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...
- Yields: Automatic document preparation.

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

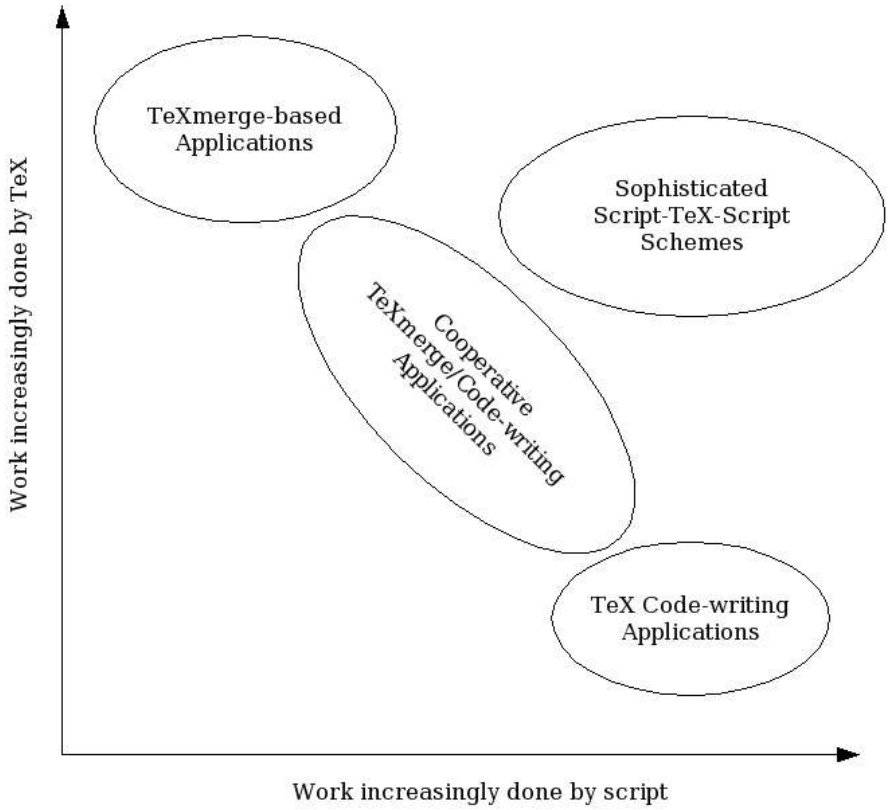
Popular Scripting Languages

- 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.

```
!/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 \
    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.
```

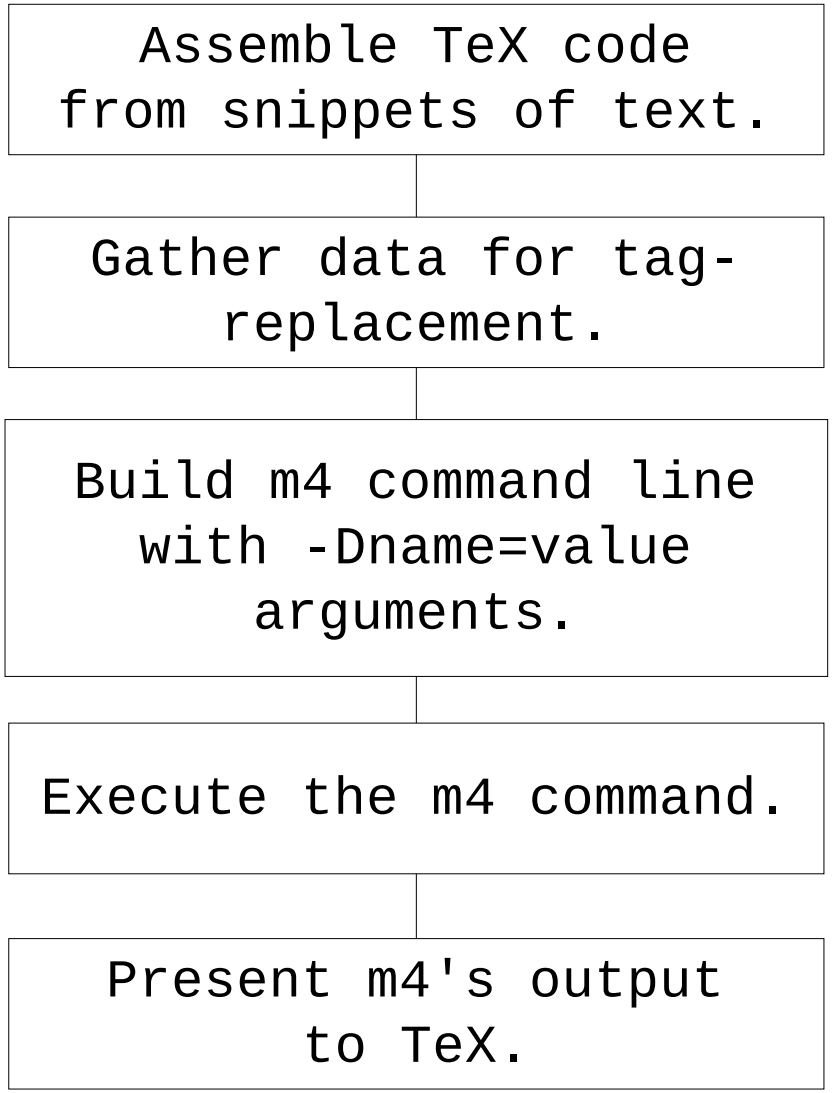
Now consider the following command:

```
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



M4 Example:

```
#!/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

```
#!/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

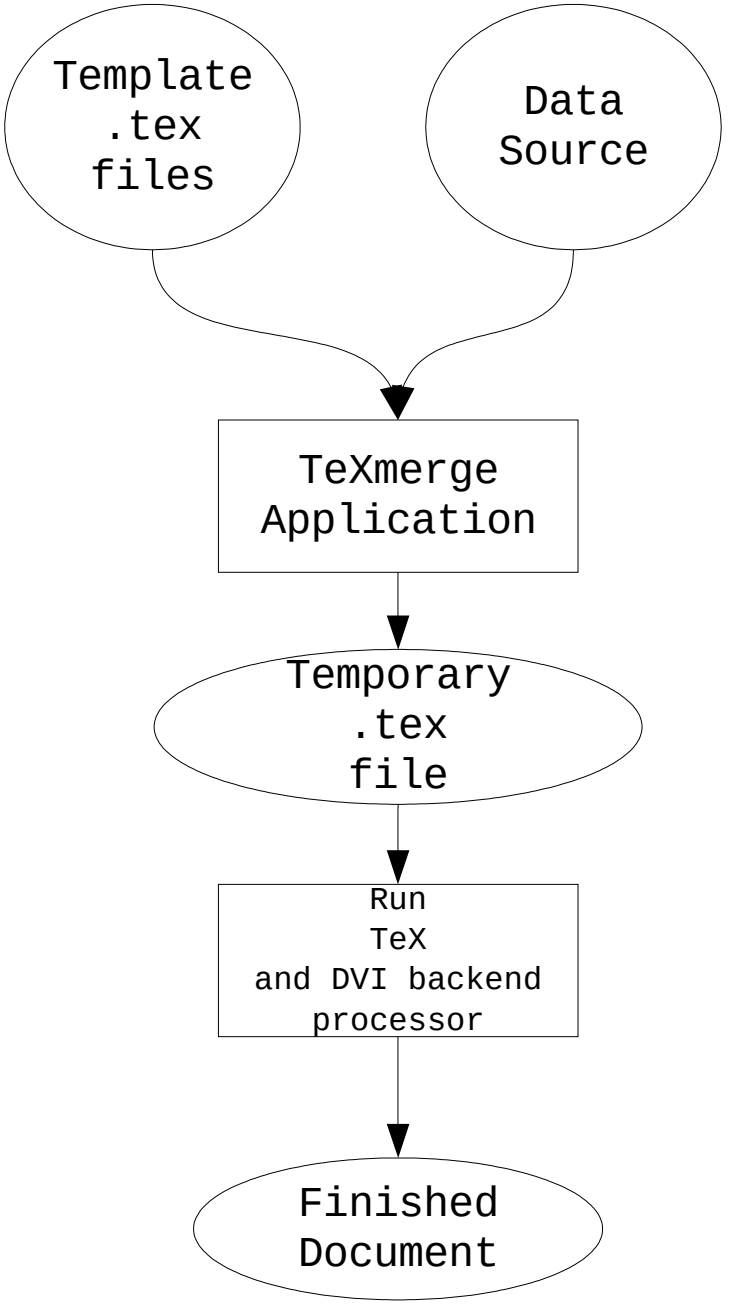
Running the previous script creates `temp.tex`:

