1 Introduction

The SwigLib project is related to Lua\TeX{} and aims as adding portable library support to this \TeX{} engine without too much fixed binding. The project does not provide Lua code, unless really needed, because it assumes that macro packages have different demands. It also fits in the spirit of \TeX{} and Lua to minimize the core components.

The technical setup is by Luigi Scarso and documentation about how to build the libraries is (will be) part of the SwigLib repository. Testing happens with help of the Con\TeX{}t (garden) infrastructure. This short document only deals with usage in Con\TeX{}t but also covers rather plain usage.

The set of supported libraries in the SwigLib subversion trunk is just a subset of what is possible and we don’t see it as the responsibility of the Lua\TeX{} team to support all that is around. The subset also serves as an example for other libraries. We also don’t ship wrappers (other that those used in Con\TeX{}t) as this is delegated to the macro packages.

2 Inside CON\TeX{}T

The recommended way to load a library in Con\TeX{}t is by using the \texttt{swiglib} function. This function lives in the global namespace.

\begin{verbatim}
local gm = swiglib("gmwand.core")
\end{verbatim}

After this call you have the functionality available in the \texttt{gm} namespace. This way of loading makes Con\TeX{}t aware that such a library has been loading and it will report the loaded libraries as part of the statistics.

If you want, you can use the more ignorant \texttt{require} instead but in that case you need to be more explicit.

\begin{verbatim}
local gm = require("swiglib.gmwand.core")
\end{verbatim}

Here is an example of using such a library (by Luigi):

\begin{verbatim}
\startluacode
local gm = swiglib("gmwand.core")
\end{verbatim}
local findfile = resolvers.findfile

if not gm then
    -- no big deal for this manual as we use a system in flux
    logs.report("swiglib","no swiglib libraries loaded")
    return
end

gm.InitializeMagick(".")

local magick_wand = gm.NewMagickWand()
local drawing_wand = gm.NewDrawingWand()
local pixel_wand = gm.NewPixelWand();

gm.MagickSetSize(magick_wand,800,600)
gm.MagickReadImage(magick_wand,"xc:gray")

gm.DrawPushGraphicContext(drawing_wand)

gm.DrawSetFillColor(drawing_wand,pixel_wand)

gm.DrawSetFont(drawing_wand,findfile("dejavuserifbold.ttf"))
gm.DrawSetFontSize(drawing_wand,96)
gm.DrawAnnotation(drawing_wand,200,200,"ConTeXt 1")

gm.DrawSetFont(drawing_wand,findfile("texgyreschola-bold.otf"))
gm.DrawSetFontSize(drawing_wand,78)
gm.DrawAnnotation(drawing_wand,250,300,"ConTeXt 2")

gm.DrawSetFont(drawing_wand,findfile("lmroman10-bold.otf"))
gm.DrawSetFontSize(drawing_wand,48)
gm.DrawAnnotation(drawing_wand,300,400,"ConTeXt 3")

gm.DrawPopGraphicContext(drawing_wand)

gm.MagickDrawImage(magick_wand,drawing_wand)

gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.png",1)
gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.jpg",1)
gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.pdf",1)

gm.DestroyDrawingWand(drawing_wand)
gm.DestroyPixelWand(pixel_wand)
gm.DestroyMagickWand(magick_wand)
\stopluacode
In practice you will probably stay away from manipulating text this way, but it illustrates that you can use the regular ConTeXt helpers to locate files.

You’d better make sure to use unique filenames for such graphics. Of course a more clever mechanism would only run time consuming tasks once for each iteration of a document.

3 Outside CONZeXT

In the ConTeXt distribution we ship some generic macros and code for usage in plain TeX but there is no reason why they shouldn’t work in other macro packages as well. A rather plain example is this:

\input luatex-swiglib.tex

\directlua {
    dofile("luatex-swiglib-test.lua")
}

\pdfximage {luatex-swiglib-test.jpg} \pdffileximage\pdflastximage
\end

Assuming that you made the luatex-plain format, such a file can be processed using:

luatex --fmt=luatex=plain luatex-swiglib-test.tex

The loaded Lua file luatex-swiglib-test.lua liike like this:

local gm = swiglib("gmwand.core")

gm.InitializeMagick(".")

local magick_wand = gm.NewMagickWand()
local drawing_wand = gm.NewDrawingWand()

gm.MagickSetSize(magick_wand,800,600)
gm.MagickReadImage(magick_wand,"xc:red")
gm.DrawPushGraphicContext(drawing_wand)
gm.DrawSetFillColor(drawing_wand,gm.NewPixelWand())
gm.DrawPopGraphicContext(drawing_wand)
gm.MagickDrawImage(magick_wand,drawing_wand)
gm.MagickWriteImages(magick_wand,"./luatex-swiglib-test.jpg",1)

gm.DestroyDrawingWand(drawing_wand)
gm.DestroyMagickWand(magick_wand)

Instead of loading a library with the swiglib function, you can also use require:

local gm = require("swiglib.gmwand.core")

Watch the explicit swiglib reference. Both methods are equivalent.

