luatools mtxrun context

Contents

1 Remark

This manual is work in progress. Feel free to submit additions or corrections. Before you start reading, it is good to know that in order to get starting with ConTEXt, the easiest way to do that is to download the standalone distribution from `contextgarden.net`. After that you only need to make sure that `luatex` is in your path. The main command you use is then `context` and normally it does all the magic it needs itself.

2 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.

3 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.Now, if you're only using ConTEXt MkIV, and this is highly recommended, you can forget about all but the Lua scripts.

4 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.

5 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 nowadays seldom used. It's just a drop in for `kpsewhich` plus a bit more. 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. It can also be used to generate format files either or not using Lua stubs but in practice this is not needed at all.

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:

```
mtxrun --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 `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`. Actually, this is one of the scripts that you might need when your font database is somehow outdated and not updated automatically:

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

```
mtxrun --script fonts --reload --force
```

Normally `mtxrun` is all you need in order to run a script. However, there are a few more options:

```
mtxrun          | ConTeXt TDS Runner Tool 1.31
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          | --internal            run script using built in libraries (same as --ctxlua)
mtxrun          | --locate              locate given filename in database (default) or system (--first --all --detail)
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          | --makestubs           create stubs for (context related) scripts
mtxrun          | --removestubs         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          | --progname=str        format or backend
mtxrun          |
mtxrun          | --edit                launch editor with found file
mtxrun          | --launch              launch files like manuals, assumes os support (--all)
mtxrun          |
mtxrun          | --timedrun            run a script and time its run
mtxrun          | --autogenerate        regenerate databases if needed (handy when used to run context in an editor)
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          | --generate            generate file database
mtxrun          |
mtxrun          | --variables           show configuration variables
mtxrun          | --configurations      show configuration order
mtxrun          |
mtxrun          | --directives          show (known) directives
mtxrun          | --trackers            show (known) trackers
mtxrun          | --experiments         show (known) experiments
mtxrun         |
mtxrun         | --expand-braces       expand complex variable
mtxrun         | --expand-path         expand variable (resolve paths)
mtxrun         | --expand-var          expand variable (resolve references)
mtxrun         | --show-path           show path expansion of ...
mtxrun         | --var-value           report value of variable
mtxrun         | --find-file           report file location
mtxrun         | --find-path           report path of file
mtxrun         |
mtxrun         | --pattern=string      filter variables
mtxrun         |
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
```

Don't worry,you only need those obscure features when you integrate ConTEXt in for instance a web service or when you run large projects where runs and paths take special care.

6 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 `mtxrun --generate`
- run `mtxrun --script font --reload`
- run `mtxrun --script context --make`

Or shorter:

- run `mtxrun --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.

7 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:

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.

8 Running CONTEXT

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

```
mtx-context     | ConTeXt Process Management 0.60
mtx-context     |
mtx-context     | basic options:
mtx-context     |
mtx-context     | --run                 process (one or more) files (default action)
mtx-context     | --make                create context formats
mtx-context     |
mtx-context     | --ctx=name            use ctx file (process management specification)
mtx-context     | --interface           use specified user interface (default: en)
mtx-context     |
mtx-context     | --autopdf             close pdf file in viewer and start pdf viewer afterwards
mtx-context     | --purge               purge files either or not after a run (--pattern=...)
mtx-context     | --purgeall            purge all files either or not after a run (--pattern=...)
mtx-context     |
mtx-context     | --usemodule=list      load the given module or style, normally part of the distribution
mtx-context     | --environment=list    load the given environment file first (document styles)
mtx-context     | --mode=list           enable given the modes (conditional processing in styles)
```

[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`.

