MetaPost developments: MPlib project report

Taco Hoekwater
Elvenkind BV
Dordrecht, The Netherlands
http://tug.org/metapost

Abstract

The initial stage of the MPlib project has resulted in a library that can be embedded in external programs such as LuaTEX, and that is also the core of the `mpost` program. This paper presents the current state of affairs, the conversion process of the MetaPost source code, and the application interface to the library.

1 MPlib project goals

The MPlib project is a logical consequence of the transfer of MetaPost development from its author John Hobby to the MetaPost development team. It originates from a desire to update MetaPost for use in the 21$^{\text{st}}$ century. The first thing that needed to be done to make that happen was updating the program source code and infrastructure to be closer to today's programming standards.

These days, programs are often written in the form of shared libraries, with a small frontend application. When written in this way, a program can not only be used as a standalone program, but can also easily and efficiently be (re)used as a plugin inside other programs, or turned into a multi-user system service.

With current MetaPost, such alternate uses are impossible because of the internals of the code. For example, MetaPost uses many internal global variables. This is a problem because when two users would be accessing a 'MetaPost' library at the same time, they would alter each other's variables. For another example, MetaPost has static memory allocation: it requests all the computer memory it will ever use right at startup. It never bothers to free that memory, because it counts on the operating system to clean up automatically after it exits. And one file example: MetaPost not only opens files at will, but it also writes to and, even more problematically, reads from the terminal directly.

A large part of updating MetaPost is therefore fixing all these issues. But while doing this, there are other weirdnesses to take care of at the same time.

The present subsystem for typesetting labels (`btex` ... `etex`) is pretty complicated, requiring an array of external programs to be installed on top of the normal `mpost` executable. And from a system viewpoint, the error handling of MetaPost is not very good: it often needs user interaction, and in most other cases it simply aborts. Finally, the whole process of installing the program is complicated: a fair bit of the TEX Live development tree is needed to compile the executable at all.

2 Solutions

Many of the problems mentioned above are a side-effect of the age of the source code: the source is largely based on METAFONT, and therefore written in Pascal WEB. And the bits that are not in Pascal WEB are an amalgam of C code borrowed from other projects, most notably pdfTEX.

Not wanting to lose the literate programming documentation, we had only one practical way to proceed: using the CWEB system. CWEB has the same functionality that Pascal WEB has, except that it uses C as the programming language instead of Pascal, and it has some extensions so that it does not get in the way of the 'normal' C build system.

Using CWEB, a single programming language now replaces all of the old Pascal and C code. The code has been restructured into a C library, the label generator `makempx` has been integrated, and compilation now depends only on the `ctangle` program and the normal system C compiler, so that a simple Autoconf script can be used for configuration of the build process.

3 Code restructuring

Whereas converting the C code of the font inclusion and label processing subsystems to CWEB was a fairly straightforward process, converting the Pascal WEB core of MetaPost was a more elaborate undertaking.

In the first stage, the Pascal code within the WEB underwent an automatic rough conversion into C code. Afterwards, the generated C code was manually cleaned up so that it compiled properly using `ctangle`. This part took roughly one month, and the end result was an executable that was 'just like'

Pascal MetaPost, but using CWEB as source language instead of Pascal WEB.

After that was done, the real work started:

- All of the global variables were collected into a C structure that represents an 'instance' of MetaPost.
- The Pascal string pool was stripped away so that it is now used only for run-time generated strings. Most internal functions now use normal C strings.
- The PostScript backend was isolated from the core of the program so that other backends can be added much more easily.
- All of the exported functions now use a C namespace.
- Where it was feasible to do so, MPlib uses dynamic memory allocation. There are a few exceptions that will be mentioned later.
- All input and output now uses function pointers that can be configured through the programming interface.
- The MPlib library never calls `exit()` itself but instead returns to the calling program with a status indicator.

4 Using the library from source code

Using the MPlib library from other program code is pretty straightforward: first you set up the MPlib options, then you create an MPlib instance with those options, then you run MetaPost code from a file or buffer, and finally finish up.

The options that can be controlled are:

- various command-line options that are familiar from `mpost`, such as whether MetaPost starts in INI mode, the `mem_name` and `job_name`, 'troff' mode, and the non-option part of the command line,
- the size of the few remaining statically allocated memory arrays,
- various function pointers like those for input and output, file searching, the generator function for typeset labels, and the 'editor escape' function,
- the start value of the internal randomizer,
- and finally a 'userdata' pointer that is never used by MPlib itself but can be retrieved by the controlling program at any time.

