luatools
mtxrun
context

Remark

This manual is work in progress. Feel free to submit additions or corrections.

Introduction

Right from the start ConTEXt came with programs that managed the process of TEX-ing. Although you can perfectly well run TEX directly, it is a fact that often multiple runs are needed as well as that registers need to be sorted. Therefore managing a job makes sense.

First we had TEXexec and TEXutil, and both were written in Modula, and as this language was not supported on all platforms the programs were rewritten in Perl. Following that a few more tools were shipped with ConTEXt.

When we moved on to Ruby all the Perl scripts were rewritten and when ConTEXt MkIV showed up, Lua replaced Ruby. As we use LuaTEX this means that currently the tools and the main program share the same language. For MkII scripts like TEXexec will stay around but the idea is that there will be Lua alternatives for them as well.

Because we shipped many scripts, and because the de facto standard TEX directory structure expects scripts to be in certain locations we not only ship tools but also some more generic scripts that locate and run these tools.

The location

Normally you don't need to know so many details about where the scripts are located but here they are:

```
<texroot>/scripts/context/perl
<texroot>/scripts/context/ruby
<texroot>/scripts/context/lua
<texroot>/scripts/context/stubs
```

This hierarchy was actually introduced because ConTEXt was shipped with a bunch of tools. As mentioned, we nowadays focus on Lua but we keep a few of the older scripts around in the Perl and Ruby paths.

The traditional finder

When you run scripts multiple times, and in the case of ConTEXt they are even run inside other scripts, you want to minimize the startup time. Unfortunately the traditional way to locate a script, using `kpsewhich`, is not that fast, especially in a setup with many large trees Also, because not all tasks can be done with the traditional scripts (take format generation) we provided a runner that could deal with this: `texmfstart`. As this script was also used in more complex workflows, it had several tasks:

- locate scripts in the distribution and run them using the right interpreter
- do this selectively, for instance identify the need for a run using checksums for potentially changed files (handy for image conversion)
- pass information to child processes so that lookups are avoided
- choose a distribution among several installed versions (set the root of the TEX tree)
- change the working directory before running the script
- resolve paths and names on demand and launch programs with arguments where names are expanded controlled by prefixes (handy for TEX-unware programs)
- locate and open documentation, mostly as part the help systems in editors, but also handy for seeing what configuration file is used
- act as a kpsewhich server cq. client (only used in special cases, and using its own database)

Of course there were the usual more obscure and undocumented features as well. The idea was to use this runner as follows:

```
texmfstart texexec <further arguments>
texmfstart --tree <rootoftree> texexec <further arguments>
```

These are just two ways of calling this program. As `texmfstart` can initialize the environment as well, it is basically the only script that has to be present in the binary path. This is quite comfortable as this avoids conflicts in names between the called scripts and other installed programs.

Of course calls like above can be wrapped in a shell script or batch file without penalty as long as `texmfstart` itself is not wrapped in a caller script that applies other inefficient lookups. If you use the ConTEXt minimals you can be sure that the most efficient method is chosen, but we've seen quite inefficient call chains elsewhere.

In the ConTEXt minimals this script has been replaced by the one we will discuss in the next section: `mtxrun` but a stub is still provided.

The current finder

In MkIV we went a step further and completely abandoned the traditional lookup methods and do everything in Lua. As we want a clear separation between functionality we have two main controlling scripts: `mtxrun` and `luatools`. The last name may look somewhat confusing but the name is just one on in a series.[1]

In MkIV the `luatools` program is mainly used to generate the (cached) file database. In that respect it's rather dumb in that it does not use the database, but clever at the same time because it can make one based on the little information available when it runs. You can consider it to replace `mktexlsr` and `kpsewhich` and friends. It is also used to generate format files either or not using Lua stubs but in practice this is only useful for ConTEXt.

[1] We have `ctxtools`, `exatools`, `mpstools`, `mtxtools`, `pdftools`, `rlxtools`, `runtools`, `textools`, `tmftools` and `xmltools`. Most if their funtionality is already reimplemented.

