
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>
```

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

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:

```
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 1.32
mtxrun          |
mtxrun          | --script              run an mtx script (lua prefered method) (--noquotes), no script gives list
mtxrun          | --evaluate            run code passed on the commandline (between quotes)
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          | --tree=pathtotree     use given texmf tree (default file: setuptex.tmf)
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          | --systeminfo=str       show current operating system, processor, etc
mtxrun          |
mtxrun          | --edit                 launch editor with found file
mtxrun          | --launch               launch files like manuals, assumes os support (--all,--list)
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          | --resolve-path         expand variable (completely resolve paths)
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 in-

[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`.

staller 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:

```
mtxrun          | ConTeXt TDS Runner Tool 1.32
mtxrun          |
mtxrun          | no script name given, known scripts:
mtxrun          |
mtxrun          | babel         1.20  Babel Input To UTF Conversion
mtxrun          | base          1.35  ConTeXt TDS Management Tool (aka luatools)
mtxrun          | bibtex              bibtex helpers
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          | colors        0.10  ConTeXt Color Management
mtxrun          | convert       0.10  ConTeXT Graphic Conversion Helpers
mtxrun          | distribution  0.10  ConTeXt Distribution Helpers
mtxrun          | dvi           0.10  ConTeXt DVI Helpers
mtxrun          | epub          1.10  ConTeXt EPUB Helpers
mtxrun          | evohome       1.00  Evohome Fetcher
mtxrun          | example       0.10  ConTeXt Example Helpers
mtxrun          | fcd           1.00  Fast Directory Change
mtxrun          | flac          0.10  ConTeXt Flac 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     0.13  ConTeXt Interface Related Goodies
mtxrun          | listen        1.00  ConTeXt Request Watchdog
mtxrun          | metapost      0.10  MetaPost to PDF processor
mtxrun          | metatex       0.10  MetaTeX Process Management
mtxrun          | modules       1.00  ConTeXt Module Documentation Generators
mtxrun          | package       0.10  Distribution Related Goodies
mtxrun          | patterns      0.20  ConTeXt Pattern File Management
mtxrun          | pdf           0.10  ConTeXt PDF Helpers
mtxrun          | plain         1.00  Plain TeX Runner
mtxrun          | profile       1.00  ConTeXt MkIV LuaTeX Profiler
mtxrun          | queue         1.00  Sequential runner
mtxrun          | rsync         0.10  Rsync Helpers
mtxrun          | scite         1.00  Scite Helper Script
```

```
mtxrun          | server        0.10  Simple Webserver For Helpers
mtxrun          | stubs         0.10  ConTeXt Stub Management
mtxrun          | swiglib       1.00  ConTeXt Swiglib Updater
mtxrun          | synctex       1.00  ConTeXt SyncTeX Checker
mtxrun          | tds           0.10  TeX Directory Structure Tools
mtxrun          | testsuite     1.00  Experiments with the testsuite
mtxrun          | texworks      1.00  TeXworks Startup Script
mtxrun          | timing        0.10  ConTeXt Timing Tools
mtxrun          | tools         1.01  Some File Related Goodies
mtxrun          | tracing       1.00  MkIV LuaTeX Profiler
mtxrun          | unicode       1.02  Checker for char-def.lua
mtxrun          | unzip         0.10  Simple Unzipper
mtxrun          | update        1.03  ConTeXt Minimals Updater
mtxrun          | update        1.02  ConTeXt Minimals Updater
mtxrun          | watch         1.00  ConTeXt Request Watchdog
mtxrun          | web           0.10  Some (Private) Webservice Goodies
mtxrun          | youless       1.10  YouLess Fetcher
```