```
\def\NAME{Sally}
\def\DATE{22-June-2004}
\input form.tex
\bye
```

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



A TeXmerge script:

```
#!/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`:

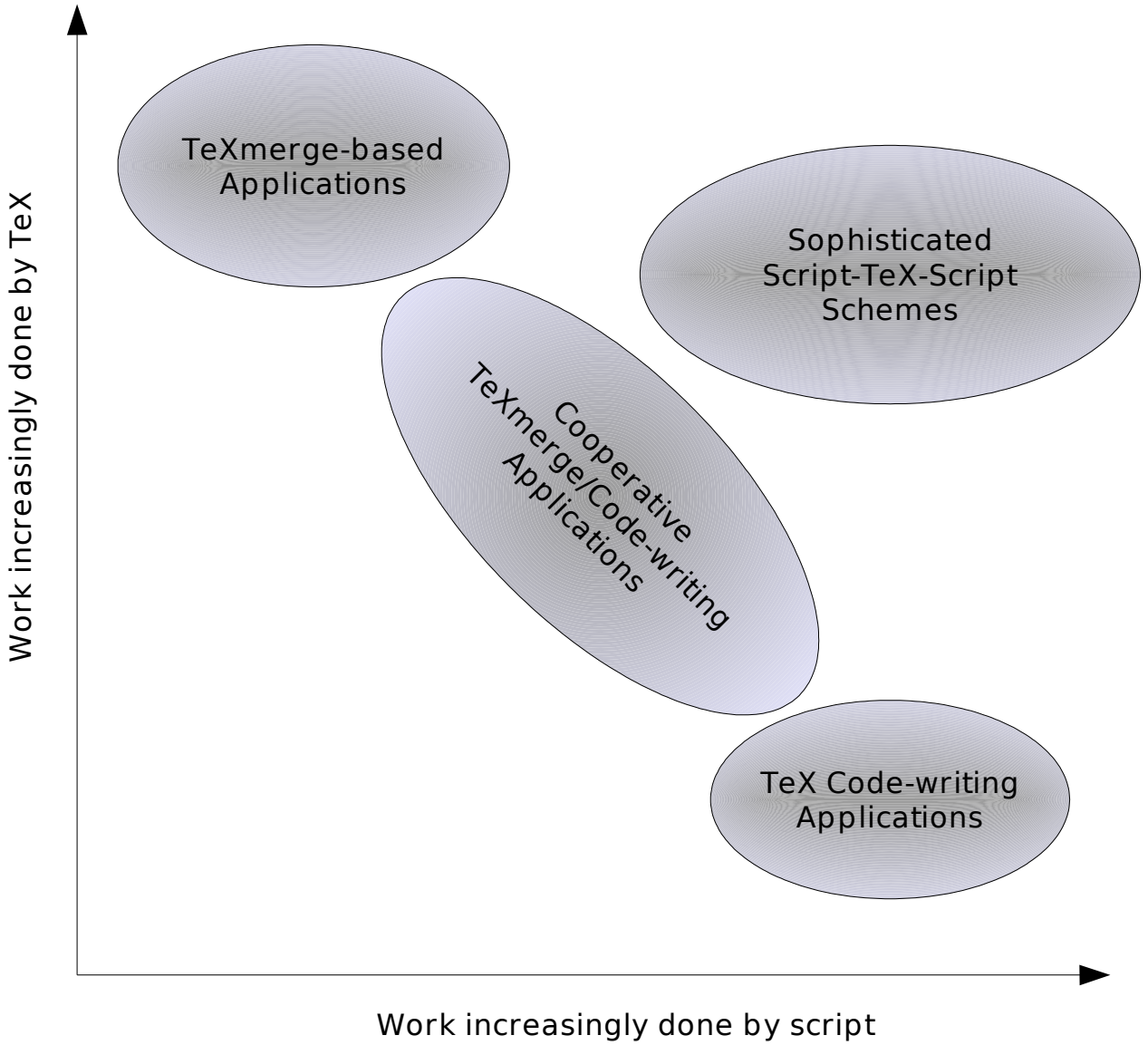
```
\batchmode
\nopagenumbers
\begingroup
\def\NAME{Sally}
\def\DATE{22-June-2004}
\input form.tex
\endgroup
\bye
```

Another Example (this time object-oriented)

```
#!/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.'
```


TeXmerge-based Applications
Sophisticated Script-TeX-Script Schemes
Cooperative TeXmerge/Code-writing Applications
TeX Code-writing Applications
Work increasingly done by TeX
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:
  - `\texmergevardef[`*`attrName=attrValue...`*`]`
- 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=radiobutton,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 the file `test.tex` below:

```
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
```