For ConTEXt users, the main invocation of this tool is when the TEX tree is updated. For instance, after adding a font to the tree or after updating ConTEXt, you need to run:

```
luatools --generate
```

After that all tools will know where to find stuff and how to behave well within the tree. This is because they share the same code, mostly because they are started using another script: `mtxrun`. For instance, you process a file with:

```
mtxrun --script context <somefile>
```

Because this happens often, there's also a shortcut:

```
context <somefile>
```

But this does use `mtxrun` as well. The help information of `mtxrun` is rather minimalistic and if you have no clue what an option does, you probably never needed it anyway. Here we discuss a few options. We already saw that we can explicitly ask for a script:

```
mtxrun --script context <somefile>
```

but

```
mtxrun context <somefile>
```

also works. However, by using `--script` you limit te lookup to the valid ConTEXt MkIV scripts. In the TEX tree these have names prefixed by `mtx-` and a lookup look for a plural as well. So, the next two lookups are equivalent:

```
mtxrun --script font
mtxrun --script fonts
```

Both will run `mtx-fonts.lua`. Normally this is all you need in order to run a script. However, there are a few more options:

```
MTXrun | TDS Runner Tool 1.24
MTXrun |
MTXrun | --script             run an mtx script (lua prefered method) (--noquotes), no script gives list
MTXrun | --execute            run a script or program (texmfstart method) (--noquotes)
MTXrun | --resolve            resolve prefixed arguments
MTXrun | --ctxlua             run internally (using preloaded libs)
MTXrun | --locate             locate given filename
MTXrun |
MTXrun | --autotree           use texmf tree cf. env 'texmfstart_tree' or 'texmfstarttree'
MTXrun | --tree=pathtotree    use given texmf tree (default file: 'setuptex.tmf')
MTXrun | --environment=name   use given (tmf) environment file
MTXrun | --path=runpath       go to given path before execution
MTXrun | --ifchanged=filename only execute when given file has changed (md checksum)
MTXrun | --iftouched=old,new  only execute when given file has changed (time stamp)
MTXrun |
```

```
MTXrun | --make               create stubs for (context related) scripts
MTXrun | --remove             remove stubs (context related) scripts
MTXrun | --stubpath=binpath   paths where stubs wil be written
MTXrun | --windows            create windows (mswin) stubs
MTXrun | --unix               create unix (linux) stubs
MTXrun |
MTXrun | --verbose            give a bit more info
MTXrun | --trackers=list      enable given trackers
MTXrun | --engine=str         target engine
MTXrun | --progname=str       format or backend
MTXrun |
MTXrun | --edit               launch editor with found file
MTXrun | --launch (--all)     launch files like manuals, assumes os support
MTXrun |
MTXrun | --internal           run script using built in libraries (same as --ctxlua)
MTXrun | --timedrun           run a script an time its run
MTXrun |
MTXrun | --usekpse            use kpse as fallback (when no mkiv and cache installed, often slower)
MTXrun | --forcekpse          force using kpse (handy when no mkiv and cache installed but less functionality)
MTXrun |
MTXrun | --prefixes           show supported prefixes
MTXrun |
MTXrun | more information about ConTeXt and the tools that come with it can be found at:
MTXrun |
MTXrun | maillist : ntg-context@ntg.nl / http://www.ntg.nl/mailman/listinfo/ntg-context
MTXrun | webpage  : http://www.pragma-ade.nl / http://tex.aanhet.net
MTXrun | wiki     : http://contextgarden.net
```

Updating

There are two ways to update ConTEXt MkIV. When you manage your trees yourself or when you use for instance TEXLive, you act as follows:

- download the file cont-tmf.zip from `www.pragma-ade.com` or elsewhere
- unzip this file in a subtree, for instance `tex/texmf-local`
- run `luatools --generate`
- run `mtxrun --script font --reload`
- run `mtxrun --script context --make`

Or shorter:

- run `luatools --generate`
- run `mtxrun font --reload`
- run `context --make`

Normally these commands are not even needed, but they are a nice test if your tree is still okay. To some extend `context` is clever enough to decide if the databases need to be regenerated and/or a format needs to be remade and/or if a new font database is needed.

Now, if you also want to run MkII, you need to add:

- run `mktexlsr`
- run `texexec --make`

The question is, how to act when `luatools` and `mtxrun` have been updated themselves? In that case, after unzipping the archive, you need to do the following:

- run `luatools --selfupdate`
- run `mtxrun --selfupdate`

For quite a while we shipped so called ConTEXt minimals. These zip files contained only the resources and programs that made sense for running ConTEXt. Nowadays the minimals are installed and synchronized via internet.[2] You can just run the installer again and no additional commands are needed. In the console you will see several calls to `mtxrun` and `luatools` fly by.

