Integrating the pool file

Taco Hoekwater

Abstract
This short article discusses the method that is used in MetaPost and luaTEX to integrate the string pool file into the program.
This method allows the redistribution of a single updated executable in place of both a program and a data file, and this makes updating those programs easier on both the user and the developer (me).

How a pool file is created

The readers who regularly update their (pdf)TEX or MetaPost executables will probably be familiar with the concept of pool files already, but I will explain the mechanics in some detail.

Programs written in the WEB language normally do not contain the strings inside the executable proper, but in a separate file, called the 'pool file'.

The most important reason for the existence of this file is that back when Knuth was working on TEX and Metafont, there was not yet a standardized way to handle strings inside the Pascal language, so he had to invent his own solution for printing messages and warnings.

In order to illustrate what is in a pool file, I will show you the required steps. First, here is a bit of WEB source from MetaPost:

```
...
if minx_val(h)>maxx_val(h) then
  print("0 0 0 0")
else begin
  ps_pair_out(minx_val(h),miny_val(h));
  ps_pair_out(maxx_val(h),maxy_val(h));
  end;
print_nl("%%Creator: MetaPost ");
print(metapost_version);
print_nl("%%CreationDate: ");
```

this excerpt is from one of the PostScript output routines. Here, there are still recognizable strings that are used as function arguments (as well as the symbolic value `metapost_version`, that is actually a macro resolving to a string).

The processor `tangle` converts this input into a proper Pascal source file. While doing so, it resolves all of the many WEB macros that are present in the code. `metapost_version` is one of those, but also the constructs like `minx_val(h)` and `maxx_val(h)`. It also removes the underscores from function names, because traditional Pascal compilers did not allow _ to appear in identifiers.

What we are focusing on now, is that it also collects all of the double–quoted strings in the input. It puts all of the unique multi–character strings into an internal array, and replaces the actual string in its output with the index number it has given the string inside that array. Of course, functions like `print()` are written in such a way that they expect numbers as arguments instead of string values.

The Pascal output file looks like this:

```
...
if mem[h+2].int>mem[h+4].int then print(1279)
else begin pspairout(mem[h+2].int,mem[h+3].int);
pspairout(mem[h+4].int,mem[h+5].int);end;
printnl(1281);print(256);printnl(1282);
```

As you can see, this file is clearly intended for a compiler only. The complete lack of indentation makes it near impossible for a human to read the generated code, but of course a Pascal compiler has no problem with it.

Nowadays, creating an executable program from the WEB source file happens in a few extra steps, and one of these steps is a conversion from Pascal to C source code, by means of the `web2c` system. You may find the output of `web2c` easier to read, because it re-indents the code for human reading:

```
...
if ( mem [h + 2 ].cint > mem [h + 4 ].cint )
  print ( 1279 ) ;
else {
  pspairout(mem [h + 2].cint,mem [h + 3].cint);
  pspairout(mem [h + 4].cint,mem [h + 5].cint);
}
printnl ( 1281 ) ;
print ( 256 ) ;
printnl ( 1282 ) ;
```

So, where did the strings go? `tangle` put the multi–character strings into a separate file, in this case named `mp.pool`. Each line of that file contains two digits indicating the length of the string, followed by the string itself. Around line 1000, you will find this:

```
...
070 0 0 0
20%%HiResBoundingBox:
20%%Creator: MetaPost
16%%CreationDate:
...
```

07 is the length in bytes of '0 0 0 0', 20 is the length of '%%HiResBoundingBox: ', including the trailing space character, etcetera. Single character strings are not written to the pool file, because there is no need: all single–character strings simply have an assumed index value matching their contents, and the first string in the `pool` file receives index number 256.

The Pascal source code (or C source code) is now converted into an executable, and you end up with `mpost.exe` as well as `mp.pool`. The pool file is stored somewhere in the TEXMF tree, and one of the very first things that the `--ini` version of MetaPost does, is that it reads `mp.pool` to initialize its internal arrays. When the user executes the `dump` command, MetaPost writes all of the string items to the `.mem` file, from where it will be retrieved by production runs of MetaPost.

There is nothing wrong with this system as such. In fact, it has worked well for nearly 30 years. But it does make updating executables a bit harder than desired: users not only have to copy the actual program to a folder in the path, but they also have to figure out where to place the new and improved `mp.pool` file.

As the maintainer of MetaPost and luaTEX, both programs that are updated frequently, I was getting annoyed with having to explain to almost each updating user what a pool file was, why it was important, and where it should go in their TEXMF tree.

How a pool file disappears again

So I decided to do something about it, and that was how the `makecpool` program was born. The concept is very simple: it converts the `mp.pool` into a C source file named `loadpool.c`. In fact, it is so obvious that the idea has been proposed a number of times already, for instance by Fabrice Popineau. But somehow it has never made it to the core TEX distribution yet.

The structure of the created file is straightforward: there is one big static array, and a fairly simple C function that replaces the Pascal procedure for pool file reading. In abbreviated form, `loadpool.c` looks like this:

```
/* This file is auto-generated by makecpool */

#include <stdio.h>
#include "mpdir/mplib.h"

static char *poolfilearr[] = {
  "1.000",

  ...

  "0 0 0 0",
  "%%HiResBoundingBox: ",
  "%%Creator: MetaPost ",
  "%%CreationDate: ",

  ...

  NULL };

int loadpoolstrings (integer spare_size) {
  char *s;
  strnumber g=0;
  int i=0,j=0;
  while ((s = poolfilearr[j++])) {
    int l = strlen (s);
    i += l;
    if (i>=spare_size) return 0;
    while (l-- > 0) strpool[poolptr++] = *s++;
    g = makestring();
    strref[g]= 127;
  }
  return g;
}
```

In the stage where the various C files are compiled into `mpost.exe`, this file is included in the list, and in that way the strings will be embedded in the program. At run-time, the C function is called to put the strings for the C array into the internal storage area instead of the original file reader.

The result: there is only one single executable file that can be freely distributed to the users. The source code for `makecpool` is part of the MetaPost and luaTEX distribution package.

Taco Hoekwater
taco(a)elvenkind.com