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 1.02
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     | --noctx               ignore ctx directives and flags
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)
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     | --strip               strip Lua code (only meant for production where no errors are expected)
mtx-context     | --errors=list         show errors at the end of a run, quit when in list (also when ----silent)
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     | --runs                process at most this many times
mtx-context     | --forcedruns          process this many times (permits for optimization trial runs)
mtx-context     |
mtx-context     | --batchmode           run without stopping and do not show messages on the console
mtx-context     | --nonstopmode         run without stopping
mtx-context     | --nosynctex           never initializes synctex (for production runs)
mtx-context     | --synctex             run with synctex enabled (better use \setupsynctex[state=start]
mtx-context     | --nodates             omit runtime dates in pdf file (optional value: a number (this 1970 offset
time) or string "YYYY-MM-DD HH:MM")
mtx-context     | --nocompression       forcefully turns off compression in the backend
mtx-context     | --trailerid           alternative trailer id (or constant one)
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 1.02
mtx-context     |
mtx-context     | expert options:
mtx-context     |
mtx-context     | --touch              update context version number (also provide --expert, optionally provide
--basepath)
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     | --keeptuc            keep previous tuc files (jobname-tuc-[run].tmp)
mtx-context     | --keeplog            keep previous log files (jobname-log-[run].tmp)
mtx-context     | --lmtx               force lmtx mode (when available)
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     | --sandbox            process file in a limited environment
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 1.32
mtxrun          |
mtxrun          |
mtxrun          | auto: env: environment: file: filename: full: home: jobpath: 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.

11 A detailed look at `mtxrun`

This section is derived from Taco Hoekwaters presentation and article for the 2018 ConTEXt meeting. You might want to read this is you want to benefit from even the most obscure features. There is a bit of repetition with the previous sections but so be it.

11.1 Common flags

Much of the code inside MkIV can alter its behaviour based on either 'trackers' (which add debugging information to the terminal and log output) or 'directives' or 'experiments' (for getting code to perform some alternate behaviour). Since this also affects the Lua code within `mtxrun` itself, it makes sense to list these options first.

Getting information on trackers, directives and experiments. Trackers enable more extensive status messages on the console or in ConTEXt additional visual clues. Directives change behaviour that are not part of the official interface and have no corresponding commands. Experiments are like directives but not official (yet).

`--trackers`
show (known) trackers

`--directives`
show (known) directives

`--experiments`
show (known) experiments

Enabling directives, trackers and experiments:

`--trackers=list`
enable given trackers

`--directives=list`
enable given directives

`--experiments=list`
enable given experiments

The next tree (hidden) options are converted into 'directives' entries, that are then enabled. These are just syntactic sugar for the relevant directive.

`--silent[=...]`
sets `logs.blocked={\%s}`

`--errors[=...]`
sets `logs.errors={\%s}`

`--noconsole`
sets `logs.target=file`

As you can see here, various directives (and even some trackers) have optional arguments, which can make specifying such directives on the command line a bit of a challenge. Explaining the details of all the directives is outside of the scope of this article, but you can look them up in the ConTEXt source by searching for `directives.register` and `trackers.register`.

In verbose mode, `mtxrun` itself gives more messages, and it also `resolvers.locating`, which is a tracker itself:

`--verbose`
give a bit more info

The `--timedlog` (hidden) option starts the `mtxrun` output with a timestamp line:

`--timedlog`
prepend output with a timestamp

11.2 Setup for finding files and configurations

The next block of options deals with the setup of `mtxrun` itself. You do not need to deal with these options unless you are messing with the ConTEXt distribution yourself instead of relying on a prepackaged solution, or you need to use kpathsea for some reason (typically in a MkII environment). In particular, `--progname` and `--tree` are often needed as well when using the `kpse` options.

`--configurations`
show configuration order, alias `--show-configurations`

`--resolve`
resolve prefixed arguments, see `--prefixes`, below

and:

`--usekpse`
use kpse as fallback (when no MkIV and cache installed, often slower)

`--forcekpse`
force using kpse (handy when no MkIV and cache installed but less functionality)

`--progname=str`
format or backend

`--tree=pathtotree`
use given texmf tree (default file: `setuptex.tmf`)

11.3 Options for finding files and reporting configurations

Once the configuration setup is done, it makes sense to have a bunch or options to use and/or query the configuration.

