-writing Applications
- TeX Code-writing Applications

Work increasingly done by script
Going Further with Macros

- Do-Nothing Macros – TeX sees them as `\relax`. Python scripts search for and act on them
  - Classic merge variable declarations
  - Extended merge variable declarations
  - Named text blocks

- Do-Something Macros and Hybrid schemes
Merge variable declarations

- Allow the author of template (form) .tex documents to explicitly state the names of all merge variables used in the document
- TeXmerge API has a method call to extract these declarations
- Two flavors:
  - Classic
  - Extended
Classic merge variable declarations

- **Usage:**
  - `\texmergevar varname`

- Simply states that *varname* will be referenced in the document as `\varname`

- Therefore, the data dictionary passed to the `merge()` method must have a tag *varname* or TeX will throw an error.
Extended merge variable declarations

• Usage:
  - \texttt{\text{\textbackslash texmergevardef}[\text{attrName}=\text{attrValue}...\text{]}}

• Attribute names:
  - Name= the name of the merge field
  - Type= the type of merge field. Intended to convey information to GUI applications
    • Entry: a simple text entry field
    • Text: a multi-line text entry field
    • Toggle: a toggle button field
    • Optionmenu: a drop-down option menu of choices
    • Radiobutton: a set of mutually exclusive toggle buttons
More on extended merge field attribute values:

- **Attribute names** (continued)
  - **Values**= a list of valid values for the variable, delimited by |'s
  - **Labels**= a list of alternate labels that should be associated with the values attribute for display purposes. Used with toggle, optionmenu, and radiobutton field types.
  - **Descr**= a textual description of the merge field's purpose
Retrieving information about extended merge fields

• Use the TeXmerge method `getExtendedNames()`

• Returns a dictionary, keyed by field name. The key's value is a dictionary of field attributes: name, type, value, etc.
getExtendedNames() example

Assume we have the file test.tex:

```
texmergevardef[name=ISTATE, type=optionmenu, values=TX|OK|AZ|CA|OR|WA, descr=Issuing state]
texmergevardef[name=ONAME, type=entry, descr=Owner name']
texmergevardef[name=APPTYPE, type=radibutton,values=1|2|3, labels=Employee|Spouse|Child,
           descr=Applicant type]
```

The call:

```
TeXmerge.getExtendedNames('test.tex')
```

will return the Python dictionary:

```
{'ISTATE': {'name': 'ISTATE', 'type': 'optionmenu',
             'values': ('TX', 'OK', 'AZ', 'CA', 'OR', 'WA'),
             'descr': 'Issuing state'},
 'APPTYPE': {'name': 'APPTYPE', 'type': 'radiobutton',
             'values': ('1', '2', '3'),
             'labels': ('Employee', 'Spouse', 'Child'),
             'descr': 'Applicant type'},
 'ONAME': {'name': 'ONAME',
             'type': 'entry', 'descr': 'Owner name'}}
```
Named Text Blocks

• Some applications have need to share identical text between two markup languages. i.e. TeX and HTML

• It is the language that needs to be shared (i.e. For legal purposes), not the structure of the text.

• The *named text blocks* technique makes a TeX document the owner of the text and the shared language is delimited by a set of special macros:
Named Text Blocks Example

This technique is best explained by example. Consider the file test.tex below:

\begin{verbatim}
This is a test document containing \textit{named text blocks.}
\StartNamedTextBlock[name=B1]
This is the first block.
\StopNamedTextBlock
Now for a second block:
\StartNamedTextBlock[name=B2]
Second block
\StopNamedTextBlock
Now for a series of sequenced blocks...
\line{\hbox{\StartNamedTextBlock[name=C1,seq=1]C1.Left\StopNamedTextBlock\hfil}
\hbox{\hfil\StartNamedTextBlock[name=C1,seq=2]C1.Right\StopNamedTextBlock}}
Finally, a named text block having a subkey:
\StartNamedTextBlock[name=D1,istate=TX]
This text is specific to the state of Texas.
\StopNamedTextBlock
\end{verbatim}

\StartNamedTextBlock[...] marks the beginning of a block.
The name=... argument assigns a name to the block.

\StopNamedTextBlock marks the end of a block.

Partial blocks can be declared with the seq=... parameter
Working with named text blocks
an interactive Python interpreter session...

```python
>>> import TeXmerge
>>> o = TeXmerge.TextBlockManager('test.tex')
>>> o
<TeXmerge.TextBlockManager instance at 0x750648>
>>> o.getBlockNames()
['C1', 'B1', 'B2', 'D1']
>>> b1 = o.getBlock('B1')
>>> b1
<TeXmerge.TextBlock instance at 0x72b5d0>
>>> b1.getText()
'This is the first block.'
>>> c1 = o['C1']
>>> c1.getTextSegments()
{1: 'C1.Left', 2: 'C1.Right'}
>>> c1.getText()
'C1.Left C1.Right'
>>> d1 = o['D1']
>>> d1.getSubkeys()
['istate']
>>> d1.getSubkeyValues('istate')
['TX']
>>> d1.getText('istate','TX')
'This text is specific to the state of Texas.'
```
Work increasingly done by TeX

- TeXmerge-based Applications
- Sophisticated Script-TeX-Script Schemes
- TeX Code-writing Applications
- Cooperative TeXmerge/Code-writing Applications

