# MetaPost library project

## Introduction

The purpose of this paper is to document the targets and implementation milestones of the MetaPost library project (MPlib). It is intended to serve both as a guideline to the developers and as a monitoring tool for the funding providers.

When it was circulated in the spring of 2007, the MetaPost library proposal paper [1] mentioned two main problem areas that are to be directly tackled by the MPlib project:

- current MetaPost cannot be used as a software component
- the handling of external text labels is becoming outdated

The proposal paper identifies those problem areas and presents a rough split in sub-tasks. The main goal of that paper was to gain funding, and because of that, but also because an in-depth analysis was still missing at that moment in time, it was very short on details.

Immediately after funding was secured, a period of analysis started. That initial information-gathering stage is now finished, and the gained knowledge can be used to further detail the objectives and stages of the project.

## Main objectives

The term 'software component' as used above is a loosely defined term. In practice it means that the current MetaPost code should be split into a core library and a client application, with the latter being as small as possible and the former allowing multiple concurrent usage.

Besides the general goal of creating a reusable component on one particular platform, the project has to make sure that MetaPost maintains at least the current level of portability, and that it keeps on having 100% identical behaviour and output across all computer platforms it runs on.

Regardless of implementation details, it is clear that for simplified maintenance and portability, the current code base has to be unified using a single programming language. A literate programming language is desired: we (the TEX community) should do its best to keep the Knuthian tradition of programs that document their implementation alive, whenever that is possible without major extra effort.

Converting the MetaPost core into a library implies adding an indirection layer to the already existing input and output subsystems: component libraries should not interact with the user directly unless explicitly asked to do so. The new to-be-written internal interface will serve to allow the configuration of logging and error handling.

A well-defined and documented interface to the internals of the program is needed to make it possible to add new back-ends as alternatives to the already existing Adobe Postscript generator.

## Implementation language

After some consideration, we believe the best choice for the implementation of the core library itself is the C programming language. There are a number of possible arguments to be made in favour of various other languages, but it seems that C has the best overall characteristics for the task at hand:

- there already is a literate programming system available for C that is very close to the current Pascal–based implementation: cweb [2].
- the C language is structurally very similar to pascal, so that identical algorithms can be used in the conversion of the existing code base.
- both the run-time speed and compactness of compiled C code are very high.
- C source code is itself very portable, and C libraries binds easily to other applications and programming languages.
- the market penetration of C is very high, aiding deployment.
- some parts of the current MetaPost are already written in C (although in a non-literate fashion).

Two specific applications of the MPlib core library are planned as part of the current project:

- a command-line C program
  This program will replace the current `mpost` executable, for redistribution in TEX distributions.
- a Lua language binding.
  This will allow the library to be easily integrated into LuaTEX [3], and it will also serve as an example of binding MPlib.

## Tasks

From the previous paragraphs, a list of sub-tasks can be derived.

1. **Conversion** and unification of the current sources into cweb.
2. Integrating the hundreds of global variables that are used by the current source into an **single data structure** (instance).
3. The addition of a **indirection layer** for all the input and output streams.
4. Adding an interface for configuration of **error handling**.
5. Writing a **Lua module** to interface to the MetaPost library.
6. Consolidating and documenting an **interface to the internal structures**.
7. Designing and implementing a new high-quality **labelling system**.

### Conversion

There are two sub-tasks: the conversion of the current Pascal Web code, and the conversion plus documentation of the current C code, resulting in a unified source.

***Conversion of current Pascal code.*** Because all of the Pascal code uses a pretty small subset of the many possible variations on the basic Pascal language, and because the cweb system itself is nearly identical to Pascal Web, using a machine-assisted approach to the translation is not only feasible but also the most effective.

A Lua script will take care of the conversion from Pascal to C while keeping the Web extensions intact. The choice of the scripting language Lua [4] is based on three considerations:

1. Efficiency of the conversion program is not relevant, as it will be used only once.
   Therefore, there is no reason to choose for a compiled language like C, and scripting languages allow for a much faster development cycle.
2. Lua has a module that allows script programs to be written that use Parsing Expression Grammars [5, 6] to parse input.
   This makes it very easy to write a proper parser for web-based programs like MetaPost.
3. Familiarity with the language.
   It would not be worth learning a different programming language just to do this conversion step, and of the languages Taco knows, Lua is best suited to the task at hand.