The application programming interface at the moment is very high-level and simplistic. It supports two modes of operation:

- emulation of the command-line `mpost` program, with traditional I/O and interactive error handling,
- an interpreter that can repeatedly execute individual string chunks, with redirected I/O and all errors treated as if `nonstopmode` is in effect.

For the string-based interpreter, there are a few extra functions:

- the runtime data can be fetched; this comprises the logging information and the internal data structure representation of any generated figures,
- the instance's error state can be queried,
- the userdata pointer can be retrieved,
- some statistics can be gathered,
- PostScript can be generated from the image output,
- and some glyph information can be retrieved; this is useful if you want to create a backend yourself.

4.1 Examples

Here is a minimalistic but complete example that uses the `mpost` emulation method in C code:

```
#include <stdlib.h>
#include "mplib.h"
int main (int argc, char **argv) {
  MP mp;
  MP_options *opt = mp_options();
  opt->command_line = argv[1];
  mp = mp_initialize(opt);
  if (mp) {
     int history = mp_run(mp);
     mp_finish(mp);
     exit (history);
  } else {
     exit (EXIT_FAILURE);
  }
}
```

Given the basic library functionality now available, it is reasonably straightforward to create bindings for other languages. We have done this for Lua, and here is a second example that uses these Lua language bindings. The Lua bindings are always based on string execution, and the option setting and instance creation are merged into a single `new` function:

```
local mplib, mp, l, chunk
mplib = require('mplib')
mp    = mplib.new ({ini_version = false,
                    mem_name    = 'plain'})
chunk = [[
        beginfig(1);
          fill fullcircle scaled 20;
        endfig;
        ]]
if mp then
```

```
  l = mp:execute(chunk)
  if l and l.fig then
    print (l.fig[1]:postscript())
  end
  mp:finish()
end
```

5 Using the command-line program

On the command line very little has changed. The executable `mpost` still exists. Now it is merely a thin wrapper that is much like the C code example shown earlier, except with a few hundred more lines because it has to set up the command-line properly.

As mentioned already, the `makempx` functionality has also been converted into a small library that is used by `mpost` to emulate the old label creation system. The programs `makempx`, `dvitomp`, `mpto`, and `dmp` have been merged into this library and no longer exist as separate programs. For backward compatibility, a user-supplied external label generation program will be called if the MPXCOMMAND environment variable is set, but normally `mpost` sets up the MPlib library to use the new embedded code.

In the normal case, the only external program that will be run is the actual typesetter (TEX or Troff). The command-line of `mpost` is extended to allow the specification of which typesetter to use.

6 Planning and TODO

Most development took place at the beginning of 2008, after which we entered a period of extensive testing. This way we were relatively confident that the first version of the library was basically usable from the start.

The first beta release (1.091) was presented at the TUG 2008 conference. The distribution contains the MPlib library source, the code for the 'mpost' frontend, code for the Lua bindings, and the C and Lua API documentation.

The final MPlib 1.100 release will be released later in 2008, and the MPlib-based distribution will replace the Pascal MetaPost distribution from that point forward.

After this release, work on the TODO list will continue. Items already on the wishlist:

- Start using dynamic memory allocation for the remaining statically allocated items: the main memory, the number of hash entries, the number of simultaneously active macro parameters, and the maximum allowed input nesting levels.
- An extension is being planned under the working name 'MegaPost' that will extend the range and precision of the internal data types.
- In the future, we want to use MPlib to generate (OpenType) fonts. This requires support from the core engine like overlap detection and calculation of pen envelopes.
- Error strategies are planned so that the behaviour of the string-chunk based interface can be configured properly.
- There are desires to expand the API. For instance, it would be nice if applications were able to use the equation solver directly.

7 Acknowledgements and contact

The MPlib project could not have been done without funding by the worldwide TEX user groups, in particular: DANTE, TUG India, TUG, NTG, CSTUG, and GUST. A big thank you goes to all of you for giving us the opportunity to work on this project.

The general contact information for MetaPost and MPlib has not changed:

- Web site and portal:
  `http://tug.org/metapost`
- User mailing list:
  `http://lists.tug.org/metapost`
- Source code and bug tracker:
  `http://foundry.supelec.fr/projects/metapost`