
Libraries
basics

Contents

1 Introduction

Since we started developing LuaTEX several methods have been explored with regards to external libraries. Now, before we go into details, it must be said that in practice this feature is hardly needed. If someone really needs it, it is likely to be in a setting where one can also write some library interface and compile LuaTEX to suit it. So, what we're talking of here is probably not of interest for all of you.

In the perspective of ConTEXt a dependency on a library is not what we have in mind when we advocate long term usage (and stability) of a workflow. If you see TEX as an independent component but still depend on its use of libraries you might consider having a backup plan in case a library is no longer maintained, of when it gets replaced by another favourite, as happens. There are several ways to use a library, and we mention a few here.

The first one is to use Lua *wrapper* libraries that interface to some *specific* library. This is what you do when you use Lua as stand alone program. In that case you depend on someone cooking up an efficient and reliable interface. Then you depend on the author or others to provide the binaries. If there is only one target platform you can wonder if you like that additional dependency. Another aspect to keep in mind is that such a wrapper library has to match both the used library and the version of Lua that you use. When we use this method in LuaTEX, one also has also to make sure that the LuaTEX binary is compiled in a way that permits loading (this has to do with exposing symbols that need a runtime binding).

Because this didn't work out well in practice, already before version 1.0 showed up we explored a flexible way to create libraries suited for LuaTEX. This project was tagged "SwigLib". An infrastructure was created and a couple of example library interfaces were provided. However, in practice this never catched on and we don't expect distributions to provide libraries in addition to the main LuaTEX binary, but the framework is there for those who want to play with it.

Not long after we released LuaTEX 1.0, we started experimenting a bit more with so called foreign function interface: ffi. Originally that interface to external libraries was only available in LuajitTEX, but a good and compatible alternative is now also available in the normal engine too. For users it is not that relevant to know how it works, as long as it works. It means that in addition to SwigLib we have a method that doesn't demand compilation as it uses normal (public) libraries.

A last alternative we mention is to just add the libraries to the LuaTEX engine. In fact, this happens: the MetaPost binary has been provided this way for quite a while now. In LuaMetaTEX

for instance some more math related libraries were added, simply because the overhead is not that large and because it is a way to extend MetaPost beyond what it provided out of the box.

Before we proceed, ask yourself this questions: “Do I really need more libraries?” For instance does it really make sense to use Curl, Ghostscript or GraphicsMagick libraries while the command line version is (normally) just as efficient and avoids a dependency. This is even more true if you realizes that for instance a fetch or conversion only needs to happen once per run or in fact only when there is some change in the resource. On the other hand, when accessing databases one can avoid the often slower command line calls and save the hassle of intermediate files. Here efficiency wins. Also, when ConTEXt is used in a high performance database backend application a distribution and the used libraries are not updated on a daily basis. Anyway, if the answer to the question, and it can depend on what variant of ConTEXt you use: MkIV or lmtx, is “Yes!” then read on.

2 Usage in MKIV

Although there are (still) some examples of SwigLib in MkIV we now assume that the ffi interface is used. Apart from some experiments we currently can use ffi interfaced libraries in:

module	**library**	**windows**	**unix**
util-crl	curl	libcurl	libcurl
util-sql-imp-ffi	mysql	libmysql	libmysqlclient
util-sql-imp-sqlite	sqlite	sqlite3	sqlite3

The profiler that we occasionally use to identify bottlenecks in the engine (for instance when we upgrade Lua) uses ffi to provide access to the high resolution timers but this is typically different per platform.

One problem with libraries, especially on Windows is that the library is found on some system path and it can happen that multiple programs ship the same library but in different versions. You can try to play safe and put libraries in the TEX tree, for instance on my system they are in:

```
c:/data/tex-context/tex/texmf-win64/bin/lib/luatex/foo/libfoo.dll
```

You can trace where libraries are looked for with:

```
\enabletrackers[resolvers.ffilib]
```

or in Lua with:

```
trackers.enable("resolvers.ffilib")
```

The library is first located on one of the valid tds paths (these are sort of standardized in TEX distributions) and then using the normal ffi loader.

3 Usage in LMTX

In ConTEXt lmtx we only have so called ‘optional’ libraries. Actually they are just simple interfaces that register themselves in the `optional` namespace and that can load the real library at runtime. The approach is rather lightweight in the sense that compilation of the LuaMetaTEX binary doesn’t depend on additional resources (there is no need to have the libraries on the system or their source code to be present) and that at runtime there is no need to have the libraries present. This sounds similar to delayed loading of Lua wrapper libraries but the difference is that there is no potential Lua version clash and we also still have one single (relatively small) binary.

This approach works ok because in practice only very few users need a library and when they do they have to compile the interface anyway. And, as they compile, they can as well add a few lines that are needed to integrate their library to the optional mechanism. If you depend on such extensions, you quite certainly know what you’re doing and need to maintain a code base very careful.

There are some optional libraries present, like:

module	**library**	**windows**	**unix**
libs-imp-curl	curl	libcurl	libcurl
libs-imp-mysql	mysql	libmysql	libmysqlclient
libs-imp-sqlite	sqlite	sqlite3	sqlite3
libs-imp-zint	libzint	libzint	libzint
libs-imp-ghostscript	ghostscript	gswin64	libgs
libs-imp-graphicsmagick	graphicsmagick	several	unknown

It is still to be decided how these will be used. An example of a library that can be used out of the box is zint for using barcodes: see `m-zint.mkxl` for an example of its usage. The sql libraries have their own manuals and have been around for a while. They have a common encapsulating infrastructure written in Lua.

It is best to keep libraries in a place where you can keep an eye on them being updated, like

```
c:/data/tex-context/tex/texmf-win64/bin/lib/luametatex/foo/libfoo.dll
```

You could of course use the ones provided by the system or maybe use symbolic links so that you still keep en eye on changes. The ConTEXt distribution might at some point provide the libraries that we support.

4 Colofon

author Hans Hagen, PRAGMA ADE, Hasselt NL
version February 9, 2020
website www.pragma-ade.nl – www.contextgarden.net
comment many thanks to Luigi Scarso for taking care of ffi support in the engines