The tools

We only mention the tools here. The most important ones are `context` and `fonts`. You can ask for a list of installed scripts with:

```
mtxrun --script
```

On my machine this gives:

```
MTXrun | TDS Runner Tool 1.24
MTXrun |
MTXrun | no script name given, known scripts:
MTXrun |
MTXrun | babel         1.20  Babel Input To UTF Conversion
MTXrun | cache         0.10  ConTeXt & MetaTeX Cache Management
MTXrun | chars         0.10  MkII Character Table Generators
MTXrun | check         0.10  Basic ConTeXt Syntax Checking
MTXrun | context       0.51  ConTeXt Process Management
MTXrun | convert       0.10  ConTeXT Graphic Conversion Helpers
MTXrun | fonts         0.21  ConTeXt Font Database Management
MTXrun | grep          0.10  Simple Grepper
MTXrun | idris         0.10  Special Hacks For Idris
MTXrun | interface     0.11  ConTeXt Interface Related Goodies
MTXrun | metatex       0.10  MetaTeX Process Management
MTXrun | modules       1.00  ConTeXt Module Documentation Generators
MTXrun | mptopdf       0.51  MetaPost to PDF Converter
MTXrun | package       0.10  Distribution Related Goodies
MTXrun | patterns      0.20  ConTeXt Pattern File Management
MTXrun | profile       1.00  ConTeXt MkIV LuaTeX Profiler
```

[2] This project was triggered by Mojca Miklavec who is also charge of this bit of the ConTEXt infrastructure. More information can be found at `contextgarden.net`.

```
MTXrun | server       0.10  Simple Webserver For Helpers
MTXrun | stubs        0.10  ConTeXt Stub Management
MTXrun | tds          0.10  TeX Directory Structure Tools
MTXrun | texutil      0.10  MkII Utility File Conversion
MTXrun | texworks     1.00  TeXworks Startup Script
MTXrun | timing       0.10  ConTeXt Timing Tools
MTXrun | tools        1.00  Some File Related Goodies
MTXrun | tracing      1.00  MkIV LuaTeX Profiler
MTXrun | unzip        0.10  Simple Unzipper
MTXrun | update       0.21  ConTeXt Minimals Updater
MTXrun | watch        1.00  ConTeXt Request Watchdog
MTXrun | web          0.10  Some (Private) Webservice Goodies
```

The most important scripts are `mtx-fonts` and `mtx-context`. By default fonts are looked up by filename (the `file:` prefix before font names in ConTEXt is default). But you can also lookup fonts by name (`name:`) or by specification (`spec:`). If you want to use these two methods, you need to generate a font database as mentioned in the previous section. You can also use the font tool to get information about the fonts installed on your system.

Running ConTEXt

The `context` tool is what you will use most as it manages your run.

```
MTXrun | TDS Runner Tool 1.24 | ConTeXt Process Management 0.51
MTXrun |
MTXrun | --run                process (one or more) files (default action)
MTXrun | --make               create context formats
MTXrun |
MTXrun | --ctx=name           use ctx file (process management specification)
MTXrun | --interface          use specified user interface (default: en)
MTXrun |
MTXrun | --autopdf            close pdf file in viewer and start pdf viewer afterwards
MTXrun | --purge(all)         purge files either or not after a run (--pattern=...)
MTXrun |
MTXrun | --usemodule=list     load the given module or style, normally part o fthe distribution
MTXrun | --environment=list   load the given environment file first (document styles)
MTXrun | --mode=list          enable given the modes (conditional processing in styles)
MTXrun | --path=list          also consult the given paths when files are looked for
MTXrun | --arguments=list     set variables that can be consulted during a run (key/value pairs)
MTXrun | --randomseed=number  set the randomseed
MTXrun | --result=name        rename the resulting output to the given name
MTXrun | --trackers=list      show/set tracker variables
MTXrun | --directives=list    show/set directive variables
MTXrun |
MTXrun | --forcexml           force xml stub (optional flag: --mkii)
MTXrun | --forcecld           force cld (context lua document) stub
MTXrun |
MTXrun | --arrange            run extra imposition pass, given that the style sets up imposition
```

```
MTXrun | --noarrange          ignore imposition specifications in the style
MTXrun |
MTXrun | --once               only run once (no multipass data file is produced)
MTXrun | --batchmode          run without stopping and don't show messages on the console
MTXrun | --nonstopmode        run without stopping
MTXrun |
MTXrun | --generate           generate file database etc. (as luatools does)
MTXrun | --paranoid           don't descend to .. and ../..
MTXrun | --version            report installed context version
MTXrun |
MTXrun | --expert             expert options
MTXrun |
MTXrun | more information about ConTeXt and the tools that come with it can be found at:
MTXrun |
MTXrun | maillist : ntg-context@ntg.nl / http://www.ntg.nl/mailman/listinfo/ntg-context
MTXrun | webpage  : http://www.pragma-ade.nl / http://tex.aanhet.net
MTXrun | wiki     : http://contextgarden.net
```

