Libraries for LUATEX

Luigi Scarso
Hans Hagen
October 2012

Introduction

For a few years LUATEX has proven to be a stable engine. Although we're not yet finished, it is time to start thinking about how to deal with external libraries. There have been experiments with as well as some publications about this topic. However, since TEX distributions and macro packages need predictable and stable components it makes sense to formalize usage of external libraries. For this we need clear descriptions, an infrastructure and a couple of examples. Of course users can always add more if needed.

In this perspective we have done some experiments and came to the conclusion that it makes sense to start a project that deals with these matters. First we explain how is possible to integrate an external library written in C/C^{++} in LUATEX. We explain the methodology used (SWIG) and the resources (i.e. the tools chain, the documentation) needed to accomplish the goal.

SWIG

SWIG (Simplified Wrapper and Interface Generator) is a program that helps to build an interface between a C/C^{++} library and another target language (in our case LUA). The interface is a binary module loaded at runtime by the target language and has the benefit of a direct access to library without the need of an intermediate program, which usually means a considerable gain in performance. Given that LUATEX embeds a full LUA interpreter, SWIG can therefore be used to extend LUATEX at runtime with a C/C^{++} library, building in this way a set of plugins for LUATEX.

SWIG is distributed as as binary for MS WINDOWS and as source for LINUX, and there are packages for most common LINUX distributions. It is under active development: the current version is 2.0.8 dated August 2012 while the first version is dated February 1996.

Integration with the regular TEX source tree

The way of building an interface is specified in an interface file (*the driver*). SWIG uses the driver to parse the headers of the library and to produce a C file. This file then needs to be compiled to obtain a static or shared module, a LUATEX module in our case. This means that in order to build a wrapper module, the headers of the library and the library itself must be available, and in order to be usable, the module must be linked with the library. The headers are not strictly necessary but, given that they are the ultimate reference of the API and it's a good practice to package them with the library. This means that a TEX installation like TEXlive should hosts both the library and the headers files. It is also important to note that the LUA specification is quite stable, so a change of the version (like a transition from a 5.1.4 and 5.2) doesn't pose particular problems when rebuilding libraries.

Note that we focus on the binary part. Generating (a reasonable subset of) libraries could become part of generating LUATEX binaries. How these libraries are used is up to the macro packages (cf. the LUATEX philosophy). We don't provide additional LUA wrapper code[1]

Platforms supported

The compilation and linking of an interface module requires the tool chain (at least the compiler and the linker) of the original library. For example if a library is compiled with the MICROSOFT compiler, then the interface module requires the MICROSOFT compiler and linker too. The driver can be composed in a modular fashion to match the tool chain of the library. The express edition of a MICROSOFT compiler, the GCC suite and the MINGW suite practically covers the majority of cases: On the other side, if a library is not provided for a given platform, it is of course impossible to produce the interface and hence the platforms supported depend from the availability of the library for those platforms.

Adherence to the library API

The interface module usually follows faithfully the API of the library, but the driver can be edited to ignore or rename some functions, in case they are out of scope or clash with LUATEX. For example if a database has both the

[1] It is quite likely that we will produce additional code but that will then be part of the CONTEXT distribution.

server and client set of APIs described in the headers files and the server APIs are irrelevant for the interface, the driver has ways to ignore them.

It is important to stay close to the original API for at least the following reasons:

1. By using the same methods as the original we can use the original documentation.

2. The less we wrap code, the less likely it becomes that stable workflows will suffer from incompatibilities.

3. By using the SWIG method, we don't depend on (possibly experimental and/or possibly not maintained third party LUA libraries.)

4. Macro packages can wrap the code in ways that suit usage in TEX best.

Even if LUATEX will always be a moving target (as extensibility is part of the concept) we want to keep the TEX tradition of long term stability as much as possible.

Documentation

The following aspects are mandatory or at least important to document:

1. How to write and compile SWIG drivers. These are quite general and most information comes from the (large) SWIG manual and the interface files of the library. A good knowledge of C/C^{++} is needed, but as C and C^{++} are stable all can be documented well.