Work increasingly done by script
DoSomething Macros
Document Templates

- Document template macros serve to produce documents where a certain structure needs to be imposed
- Follows a “plug-and-socket” model
- Three types of macro arguments:
  - Simple parameters (mp....)
  - Data sockets (sd...)
  - Slots (sl...)
CONFIRMATION OF CHANGE OF ADDRESS

June 22, 2004
Jimmy Studebaker
814 Allota St.
Millburg, MI 4513

RE: Insured: Gordon Arnett
Policy No: 0001131

FAX TO
INSURABLE SPOUSE
214-745-6391

779 Sunset Hills Dr.
Millburg, MI 45582-3366

This letter is to confirm that the address for the above referenced policy has been updated recently. The address for this policy and any other policies you may have with Texas Life has been changed to:

If the information shown above is not correct, please contact our office at 1-800-283-5515 and ask for extension 8813, or write to the address shown below.

Kathy D. Jackson 00002120042
Hybrid Script-TeX-Script Schemes

If we have a complex application where a substantial amount of the document's content may vary, the merge paradigm of TeXmerge begins to break down under the complexity of so many variables. This is especially true of variable tabular data.
Premium “Mode” varies: i.e. Monthly, Semi-Monthly, Annual, etc.

“Underwriting class” varies: ‘Express’ or ‘Simplified’

Each cell value and footnote label must be calculated.

Form number is variable.

<table>
<thead>
<tr>
<th>Age (Years)</th>
<th>Premium Column</th>
<th>Footnote Label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Premium waiver, ADB and group-size optional / variable.

Premium column headers are variable and changes affect calculated cell values.

<table>
<thead>
<tr>
<th>Life Insurance Face Amount Premiums for Semi-Monthly Premiums Shown</th>
<th>Period To Age to Which Coverage Continues</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Each cell value and footnote label must be calculated.

Form number is variable.
Rate Sheet Example

• The preceding page is complex.
  - Only one page of a larger document
  - Remainder of document has merge variables and works well in the TeXmerge framework
  - This sheet needs to be embeddable into many documents

• We desire a macro to make implementation simple and painless...
The MakeRateSheet macro

\MakeRateSheet[uwclass=express,
    mode=semi-monthly,
    groupsize=150,
    formno=test,
    waiver=yes,
    adb=yes
]

- Gathers macro arguments and marshal them into a Python script command-line. The python script's function is to generate TeX code that will format the rate sheet based on passed arguments.
- Executes the python script.
- \input's the file of code produced by the python script.
Script writes TeX code

TeX interprets code, encounters script-enabled macros

Secondary script executed via \write18 with possible arguments.

Temporary TeX code

Run DVI backend (dvips)

Post-process PostScript

Printer-ready file

Schematic Overview of the Hybrid Script-TeX-Script Method
GUI Applications
# TeXmerge – the application

![TeXmerge Application Screenshot](image.png)

### Multi-Document View

| POLNUM | 0001111 |
| AGTNAME | Myrt D. Jackson |
| AGTNUMB | 0001292042 |
| FRANCHG | |
| INAME | Gordon Arnett |
| INAMES | |
| LADDR | 614 Ailes St. Millersburg, MI 85598-4513 |
| OADDR | 179 Sunset Hills Dr. Millersburg, MI 85593-3366 |
| ONAME | Jimmy Studebaker |
| SDATE | June 22, 2004 |
TeXtool
Document selection panel

The screenshot shows a TeXtool interface with various options and fields. The left panel displays a list of source files, including `example1`, `example2`, and others. The right panel contains fields for `Main`, `Merge Fields`, and `Option Menu test`, with values such as `Edit /usr1/hcswmr/sftug/t1.tex` and `v1` selected. The `Source Files` section lists various LaTeX files, and the `Issuing State` field shows `AZ`. The `Preview` pane on the right displays a document, and the `Log` pane shows some typeset text. The bottom of the interface includes buttons for `Open`, `New File`, and `Refresh`.
TeXtool
Document editor panel

There are three ways to integrate \TeX with scripting languages:

1. **The simplest way** is to write a script which produces \TeX code in a file, and then automatically run that file.

2. **Second way** is to write the \TeX code manually, placing "placeholders" in the file at points where merge variables are to be inserted. The process of replacing these placeholders with actual data is carried out via the program \texttt{bfm4}.

3. **Finally, the third way** is to use \TeX's own macro facilities for replacing placeholders with merge data.

Now it's time to think about the Preferences class. The following items are needed:

- Rebuild (timeout) \texttt{FORMAT} \texttt{show log on error or not}
- Directory expansion in file tree
- (lefttop offsets for dvipng)
- (Render DPI)
- (Number of backup copies to make)
- (Reopen last opened file)
- Bigskip

%\texttt{Wfileject}

now is the time. --- now is not the time.
TeXtool
Document preferences panel

TeX
TeX Format: Determine automatically
Rebuild timeout (in milliseconds): 700

Misc
Number of backup versions: 1
Preferred printer queue: 1

Make PDF snapshot  Make PostScript snapshot
The big picture at Texas Life
Conclusion
Why we like hacking Python and TeX

- TeX is a powerful page description language
- Conditional typesetting / intelligent documents
- Python is a malleable, high level language
- Documents are simple ASCII text files
- Independent of proprietary technology
- Bounded only by Imagination
- It's fun!