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

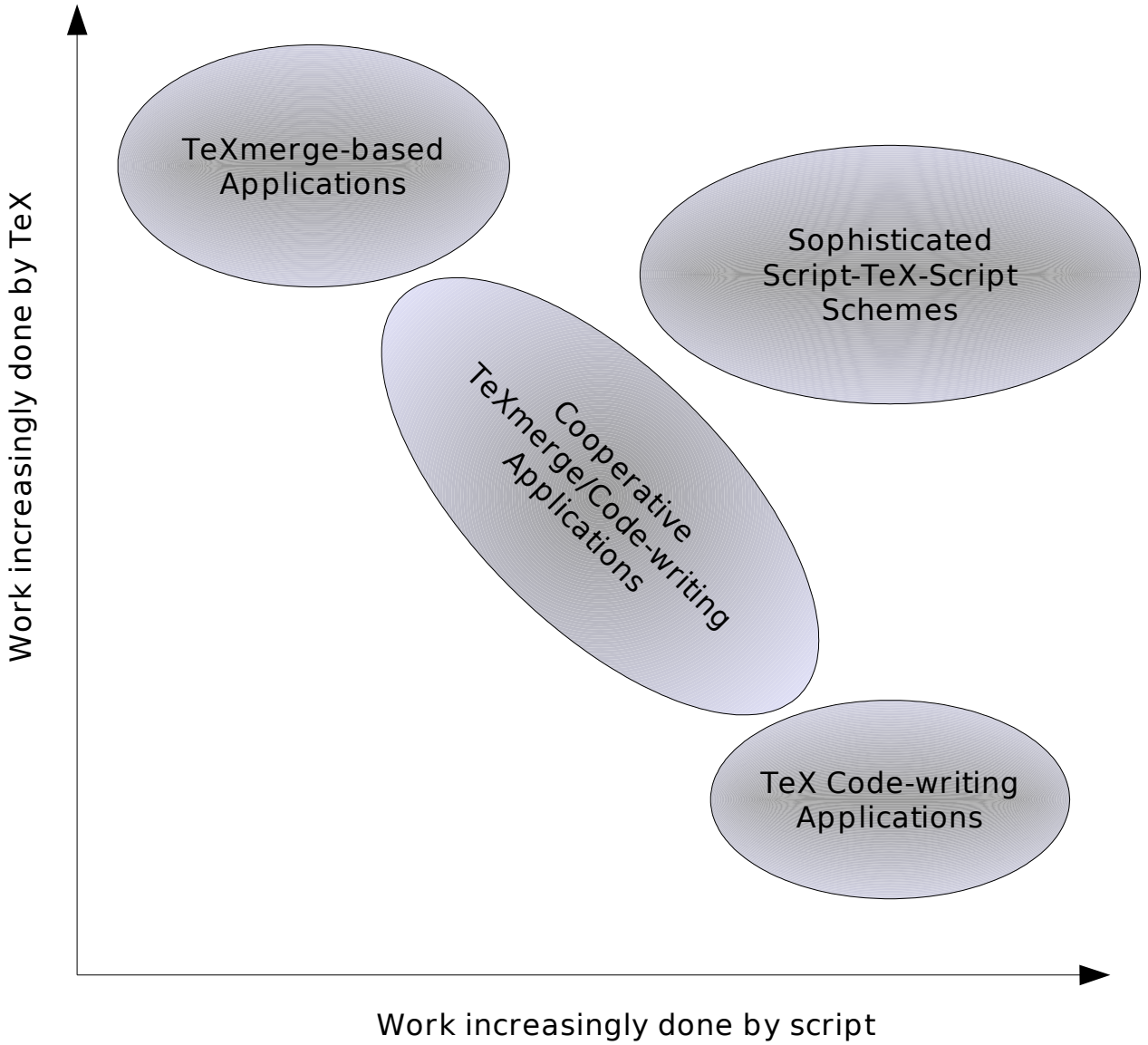
`\StopNamedTextBlock` marks the end of a block.

Partial blocks can be declared with the `seq=...` parameter

Working with named text blocks

an interactive Python interpreter session…

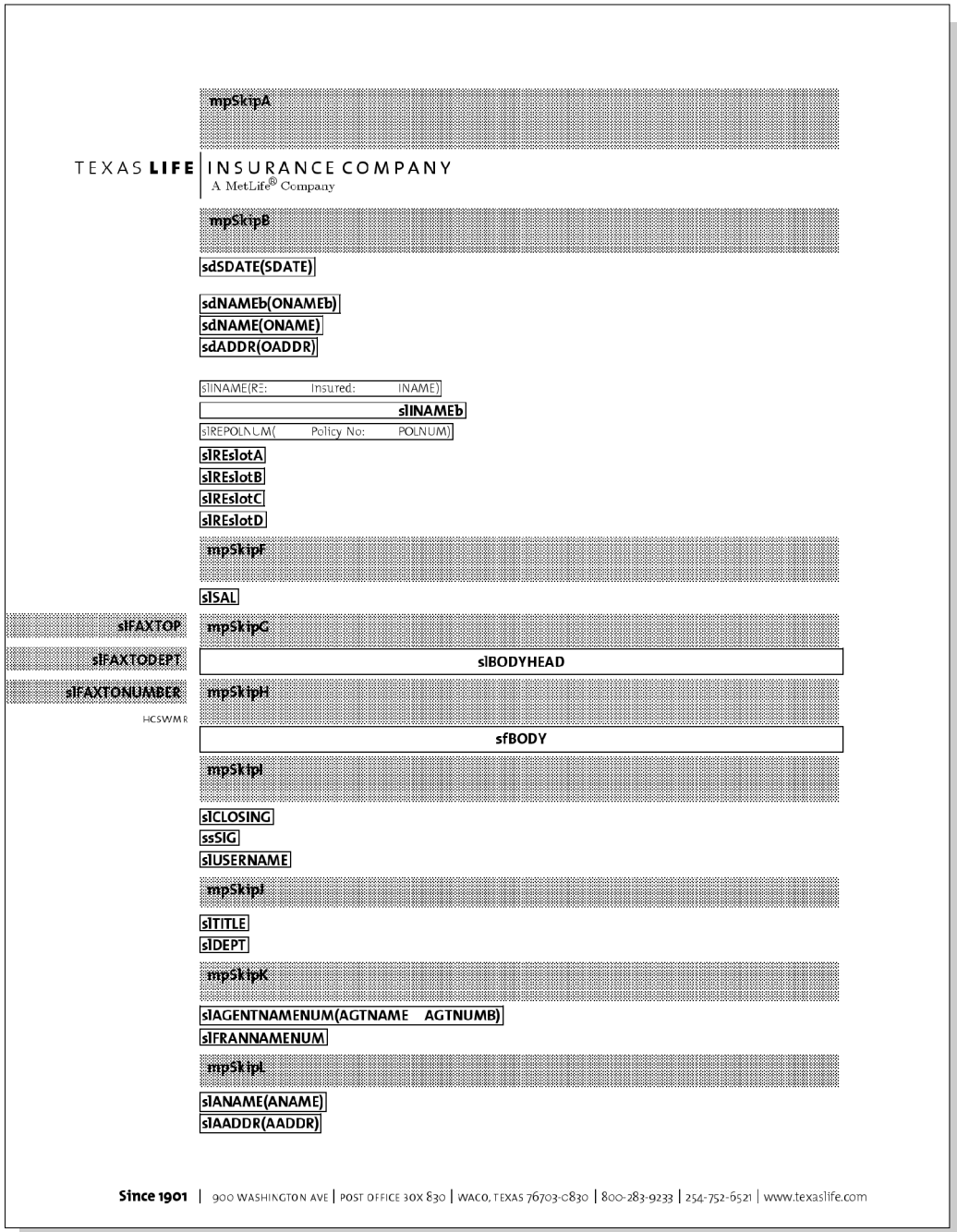
```
[hawkeye2:~/sftug] williamr% python
Python 2.3.2 (#1, Nov  6 2003, 13:18:07)
[GCC 2.95.2 19991024 (release)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import TeXmerge
>>> o = TeXmerge.TextBlockManager('test.tex')
>>> o
<TeXmerge.TextBlockManager instance 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.'
```