***Conversion of current C code.*** The main MetaPost program is written in Pascal Web, but some (about 5000 lines, mostly the font inclusion) is written directly in C code. This code is not currently not written in a literate programming fashion, and it will be worthwhile to do that conversion. This has to be done by hand, but the amount of code is not that large and it will help to create a more consistent distribution and build process.

### Single data structure

For a code library to be as widely usable as possible, it should not use any non-local variables. Instead, a code library normally defines a single function, that will allocate and return a data structure that will from then on be passed on from one library function to the next until the application is done with it.

The current internals have to reworked extensively to allow this: MetaPost as it stands uses about three hundred global variables. Some of these are just for internal communication between functions (a side–effect of limitations in Pascal Web), and some are just global constant values, but most are used to store the internal state of the MetaPost engine.

Actions needed:

- all of the globals have to be incorporated into a big single structure.
- nearly all functions have to be altered to use this structure are the first argument.
- the memory `dump/undump` code has to be rewritten to handle this new data model correctly.

When this task is complete, the cweb code can provisionally be split into a library and an application.

For the moment, the Knuthian heap-based memory management and string pool handling will be converted to C without functional changes, but at a later moment these bits may be converted to use dynamic memory allocation throughout.

### Indirection layer

We envisage that it will be possible to register callback functions that will make it possible to override default behaviour that is very close or identical to the current state.

The advantage of setting it up in this way is that application programs that want to be compatible with the way the old `mpost` executable can be quite small, and for the moment at least, this seems desirable.

### Error handling

To allow the user to configure the error handling, first the existing error handling code has to be unified so that all the possible errors pass through the same interface. After that is done, it will be possible to add a callback there to allow configurability.

### Lua module

At the base functionality level, this is a straightforward process of interfacing C functions to Lua scripting, that can handled almost totally in an automated fashion by already existing tools like [7].

But optimization of the user interface so that it is easy to use has to be done manually. Experiences has shown that C libraries almost always are too low-level for a scripting language, and the necessary glue code has to be written by hand.

Because much of this glue code will itself depend on user feedback, it follows that while a first version of Lua module will probably be available at the time of the first release of MPlib(summer 2008), the module will probably not be finalized until quite time afterward.

### Interface to the internal structures

While we are strictly dealing with the problem of how to use MetaPost as a component, it is sufficient to just have a 'constructor' function, a 'destructor' function, and a 'run' function.

But that is not good enough if we want to alter the program, for example by adding a new back-end. For this, it is necessary to have access to at least some of the actual internals of the program.

For such interfaces to be useful, a bit of internal cleanup is needed:

- Some of the interface structures are not as clean cut as they could be (or should be, according to current practice.

  Some cleanup is needed to provide a clean front.
- most of the required documentation is already present in the Pascal Web source, but not presented in the best possible way.

  This documentation should be copied out of the source and placed in a separate manual.

### Labelling system

When all of the above are complete, it is time to think about a new external labelling system to replace `btex ...etex`.

New insights are expected to come from the use of MPlib inside LuaTEX, so writing this section now would be premature.

## Time line

The development of MPlib is expected to progress linearly through the first few point, up to first basic 'Lua module' implementation. After that, work will continue more or less simultaneous on the remaining items.

The stage 'single data structure' is expected to be complete around end of February, and everything up to the basic 'Lua module' will be presented at the Cork TUG meeting in the summer of 2008.

## References

[1] Hans Hagen and Taco Hoekwater. *Metapost library proposal*. The Netherlands, 2007.

[2] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0*. Addison-Wesley, Reading, MA, USA, 1993.

[3] Hans Hagen and Taco Hoekwater.
http://www.luatex.org
The official LuaTEX website.

[4] Roberto Ierusalimschy, Waldemar Celes and Luiz Henrique de Figueiredo.
http://www.lua.org
The official Lua website.

[5] Bryan Ford.
http://pdos.csail.mit.edu/~baford/packrat/
This website explains the theory behind Parsing Expression Grammars, and Packrat Parsing in general.

[6] Roberto Ierusalimschy.
http://www.inf.puc-rio.br/~roberto/lpeg.html
The homepage of the Lua module that implements PEGs.

[7] Waldemar Celes.
http://www.techgraf.puc-rio.br/~celes/tolua
The homepage of toLua, a program that generates C/C++ to Lua bindings.