There are few exert options too:

```
MTXrun | TDS Runner Tool 1.24 | ConTeXt Process Management 0.51
MTXrun |
MTXrun | expert options:
MTXrun |
MTXrun | --touch              update context version number (remake needed afterwards, also provide --expert)
MTXrun | --nostats            omit runtime statistics at the end of the run
MTXrun | --update             update context from website (not to be confused with contextgarden)
MTXrun | --profile            profile job (use: mtxrun --script profile --analyse)
MTXrun | --timing             generate timing and statistics overview
MTXrun | --tracefiles         show some extra info when locating files (at the tex end)
MTXrun |
MTXrun | --extra=name         process extra (mtx-context-<name> in distribution)
MTXrun | --extras             show extras
MTXrun |
MTXrun | private options:
MTXrun |
MTXrun | --dumphash           dump hash table afterwards
MTXrun | --dumpdelta          dump hash table afterwards (only new entries)
MTXrun |
MTXrun | special options:
MTXrun |
MTXrun | --pdftex             process file with texexec using pdftex
MTXrun | --xetex              process file with texexec using xetex
MTXrun |
MTXrun | --pipe               don't check for file and enter scroll mode (--dummyfile=whatever.tmp)
MTXrun |
MTXrun | more information about ConTeXt and the tools that come with it can be found at:
MTXrun |
MTXrun | maillist : ntg-context@ntg.nl / http://www.ntg.nl/mailman/listinfo/ntg-context
```

```
MTXrun | webpage  : http://www.pragma-ade.nl / http://tex.aanhet.net
MTXrun | wiki     : http://contextgarden.net
```

You might as well forget about these unless you are one of the ConTEXt developers.

Prefixes

A handy feature of `mtxrun` (and as most features an inheritance of `texmfstart`) is that it will resolve prefixed arguments. This can be of help when you run programs that are unaware of the TEX tree but nevertheless need to locate files in it.

```
MTXrun | TDS Runner Tool 1.24
MTXrun |
MTXrun | auto: env: environment: file: filename: full: kpse: loc: locate: machine: nodename: path: pathname: rel: relative:
release: sysname: version:
```

An example is:

```
mtxrun --execute xsltproc file:whatever.xsl file:whatever.xml
```

The call to `xsltproc` will get two arguments, being the complete path to the files (given that it can be resolved). This permits you to organize the files in a similar was as TEX files.

Stubs

As the tools are written in the Lua language we need a Lua interpreter and or course we use LuaTEX itself. On Unix we can copy `luatools` and `mtxrun` to files in the binary path with the same name but without suffix. Starting them in another way is a waste of time, especially when `kpsewhich` is used to find then, something which is useless in MkIV anyway. Just use these scripts directly as they are self contained.

For `context` and other scripts that we want convenient access to, stubs are needed, like:

```
#!/bin/sh
mtxrun --script context "$@"
```

This is also quite efficient as the `context` script `mtx-context` is loaded in `mtxrun` and uses the same database.

On Windows you can copy the scripts as-is and associate the suffix with LuaTEX (or more precisely: `texlua`) but then all Lua script will be run that way which is not what you might want.

In TEXLive stubs for starting scripts were introduced by Fabrice Popineau. Such a stub would start for instance `texmfstart`, that is: it located the script (Perl or Ruby) in the TEX tree and launched it with the right interpreter. Later we shipped pseudo binaries of `texmfstart`: a Ruby interpreter plus scripts wrapped into a self contained binary.

For MkIV we don't need such methods and started with simple batch files, similar to the Unix startup scripts. However, these have the disadvantage that they cannot be used in other batch files without using the `start` command. In TEXLive this is taken care of by a small binary written bij T.M. Trzeciak so on TEXLive 2009 we saw a call chain from `exe` to `cmd` to `lua` which is somewhat messy.

This is why we now use an adapted and stripped down version of that program that is tuned for `mtxrun` and `luatools`. So, we moved from the original `cmd` based approach to an `exe` one.

`mtxrun.dll`
`mtxrun.exe`

You can copy `mtxrun.exe` to for instance `context.exe` and it will still use `mtxrun` for locating the right script. It also takes care of mapping `texmfstart` to `mtxrun`. So we've removed the intermediate `cmd` step, can not run the script as any program, and most of all, we're as efficient as can be.

Of course this program is only meaningful for the ConTEXt approach to tools.

It may all sound more complex than it is but once it works users will not notice those details. Als, in practice not that much has changed in running the tools between MkII and MkIV as we've seen no reason to change the methods.

Hans Hagen
Hasselt NL
2009 — . . .