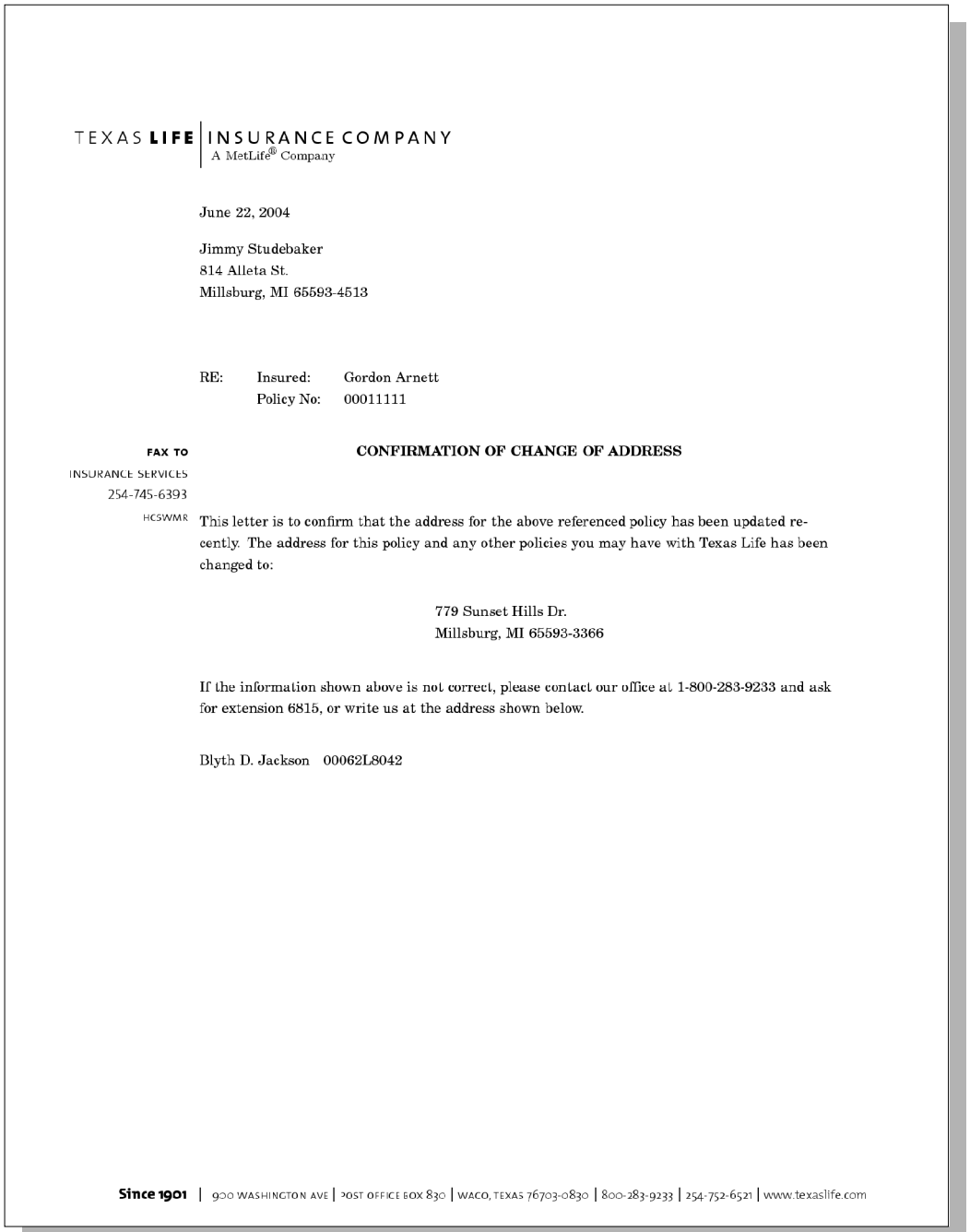

TeXmerge-based Applications
Sophisticated Script-TeX-Script Schemes
Cooperative TeXmerge/Code-writing Applications
TeX Code-writing Applications
Work increasingly done by TeX
Work increasingly done by script

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…)

Client Letter document template, left, showing sockets and slots. Actual document produced by `\StartClientLetter[]`, right.





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.

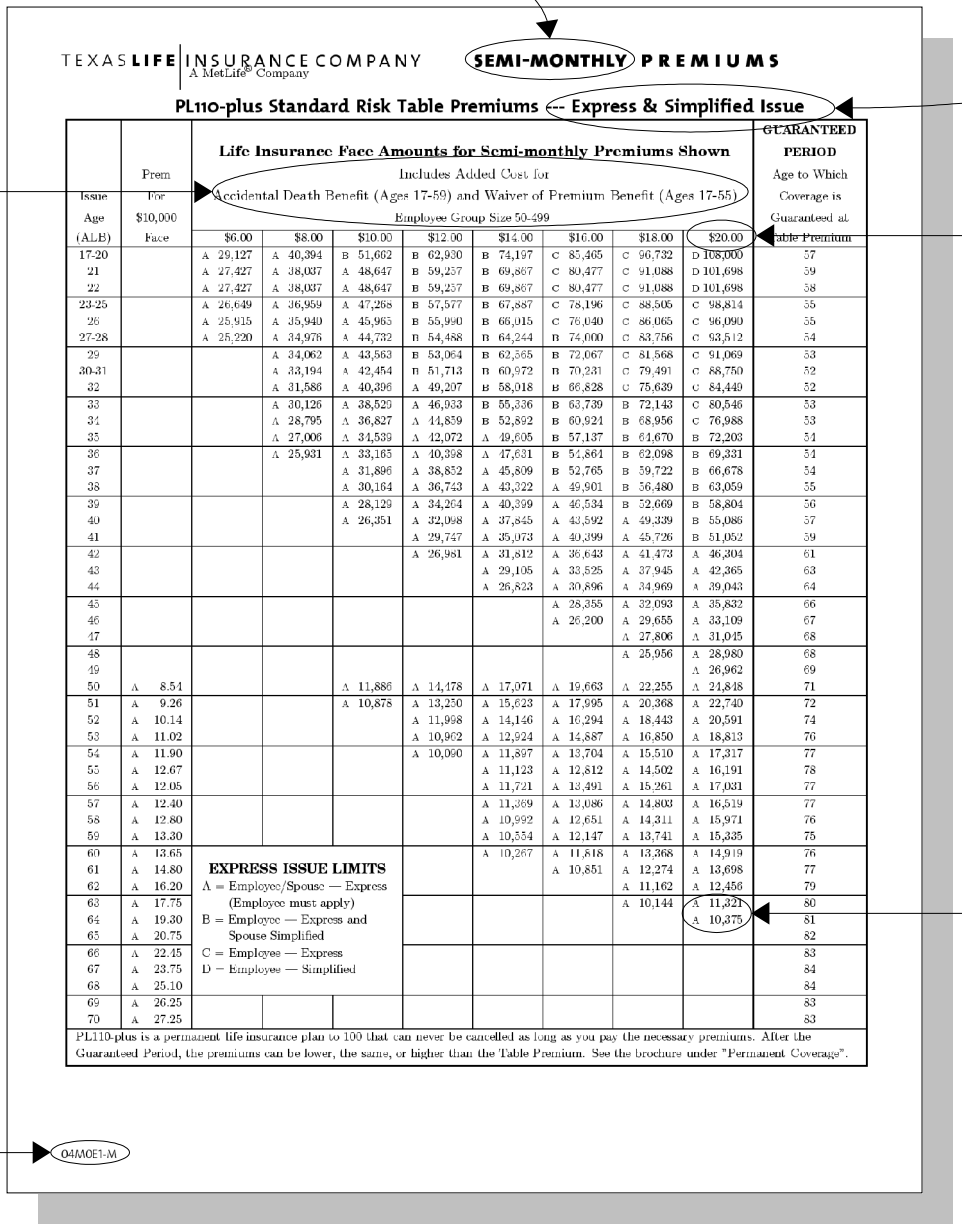
“Underwriting class” varies: 'Express' or 'Simplified'

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

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

Each cell value and footnote label must be calculated.