### 4 The libraries

Most libraries are small but some can be rather large and have additional files. This is why we keep them separated. On my system they are collected in the platform binary tree:

```
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/gmwand
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/....
```

One can modulate on this:

```
...tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql/core.dll
...tex/texmf-mswin/bin/lib/luajittex.lua/swiglib/mysql/core.dll
...tex/texmf-mswin/bin/lib/luatex/context/lua/swiglib/mysql/core.dll
```

are all valid. When versions are used you can provide an additional argument to the swiglib loader:

```
tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql/5.6/core.dll
```

This works with:

```
local mysql = swiglib("mysql.core","5.6")
```

as well as:

```
local mysql = swiglib("mysql.core")
```

It is hard to predict how operating systems look up libraries and especially nested loads, but as long as the root of the swiglib path is known to the file search routine. We’ve kept the main conditions for success simple: the core library is called core.dll or core.so. Each library has an (automatically called) initialize function named luaopen_core. There is no reason why (sym)links from the swiglib path to someplace else shouldn’t work.
In \texttt{texmfcnf.lua} you will find an entry like:

\begin{verbatim}
CLUAINPUTS = ".;\$SELFAUTOLOC/lib/{$engine/context,$engine}/lua//"
\end{verbatim}

Which in practice boils down to a search for \texttt{luatex} or \texttt{luajittex} specific libraries. When both binaries are compatible and there are no \texttt{luajittex} binaries, the regular \texttt{luatex} libraries will be used.

The \texttt{swiglib} loader function mentioned in previous sections load libraries in a special way: it changes dir to the specific path and then loads the library in the usual way. After that it returns to the path where it started out. After this, when the library needs additional libraries (and for instance graphicmagick needs a lot of them) it will first look on its own path (which is remembered).

The MkIV lookups are somewhat more robust in the sense that they first check for matches on engine specific paths. This comes in handy when the search patterns are too generic and one can match on for instance \texttt{luajittex} while \texttt{luatex} is used.

## 5 Compiling

Normally you will take the binaries from the ConTExt garden but if you ever want to compile yourself, it’s not that hard to do. For \texttt{linux} you need to install the compilers:

\begin{verbatim}
apt-get install gcc
apt-get install g++
\end{verbatim}

Then you need to make sure you have a copy of the LuaT\TeX{} sources (you need to use your own paths):

\begin{verbatim}
cd /data
svn checkout https://foundry.supelec.fr/svn/luatex/trunk luatex-trunk
\end{verbatim}

or update with:

\begin{verbatim}
cd /data
svn update luatex-trunk
\end{verbatim}

and then export with:

\begin{verbatim}
cd /data
svn export --force /data/luatex-trunk /data/luatex-trunk-export
\end{verbatim}

We go to the export directory and compile LuaT\TeX{}:

\begin{verbatim}
cd /data/luatex-trunk-export
./build.sh --jit
\end{verbatim}

The binaries are already stripped (i.e. symbols get removed) which makes them much smaller.
The native windows binaries are kept very up-to-date but you can cross compile your own if needed. You need to make sure that the cross compiler is installed.

```
apt-get install gcc-mingw-w64-x86-64
apt-get install g++-mingw-w64-x86-64
apt-get install binutils-mingw-w64
```

Given that you have exported the sources you can now run:

```
./build.sh --jit --mingw64
```

Of course we assume a recent linux installation here but on Windows you can the ‘linux subsystem for Windows’ too. The files can be found in a dedicated build directory:

```
cp data/luatex/luatex-trunk-export/build-windows64/texk/web2c/luatex.exe 
/data/context/tex/tex-context/tex/texmf-linux-64/bin

cp data/luatex/luatex-trunk-export/build-windows64/texk/web2c/luajittex.exe 
/data/context/tex/tex-context/tex/texmf-linux-64/bin
```

You need to wipe out old traces of binaries, because these can confuse the mtxrun stub that checks for them, so we do:

```
rm /data/context/tex/tex-context/tex/texmf-win64/bin/luajittex.dll
rm /data/context/tex/tex-context/tex/texmf-win64/bin/luatex.dll
```

The libraries are compiled in a similar way. This time we get the sources from another repository:

```
cd /data
svn checkout https://foundry.supelec.fr/svn/swiglib/trunk swiglib-trunk
```

or update with:

```
cd /data
svn update swiglib-trunk
```

and then export with:

```
cd /data
svn export --force /data/swiglib-trunk /data/swiglib-trunk-export
```

This time you need to be quite explicit with respect to the libraries you want to compile:

```
cd /data/swiglib-trunk-export
```
You can save yourself some work with:

```
mtxrun --script --svnroot=/data/swiglib-trunk-export --make
```

which will create a shell script `swiglib-make.sh` with commands that make all available libraries. After running that script you can update your tree with:

```
mtxrun --script --svnroot=/data/swiglib-trunk-export --update
```

For Windows a similar route is followed but first you need to make sure that your binaries are able to deal with shared libraries:

```
./build-shared.sh --jit --mingw64 --shared
```

Compiling is done as with linux but you need to provide the `--mingw64` flag. Copying is done with:

```
cp /data/luatex/luatex-trunk-export/build-windows64-shared\
   /texk/web2c/.libs/luatex.exe /data/context/tex/texmf-win64/bin

cp /data/luatex/luatex-trunk-export/build-windows64-shared\
   /texk/web2c/.libs/luajittex.exe /data/context/tex/texmf-win64/bin

cp /data/luatex/luatex-trunk-export/build-windows64-shared\n   /libs/luajit/.libs/texluajit*dll /data/context/tex/texmf-win64/bin
```

If you’re recompiling `--make` can save you some time. If your machine can handle it `--parallel` can speed up the process.

### 6 Colofon

**author**  
Hans Hagen, PRAGMA ADE, Hasselt NL

**version**  
August 31, 2016

**website**  

**comment**  
The swiglib infrastructure is implemented by Luigi Scarso