

TeXmerge

A System For Generating Form-based Documents

William M. Richter

Texas Life Insurance
Waco, Texas, USA
hcsumr@texlife.com

1. Introduction

TeXmerge is a C-language API for merging variable data into a pre-existing TeX document. The API is simple, light-weight, and easy to integrate into applications. The power and flexibility of TeX make the API useful in high-capacity production-oriented systems.

2. The TeXmerge API

The API consists of a small number of functions, here listed in the normal order of use. Most functions return an integer result code. Zero implies successful return. The integer result may be passed to the function `TeXmerge_GetErrorString()` to retrieve the corresponding error text string.

2.1. `TeXmergeName_t *TeXmerge_AllocNames(int count)`

Variable data is passed to TeXmerge as an associative array. Each entry in the array is a name/value pair. This function allocates an array of such name/value pairs containing `count` entries.

2.2. `TeXmergeName_t *TeXmerge_FreeNames(TeXmergeName_t *array, int count)`

This functions frees a name/value array. Obviously this one is not listed in order of use!

2.3. `int TeXmerge_SetArrayEntry(char *name, char *value, TeXmergeName_t *entry)`

This function sets `entry` to the passed `name` and `value` strings. The next two functions are for convenience and their use is not mandatory. In this and the next function, `value` may be passed as `NULL` to indicate 'no value'.

2.4. `int TeXmerge_SetName(char *name, char *value, TeXmergeName_t *array, int count)`

This function searches `array` for an entry whose name value matches `name`, and then sets the entry's value to `value`.

2.5. `char *TeXmerge_GetName(char *name, TeXmergeName_t *array, int count)`

This function searches `array` for an entry whose name value matches `name`, and then returns the entry's value string. The function returns `NULL` if `name` is not found in `array`.

2.6. `char *TeXmerge_GetNames(char *pathname, TeXmergeName_t **array, int *count)`

This function searches the `.tex` file specified in `pathname` for occurrences of lines having the format:

```
\texmergevar NAME
```

Lines of this form enumerate the merge variables that the document will use. The function returns a list of the names in the pointer variable pointed to by `array`. The number of elements in the array is returned in the integer pointed to by `count`. `array` should be freed when no longer needed with a call to `TeXmerge_FreeNames()`.

2.7. `int TeXmerge_OpenOutput(char *pathname, FILE **outFP, char *preamble)`

This function creates an output file named `pathname`. An opened `FILE` pointer is returned in `outFP`. `preamble` is an optional "snippet" of `TEX` code that should be written at the beginning of the file. If `preamble` is passed as null, then no preamble code is generated.

2.8. `int TeXmerge(char *pathname, TeXmergeName_t *array, int count, FILE *fp, int options)`

This function is the heart of the API. `pathname` is the name of a TeX form file containing invocations of macros whose names are the name values set in the passed `array`. `fp` is the FILE pointer returned by `TeXmerge_OpenOutput()`. `options` controls the merge operation. Currently the only option is whether or not to draw a frame around the merged variables (`TXM_FRAMEVARS`).

2.9. `int TeXmerge_CloseOutput(FILE *fp)`

After all invocations of the above functions are complete, this function should be called to close the output file. `fp` is the FILE pointer returned from `TeXmerge_OpenOutput()`.

2.10. `int TeXmerge_Process(char *pathname, char *dvidrv_name)`

Once the output file has been closed, it is ready for backend processing by TeX. This function invokes TeX and then the dvi driver named in `dvidrv_name`. All temporary `.log` and `.dvi` files are removed after use.

2.11. `int TeXmerge_View(char *pathname, int waitOption)`

A convenience function to run TeX and then run `xdvi`. `waitOption` is one of `TXM_WAIT` or `TXM_NOWAIT`. If `TXM_NOWAIT` is passed, then the current process is forked and then `xdvi` is run in a child process.

3. `int TeXmerge_Print(char *pathname, char *lpargs, char *output_pathname)`

A convenience function to run TeX and then run `dvilj`. If `lpargs` is non-NULL then it is used as switches for the `lp` command and the resulting `.lj` file will be queued for printing via the `lp` system. The pathname of the resulting `.lj` is returned in the character array pointed to by `output_pathname`.

3.1. `char *TeXmerge_GetErrorString(int)`

This function returns a character string description corresponding to the passed integer value.

4. Example Program

A straightforward application of the TeXmerge API is illustrated in the following simple C code. This program,

- Creates an associative array with three elements.
- Sets the elements to have names THISVAR, THATVAR, and ANOTHERVAR, respectively.
- Opens an output file.
- Merges the associative array into an existing TeXfile called `test_form.tex` to create a temporary file called `temp.tex`.
- Closes the output file and processes it for viewing with `xdvi`.

```
#include "stdio.h"
#include "TeXmerge.h"

int main(int argc, char **argv) {
TeXmergeName_t *array?
int count=3?
int ret?
FILE *fp?
char *out_pathname="temp.tex"?
char *form_pathname="test_form.tex"?

    array = TeXmerge_AllocNames(count)?
    TeXmerge_SetArrayEntry("THISVAR", "some value", &array[0])?
    TeXmerge_SetArrayEntry("THATVAR", "blah, blah", &array[1])?
    TeXmerge_SetArrayEntry("ANOTHERVAR", "la-te-dah", &array[2])?
    ret = TeXmerge_OpenOutput(out_pathname, &fp, 0)?
    if (ret != TXM_OK) {
        fprintf(stderr, "TeXmerge_OpenOutput(%s): %s\n", out_pathname,
TeXmerge_GetErrorString(ret))?
        return(1)?
    }
    ret = TeXmerge(form_pathname, array, count, 0)?
    if (ret != TXM_OK) {
        fprintf(stderr, "TeXmerge(%s): %s\n", form_pathname,
TeXmerge_GetErrorString(ret))?
        return(1)?
    }
}
```

```

}
TeXmerge_CloseOutput(fp)?
TeXmerge_View(out_pathname, TXM_WAIT)?
return(0)?