Form number is variable.



TEXAS LIFE INSURANCE COMPANY
A MetLife® Company

SEMI-MONTHLY PREMIUMS

PL110-plus Standard Risk Table Premiums --- Express & Simplified Issue

Issue Age (ALB)	Prem For $10,000 Face	Life Insurance Face Amounts for Semi-monthly Premiums Shown — Includes Added Cost for Accidental Death Benefit (Ages 17-59) and Waiver of Premium Benefit (Ages 17-55) — Employee Group Size 50-499								GUARANTEED PERIOD Age to Which Coverage is Guaranteed at Table Premium
		$6.00	$8.00	$10.00	$12.00	$14.00	$16.00	$18.00	$20.00	
17-20		A 29,127	A 40,394	B 51,662	B 62,930	B 74,197	C 85,465	C 96,732	D 108,000	57
21		A 27,427	A 38,037	A 48,647	B 59,257	B 69,867	C 80,477	C 91,088	D 101,698	59
22		A 27,427	A 38,037	A 48,647	B 59,257	B 69,867	C 80,477	C 91,088	D 101,698	58
23-25		A 26,649	A 36,959	A 47,268	B 57,577	B 67,887	C 78,196	C 88,505	C 98,814	55
26		A 25,915	A 35,940	A 45,965	B 55,990	B 66,015	C 76,040	C 86,065	C 96,090	55
27-28		A 25,220	A 34,976	A 44,732	B 54,488	B 64,244	B 74,000	C 83,756	C 93,512	54
29			A 34,062	A 43,563	B 53,064	B 62,565	B 72,067	C 81,568	C 91,069	53
30-31			A 33,194	A 42,454	B 51,713	B 60,972	B 70,231	C 79,491	C 88,750	52
32			A 31,586	A 40,396	A 49,207	B 58,018	B 66,828	C 75,639	C 84,449	52
33			A 30,126	A 38,520	A 46,933	B 55,336	B 63,739	B 72,143	C 80,546	53
34			A 28,795	A 36,827	A 44,859	B 52,892	B 60,921	B 68,956	C 76,988	53
35			A 27,006	A 34,539	A 42,072	A 49,605	B 57,137	B 64,670	B 72,203	54
36			A 25,931	A 33,165	A 40,398	A 47,631	B 54,864	B 62,098	B 69,331	54
37				A 31,896	A 38,852	A 45,809	B 52,765	B 59,722	B 66,678	54
38				A 30,164	A 36,743	A 43,322	A 49,901	B 56,480	B 63,059	55
39				A 28,129	A 34,264	A 40,399	A 46,534	B 52,669	B 58,804	56
40				A 26,351	A 32,098	A 37,845	A 43,592	A 49,339	B 55,086	57
41					A 29,747	A 35,073	A 40,399	A 45,726	B 51,052	59
42					A 26,981	A 31,812	A 36,643	A 41,473	A 46,304	61
43						A 29,105	A 33,525	A 37,945	A 42,365	63
44						A 26,823	A 30,896	A 34,969	A 39,043	64
45							A 28,355	A 32,093	A 35,832	66
46							A 26,200	A 29,655	A 33,109	67
47								A 27,806	A 31,045	68
48								A 25,956	A 28,980	68
49									A 26,962	69
50	A 8.54			A 11,886	A 14,178	A 17,071	A 19,663	A 22,255	A 24,848	71
51	A 9.26			A 10,878	A 13,250	A 15,623	A 17,995	A 20,368	A 22,740	72
52	A 10.14				A 11,998	A 14,146	A 16,294	A 18,443	A 20,591	74
53	A 11.02				A 10,962	A 12,924	A 14,887	A 16,850	A 18,813	76
54	A 11.90				A 10,090	A 11,897	A 13,704	A 15,510	A 17,317	77
55	A 12.67					A 11,123	A 12,812	A 14,502	A 16,191	78
56	A 12.05					A 11,721	A 13,491	A 15,261	A 17,031	77
57	A 12.40					A 11,369	A 13,086	A 14,803	A 16,519	77
58	A 12.80					A 10,992	A 12,651	A 14,311	A 15,971	76
59	A 13.30					A 10,554	A 12,147	A 13,741	A 15,335	75
60	A 13.65					A 10,267	A 11,818	A 13,368	A 14,919	76
61	A 14.80	**EXPRESS ISSUE LIMITS**					A 10,851	A 12,274	A 13,698	77
62	A 16.20	A = Employee/Spouse — Express (Employee must apply)						A 11,162	A 12,456	79
63	A 17.75							A 10,144	A 11,321	80
64	A 19.30	B = Employee — Express and Spouse Simplified							A 10,375	81
65	A 20.75									82
66	A 22.45	C = Employee — Express								83
67	A 23.75	D = Employee — Simplified								84
68	A 25.10									84
69	A 26.25									83
70	A 27.25									83

PL110-plus is a permanent life insurance plan to 100 that can never be cancelled as long as you pay the necessary premiums. After the Guaranteed Period, the premiums can be lower, the same, or higher than the Table Premium. See the brochure under "Permanent Coverage".