```
mtx-context     | --path=list           also consult the given paths when files are looked for
mtx-context     | --arguments=list      set variables that can be consulted during a run (key/value pairs)
mtx-context     | --randomseed=number   set the randomseed
mtx-context     | --result=name         rename the resulting output to the given name
mtx-context     | --trackers=list       set tracker variables (show list with --showtrackers)
mtx-context     | --directives=list     set directive variables (show list with --showdirectives)
mtx-context     | --silent=list         disable logcatgories (show list with --showlogcategories)
mtx-context     | --noconsole           disable logging to the console (logfile only)
mtx-context     | --purgeresult         purge result file before run
mtx-context     |
mtx-context     | --forcexml            force xml stub
mtx-context     | --forcecld            force cld (context lua document) stub
mtx-context     | --forcelua            force lua stub (like texlua)
mtx-context     | --forcemp             force mp stub
mtx-context     |
mtx-context     | --arrange             run extra imposition pass, given that the style sets up imposition
mtx-context     | --noarrange           ignore imposition specifications in the style
mtx-context     |
mtx-context     | --jit                 use luajittex with jit turned off (only use the faster virtual machine)
mtx-context     | --jiton               use luajittex with jit turned on (in most cases not faster, even slower)
mtx-context     |
mtx-context     | --once                only run once (no multipass data file is produced)
mtx-context     | --batchmode           run without stopping and do not show messages on the console
mtx-context     | --nonstopmode         run without stopping
mtx-context     | --synctex             run with synctex enabled (optional value: zipped, unzipped, 1, -1)
mtx-context     |
mtx-context     | --generate            generate file database etc. (as luatools does)
mtx-context     | --paranoid            do not descend to .. and ../..
mtx-context     | --version             report installed context version
mtx-context     |
mtx-context     | --global              assume given file present elsewhere
mtx-context     | --nofile              use dummy file as jobname
mtx-context     |
mtx-context     |
mtx-context     | More information about ConTeXt and the tools that come with it can be found at:
mtx-context     |
mtx-context     | maillist : ntg-context@ntg.nl / http://www.ntg.nl/mailman/listinfo/ntg-context
mtx-context     | webpage  : http://www.pragma-ade.nl / http://tex.aanhet.net
mtx-context     | wiki     : http://contextgarden.net
```

There are few exert options too:

```
mtx-context     | ConTeXt Process Management 0.60
mtx-context     |
mtx-context     | expert options:
mtx-context     |
mtx-context     | --touch               update context version number (remake needed afterwards, also provide --expert)
mtx-context     | --nostatistics        omit runtime statistics at the end of the run
mtx-context     | --update              update context from website (not to be confused with contextgarden)
```

```
mtx-context     | --profile             profile job (use: mtxrun --script profile --analyze)
mtx-context     | --timing              generate timing and statistics overview
mtx-context     |
mtx-context     | --extra=name          process extra (mtx-context-... in distribution)
mtx-context     | --extras              show extras
mtx-context     |
mtx-context     | special options:
mtx-context     |
mtx-context     | --pdftex              process file with texexec using pdftex
mtx-context     | --xetex               process file with texexec using xetex
mtx-context     | --mkii                process file with texexec
mtx-context     |
mtx-context     | --pipe                do not check for file and enter scroll mode (--dummyfile=whatever.tmp)
mtx-context     |
mtx-context     |
mtx-context     | More information about ConTeXt and the tools that come with it can be found at:
mtx-context     |
mtx-context     | maillist : ntg-context@ntg.nl / http://www.ntg.nl/mailman/listinfo/ntg-context
mtx-context     | webpage  : http://www.pragma-ade.nl / http://tex.aanhet.net
mtx-context     | wiki     : http://contextgarden.net
```

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

9 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          | ConTeXt TDS Runner Tool 1.31
mtxrun          |
mtxrun          |
mtxrun          | auto: env: environment: file: filename: full: home: kpse: loc: locate: machine: nodename: path: pathname:
rel: relative: release: selfautodir: selfautoloc: selfautoparent: sysname: toppath: 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.

10 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, luatools and context. 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.

Colofon

author	Hans Hagen, PRAGMA ADE, Hasselt NL
version	March 22, 2013
website	www.pragma-ade.nl – www.contextgarden.net
copyright	🅭 🅯 🄎 🄏