```

After the call to `TeXmerge_CloseOutput()` the contents of `temp.tex` would appear as follows:

```

\def\THISVAR{some value}
\def\THATVAR{blah, blah}
\def\ANOTHERVAR{la-te-dah}
\input test_form.tex
\bye

```

`test_form.tex` can have any TeX code of your choosing, including invocations of `\THISVAR`, `\THATVAR`, and `\ANOTHERVAR`.

5. Python Binding

A Python binding for the TeXmerge API is also available. A reimplementa-tion of the previous C-code is given below.

```

import TeXmerge
import sys

array = {'THISVAR': 'some value',
         'THATVAR': 'blah, blah',
         'ANOTHERVAR': 'la-te-dah'}
out_pathname = 'temp.tex'
form_pathname = 'test_form.tex'
try: fp = TeXmerge.openOutput(out_pathname)
except IOError, errmsg:
sys.stderr.write('TeXmerge.openOutput(%s): %s\n' %
                 (out_pathname, errmsg))
TeXmerge.merge(form_pathname, array, fp)
TeXmerge.closeOutput(fp)
TeXmerge.view(out_pathname, TeXmerge.TXM_NOWAIT)

```

Notes:

- The Python version is much more clean,
- Error detection via return values has been replaced with Python's exception mechanism. i.e. Instead of methods returning integer result codes they throw exceptions of the appropriate type which may be caught via the `try/except` construct,
- The `TeXmergeName_t` arrays used in the C-code example just use simple Python dictionary objects. As a result the nagging `count` integer with tracks the number of array elements is no longer needed.
- Constants defined in `TeXmerge.h` are accessed as attributes of the Python `TeXmerge` module.

6. Applications of `TeXmerge` at Texas Life

The `TeXmerge` API has been used to build a number of independent applications.

6.1. Interactive `TeXmerge`

The initial application of the API was in an X-windows program called `TeXmerge`. In this application a given directory holds a number of `TeX`-based form letters. The application allows the user to select a document, which is then scanned for merge variable names (via a call to `TeXmerge.GetNames()`). A frame is then displayed containing prompts and text-entry fields for each variable name. Once data is entered in all the text fields, the print function initiates the merge/print process, producing a completed document which may be viewed on-screen or printed.

At Texas Life the application was integrated with the policy administrative database in the following fashion. A policy database interface layer was built to return a associative array of selected basic data given a policy number. If one of the merge variables in the form document is named `POLNUM`, then when a value is entered in that field the policy database interface is used to make a query using the entered `POLNUM` value. If a match is found in the database, then other merge variable fields on the frame that have matching names in the policy data array are automatically populated with the corresponding retrieved data.

6.2. Policy Print

Each policy issued by Texas Life requires a *policy contract* to be printed. Most contracts are around twenty pages in length and include a large amount of variable data depending on the insurance product, the insured's issue age, smoker class, and other factors. Contracts also vary by the state in which they are written. Variable data is prepared by the policy administrative system, and the T_EXmerge API is used to merge the data into the appropriate contract form to produce a printable document. Because decisions may be made directly in the T_EX document (via `\ifx` and others) a single `.tex` file may be used to generate policy forms for multiple states.

6.3. Automated Correspondence

Texas Life's policy administrative system generates a large amount of correspondence to policyholders. These letters are form letters written in T_EX, and the T_EXmerge API is used to merge the data produced by the admin. system onto the appropriate letter form.

6.4. Customized Applications

Various other administrative applications that perform complex tasks have integrated the T_EXmerge API as the facility to produce specialized printed output.

6.5. Forms

Texas Life uses a number of pre-printed forms that must be filled out the policy holders, agents, internal employees, etc, to request specific tasks to be performed (i.e., change of address, change of beneficiary, etc) These forms are produced using T_EX. The resulting forms are used in two different ways. One is in the interactive T_EXmerge application where the form may simply be printed or filled out online. The second is for selected forms, PostScript and PDF output is placed on the Texas Life web site for downloading by appropriate users.

Forms-on-the-web is currently an active area a development. A work-flow system is being built that will turn simple downloading of forms into a transaction management system. Instead of policyholders or agents downloading forms, filling them out, signing them and returning them to Texas Life, they will initiate a transaction via their web browser and enter data online that they would normally write on a pre-printed form. After the data is entered a

“turn-around” key will be assigned to the transaction and the form will be printed populated with the data entered and a bar-coded turn-around key. The policyholder/agent mails the form to Texas Life (most forms require handwritten signatures) where it is scanned into the document management system and the specific transaction resumes processing via the turn-around key detected on the document.

6.6. Document Archival

Documents produced by all of the above systems are archived in a centralized repository call the document imaging system. Storing \TeX merge documents is efficient? only two items are stored to reproduce a document. One is a pointer to the `.tex` form document (there is a way to accommodate historical revisions of any specific document). The second is the associative data array for the variable data in the document. Each time the document is displayed, a sort of “JIT” technology is used. \TeX merge first merges the variable data into the form, then \TeX is run on the merged form, and finally, `xdvi` or `dvilj` is invoked to display/print the resulting document.