04M0E1-M

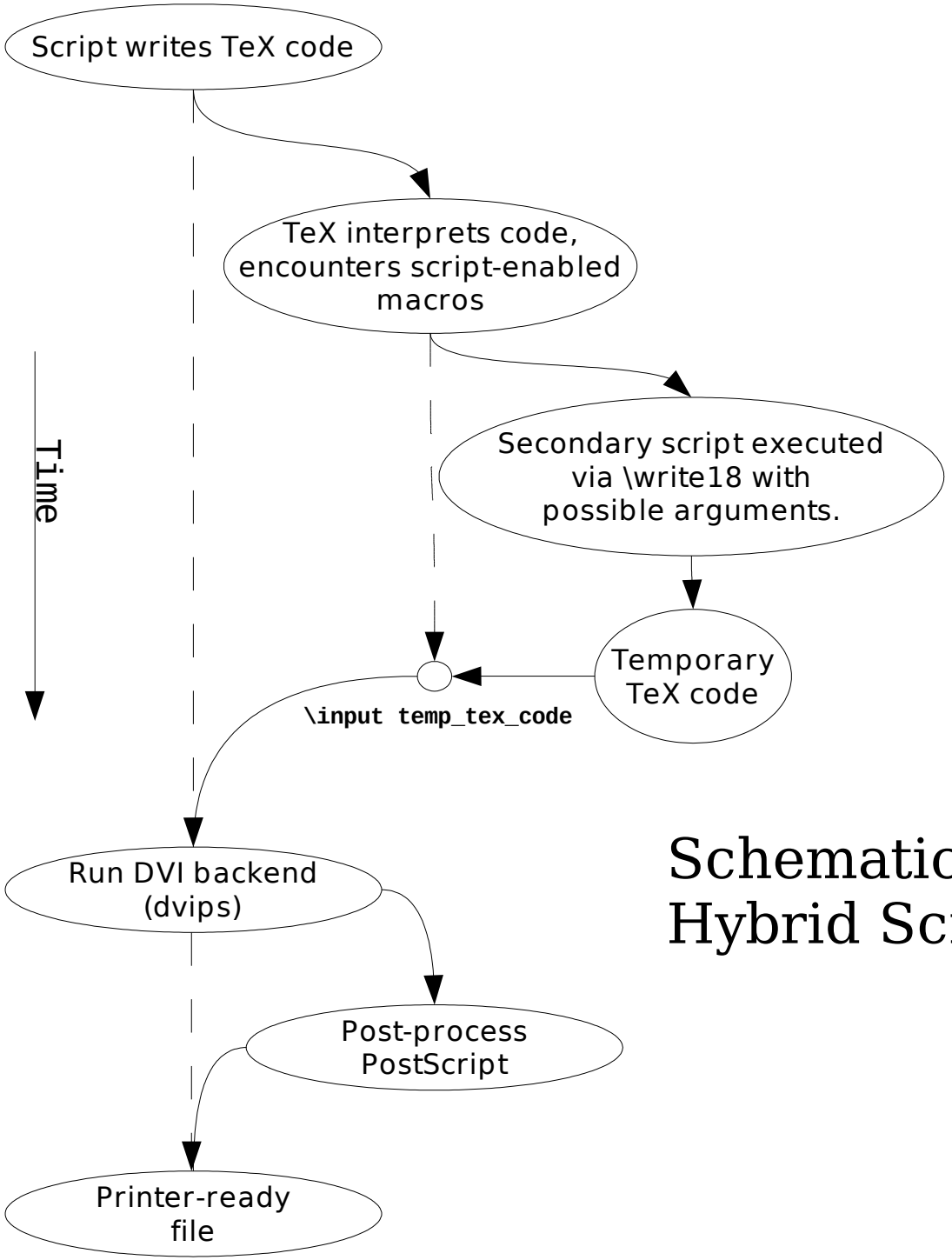
Rate Sheet Example

- 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
]
```

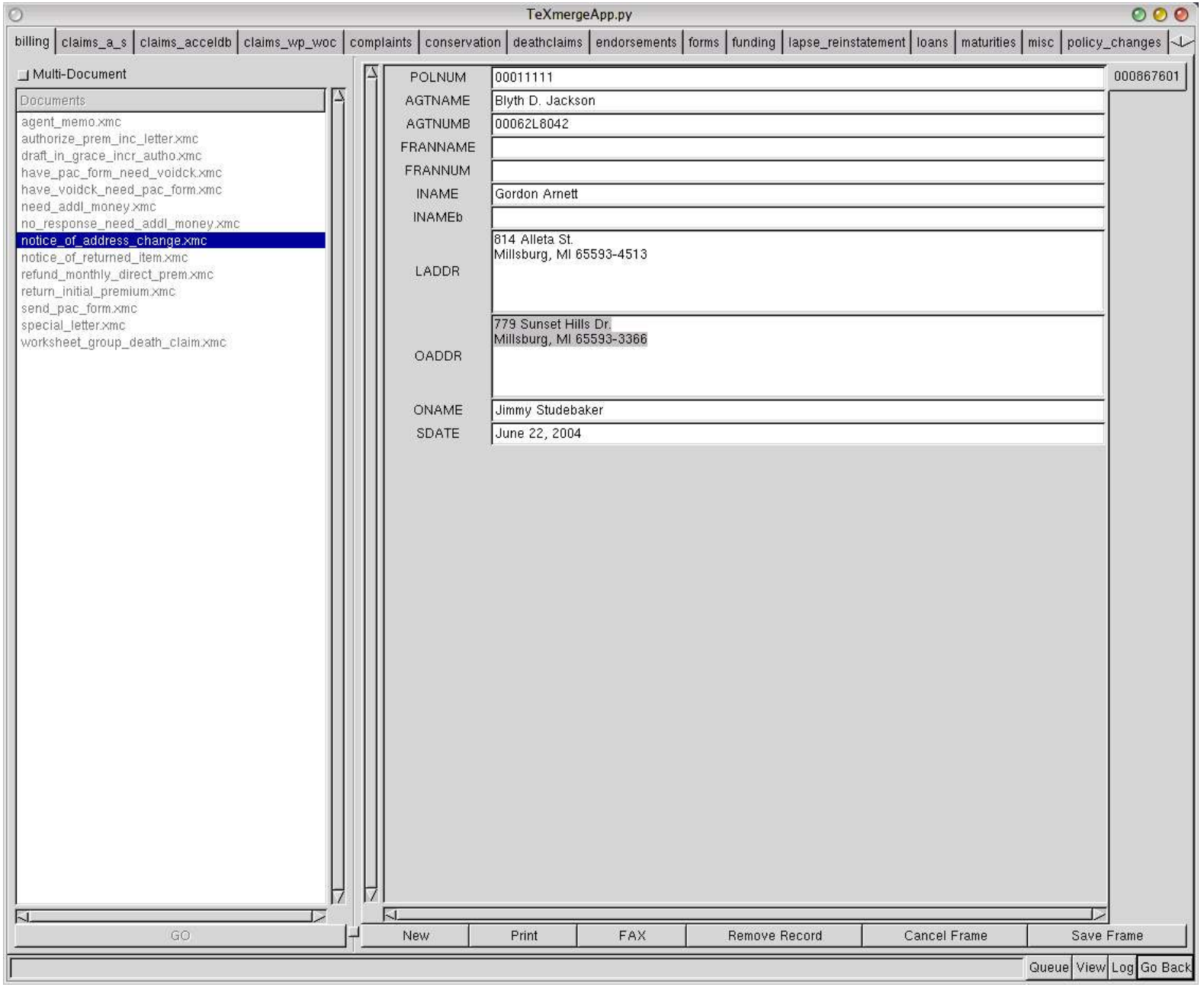
- Gathers macro arguments and marshal them into a Python script command-line. The python script's function is to generate