`--locate`
locate given filename in database (default) or system (uses the sub--options `--first`, `--all` and `--detail`)

`--autogenerate`
regenerate databases if needed (handy when used to run context in an editor)

`--generate`
generate file database

`--prefixes`
show supported prefixes for file searches

`--variables`
show configuration variables (uses the sub--option `--pattern`, and an alias is `--show-variables`)

`--expansions`
show configuration variable expansion (uses the sub--options `--pattern`, alias `--show-expansions`)

`--expand-braces`
expand complex variable

`--resolve-path`
expand variable (completely resolve paths)

`--expand-path`
expand variable (resolve paths)

`--expand-var`
expand variable (resolves references inside variables, alias `--expand-variable`)

`--show-path`
show path expansion of `...` (alias `--path-value`)

`--var-value`
report value of variable (alias `--show-value`)

`--find-file`
: report file location; it uses the sub–options `--all`, `--pattern`, and `--format`

`--find-path`
: report path of file

Hidden option:

`--format-path`
: report format path

11.4 Running code

Here we come to the core functionality of `mtxrun`: running scripts. First there are few options that trigger how the running takes place:

`--path=runpath`
: go to given path before execution

`--ifchanged=filename`
: only execute when given file has changed (this loads and saves an md5 checksum from `filename.md5`)

`--iftouched=old,new`
: only execute when given file has changed (time stamp)

`--timedrun`
: run a script or program and time its run (external)

Specifying both `--iftouched` and `--ifchanged` means both are tested, and when either one is false, nothing will happen. These options have to come before one of the next options:

`--script`
: run an mtx script (where Lua is the preferred method); it has the sub–options `--nofiledatabase`, `--autogenerate`, `--load`, and `--save`. The latter two are currently no-ops

`--execute`
: run a script or program externally (`texmfstart` method); it has sub–option `--noquotes`

`--internal`
: run a script using built-in libraries (alias is `--ctxlua`)

`--direct`
: run an external program; it has the sub–option `--noquotes`

Since scripts potentially have their own options, any options intended for `mtxrun` itself have to come *before* the option that specifies the script to run, and options for the script itself have to come *after* the option that gives the script name. This is especially true when using `--script`, so it is important to check the order of your options.

Of the four above options, `--script` is the most important one, since that is the one that finds and executes the Lua `mtxrun` scripts provided by the distribution. With `--nofiledatabase`, it will not attempt to resolve any file names (which means you need either a local script or a full path name). On the opposite side, when you also provide `--autogenerate`, it will not only attempt to resolve the file name, it will also regenerate the database if it cannot find the script on the first try. In a future version of ConTEXt, the `--load` and `--save` will allow you to save/restore the current command line in a file for reuse.

The shell return value of `mtxrun` indicates whether the script was found. When you specify something like `--script base`, `mtxrun` actually searches for `mtx-base.lua`, `mtx-bases.lua`, `mtx-t-base.lua`, `mtx-t-bases.lua`, and `base.lua`, in that order. The distribution–supplied scripts normally use `mtx-<name>.lua` as template. The template names with `mtx-t-` prefix is reserved for third–party scripts, and `<name>.lua` is just a last-ditch effort if nothing else works. Scripts are looked for in the local path, and in whatever directories the configuration variable `LUAINPUTS` points to.

The `--execute` options exists mostly for the non–Lua MkII scripts that still exist in the distribution. It will try to find a matching interpreter for non–Lua scripts, and it is aware of a number of distribution–supplied scripts so that if you specify `--execute texexec`, it knows that what you really want to execute is `ruby texexec.rb`. Support is present for Ruby (`.rb`, Lua (`.lua`), python (`.py`) and Perl (`.pl`) scripts (tested in that order). File resolving uses `TEXMFSCRIPTS` from the configuration. The shell return value of `mtxrun` indicates whether the script was found and executed.