2. What is needed to write a SWIG driver for at LUA. This requires a good knowledge of LUA (memory management, mapping onto known data structures). Keep in mind that LUATEX can also operate as stand alone LUA interpreter which helps to keep TEX based workflows consistent with the formatting engine.

3. What additional interfacing is needed for LUATEX. We don't expect too many issues here.

4. It should be well documented how to use the interface. Examples have to be valid for the current version of LUATEX.

Past experiences

We already have some experiences with using libraries in LUATEX but it never went beyond playing around:

- pari/gp (GCC)
- mupdf
- gsl
- leptonica.
- curl
- qpdf (GCC, MINGW)
- graphicmagick (GCC, MINGW)
- mysql (MS WINDOWS cl)

There are papers that show examples of the use of PARI/GP (*Extending CONTEXT MkIV with PARI/GP*, EUROBACHOTEX 2011) and GraphicMagick under CONTEXT MKIV (*Extending CONTEXT with GraphicMagick: When bitmap beats vector*, Fifth CONTEXT meeting, 2011). They should be considered experimental because the lack of extensive testing.[2]

Conclusion

Extending LUATEX with an external library opens new possibility to employ LUATEX. To some extend LUATEX itself might benefit of a set of plugins, because it can delegate certain tasks to external modules.

Of course in a running system one needs to make sure that such libraries don't conflict and macro packages should offer a consistent way to deal with them. However, this is not much different from managing components that make up a macro package.

Proposal

Hereby we propose a project funded by TEX user groups that will lead to the following outcomes:

[2] In fact, experiments with MYSQL binding are the main reason for starting this project: existing libraries were unstable (or broken) and hard to compile, so we looked into the SWIG way to get it done.

1. An infrastructure for building libraries for LUATEX.
2. Documentation on how to use this infrastructure and how to add more.
3. A couple of working examples of libraries, available for all major platforms.

This project will be done by Luigi Scarso. The CONTEXT community already has a system set up for compiling intermediate versions of LUATEX so it makes sense to use that network, so that we can test all main platforms. Luigi himself has access to MS WINDOWS and LINUX computers (although they might need upgrading) and for MACOSX we can use cross-compilers. For the time being we can use the module reposititory for distributing components.[3]

We have to choose some not so straightforward libraries as examples. For instance graphicmagic depends on other libraries, as does curl, while mysql has some platform related issues. It also makes sense to look into some (scientific) graphic libraries. If we can handle complex libraries, supporting simple text manipulation ones are trivial.

In order to map data provided by libraries conveniently at the LUA end we will look into a consistent set of helpers. For example, the MYSQL library returns arrays representing rows of data, and for large quantities it is more efficient to map this onto a indexed LUA table. At this moment we only have some rough experience with this but we will investigate this in more detail.

A first estimate of the work to be done and time needed indicate that we need support in the range 8.000–10.000 €. Luigi can spend 50% of his time on the project. Around April 2013 we expect most to be in place. This gives enough time to deal with integration in TEXlive. Around the next code freeze of TEXlive all can be moved into the regular source trees. At the next CONTEXT conference (end of September 2013, Czech) Luigi and Hans hope to present some use cases.

While preparing this project proposal, we talked to NTG, DANTE and CSTUG representatives in order to check if such a project is feasible and so far we

[3] Hans will do testing and explore possibilities with Luigi and Taco will help out with specific interfacing issues, especially when we move on to LUA 5.2, but the funding is meant for Luigi.

got positive reactions. As this is an important milestone in the development of LUATEX we hope that user groups will consider supporting it.

Follow up

In a next stage we want to look into two way communication. For instance it would be nice to have the ability to use a variant of GSL Shell http://www.nongnu.org/gsl-shell/) as a library. This is a LUA wrapper around GSL. Here the challenge is to share the LUA states. There might be other candidates. Our main objective here is to have powerful extensions that can for instance be used to create graphics, in which case we have a close cooperation between LUATEX, METAPOST and such a library. At this moment we have no indication of how much funding is needed for this.