TeX code that will format the rate sheet based on passed arguments
- Executes the python script
- `\input`'s the file of code produced by the python script



Schematic Overview of the Hybrid Script-TeX-Script Method

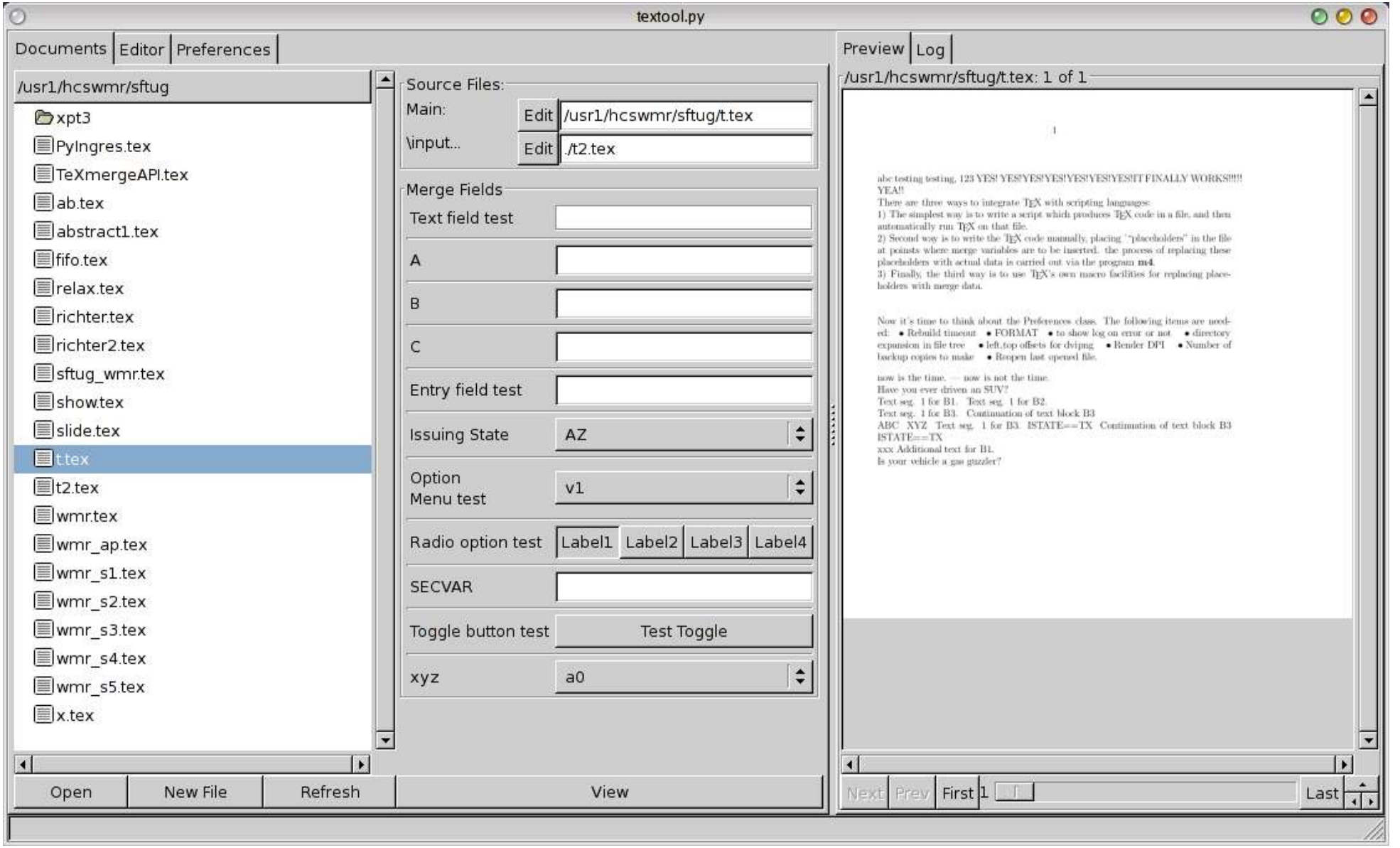
GUI Applications

TeXmerge – the application



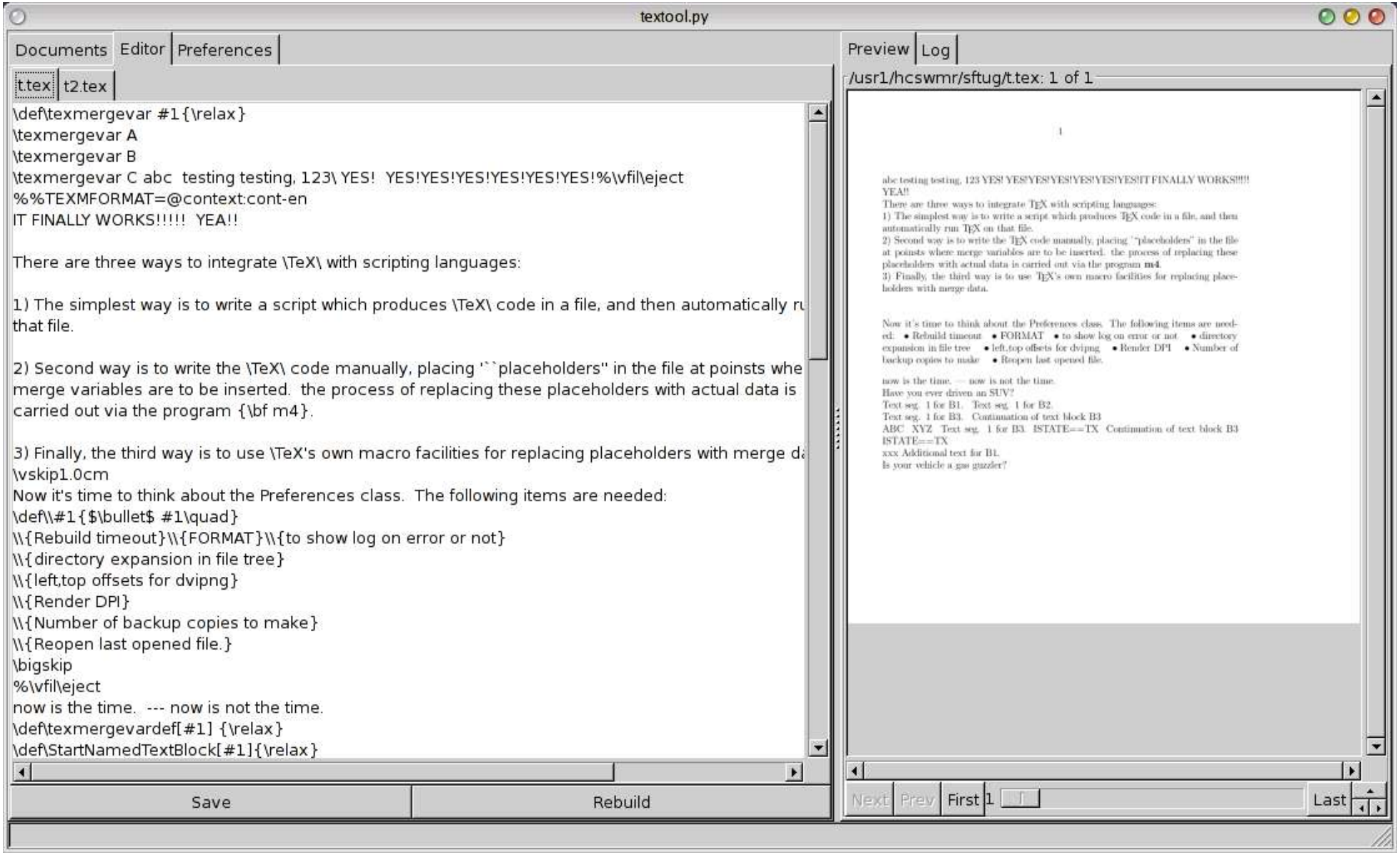
TeXtool

Document selection panel



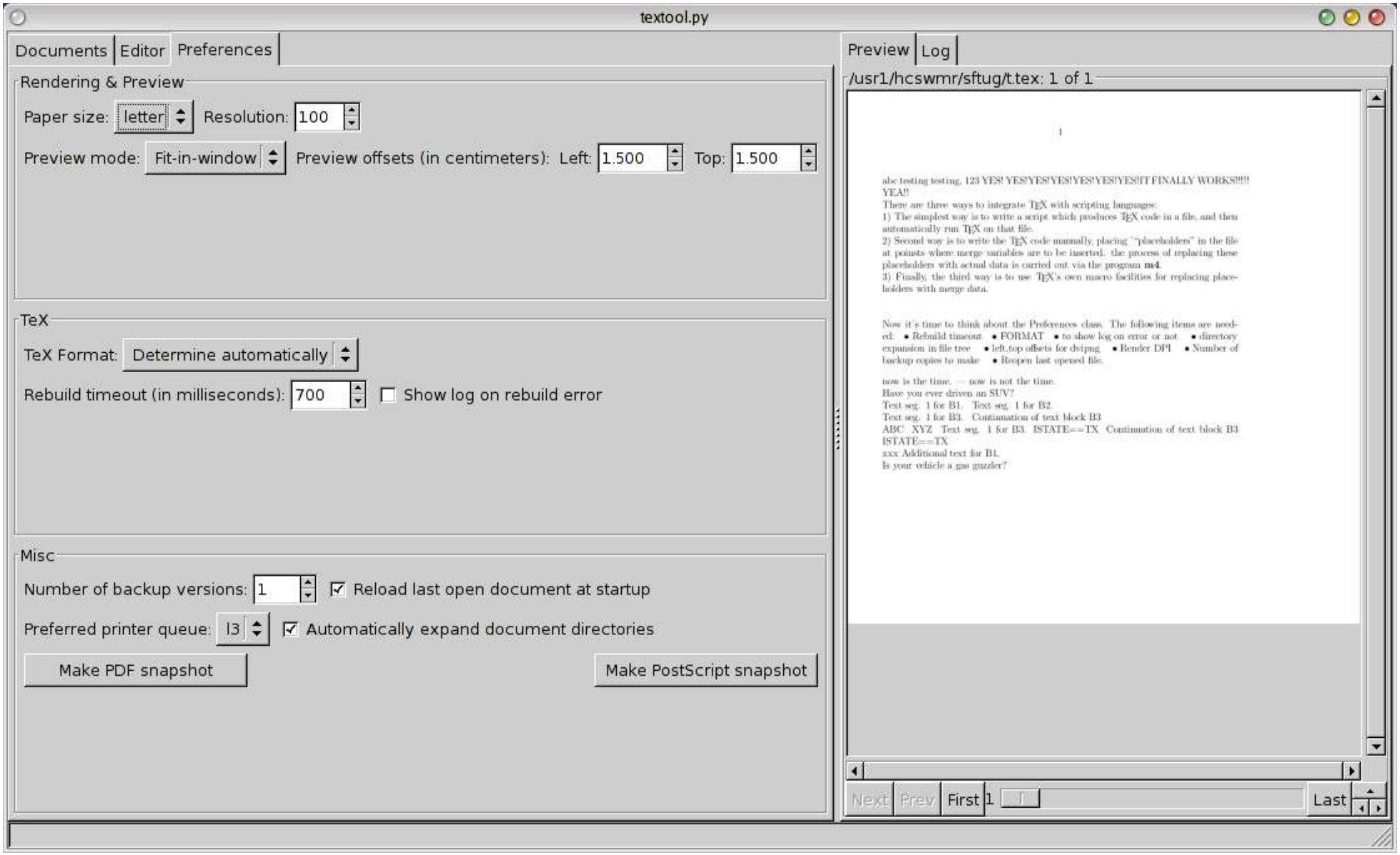
TeXtool

Document editor panel

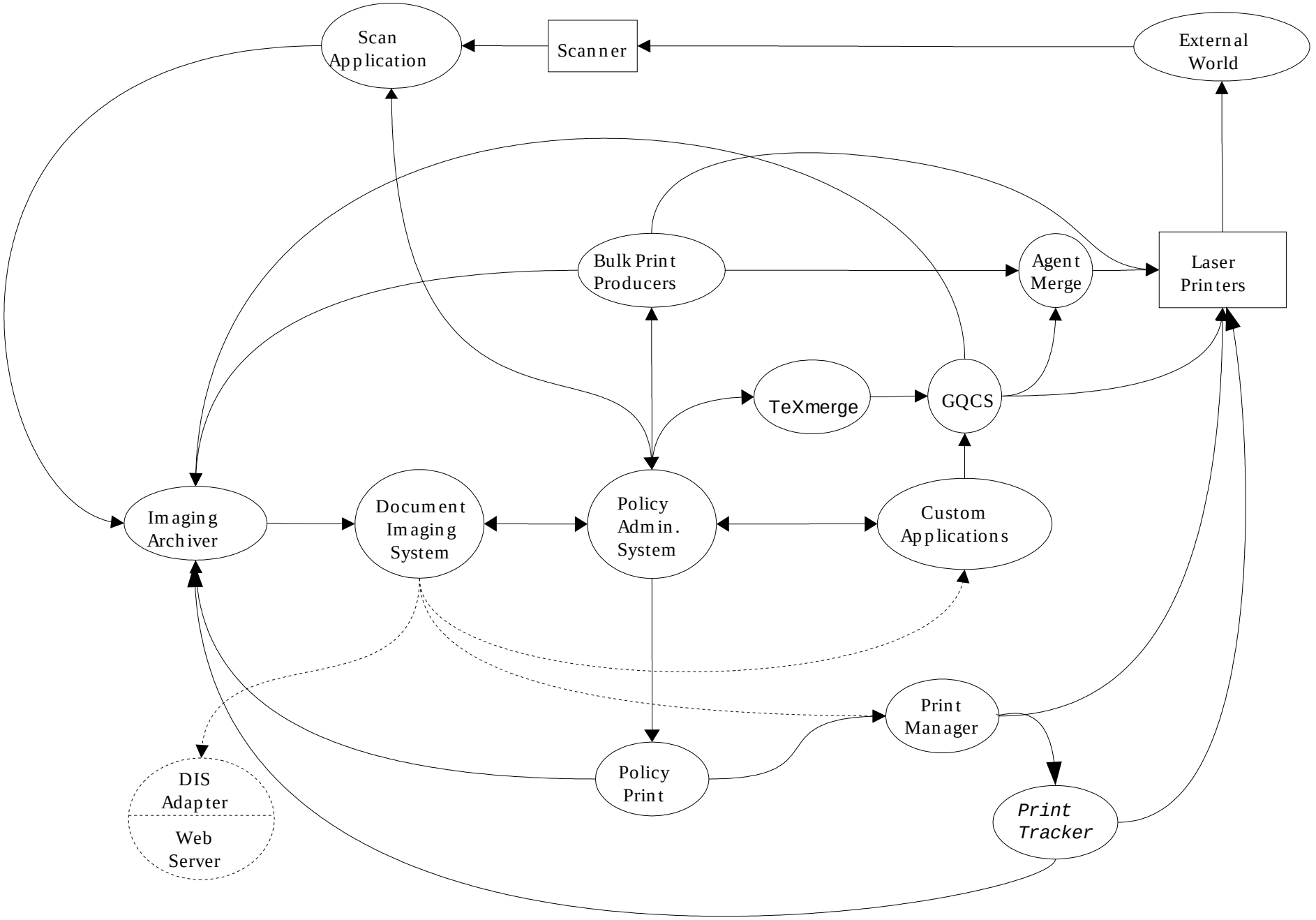


TeXtool

Document preferences panel



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!