The `--internal` option uses the file search method of `--execute`, but then assumes this is a Lua script and executes it internally like `--script`. This is useful if you have a Lua script in an odd location.

The last of the four options, `--direct`, directly executes an external program. You need to give the full path for binaries not in the current shell `PATH`, because no searching is done at all. The shell return value of `mtxrun` in this case is a boolean based on the return value of `os.exec()`.

It is also possible to execute bare Lua code directly:

`--evaluate`
: run code passed on the command-line (between quotes)

11.5 Options for maintenance of `mtxrun` itself

None of these are advertised. Normally developers should be the only ones needing them, but if you made a change to one of the distributed libraries (maybe because of a beta bug), you may need to run `--selfmerge` and `--selfupdate`.

`--selfclean`
: remove embedded libraries

`--selfmerge`
: update embedded libraries in `mtxrun.lua`

`--selfupdate`
: copy `mtxlua.lua` to the executable directory, renamed `mtxrun`

11.6 Creating stubs

Stubs are little shortcuts that live in some binaries directory. For example, the content of the Unix–style `context` shell command is:

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

Apart from the `context` command itself (which is provided by the distribution), use of stubs is discouraged. Still, the `mtxrun` options are there because sometimes existing workflows depend on executable tool names like `ctxtools`.

`--makestubs`
: create stubs for (context related) scripts

`--removestubs`
: remove stubs (context related) scripts

`--stubpath=binpath`
: paths where stubs will be written

`--windows`
: create windows (mswin) stubs (alias `--mswin`)

`--unix`
: create unix (linux) stubs (alias `--linux`)

11.7 Remaining options

The remaining options are hard to group into a subcategory. These are the advertised options:

`--systeminfo`
: show current operating system, processor, et cetera

`--edit`
: launch editor with found file; the editor is taken from the environment variable `MTXRUN_EDITOR`, or `TEXMFSTART_EDITOR`, or `EDITOR`, or as a last resort: `gvim`

`--launch`
: launch files like manuals, assumes os support (uses the sub–options `--all`, `--pattern` and `--list`)

While these are sort of hidden options:

`--ansi`
: colorize output to terminal using ansi escapes

`--associate`
: launch files like manuals, assumes os support. this function does not do any file searching, so you have to use either a local file or a full path name

`--exporthelp`
output the `mtxrun` xml help blob (useful for creating man and html help pages)

`--fmt`
shortcut for `--script base --fmt`

`--gethelp`
attempt to look up remote `context` command help (uses the sub–options `--command` and `--url`)

`--help`
print the `mtxrun` help screen

`--locale`
force setup of locale; unless you are certain you need this option, stay away from it, because it can interfere massively with ConTEXt's Lua code

`--make`
(re)create format files (aliases are `--ini` and `--compile`)

`--platform`
(alias is `--show-platform`)

`--run`
shortcut for `--script base --run`

`--version`
print `mtxrun` version

11.8 Known scripts

When you run `mtxrun --scripts`, it will output a list of 'known' scripts. The definition of 'known' is important here: the list comprises the scripts that are present in the same directory as `mtx-context.lua` that do not have an extra hyphen in the name (like `mtx-t-...` scripts would have). In a normal installation, this means it 'knows' almost all the scripts that are distributed with ConTEXt. Note: it skips over any files from the distribution that do have an extra hyphen, like the `mtx-server` support scripts.

Since this section is about `mtxrun`, I'll just present the list of the scripts that are 'known' in the current ConTEXt beta as output by `mtxrun` itself, and not get into detail about all of the script functionality (they all have `--help` options if you want to find out more). Where we still felt the need to explain something, there is an extra bit of text in italics.

`babel`
: Babel Input To UTF Conversion

`base`
: ConTeXt TDS Management Tool (aka luatools)

`bibtex`
: bibtex helpers (obsolete)

`cache`
: ConTeXt & MetaTeX Cache Management

`chars`
: MkII Character Table Generators

`check`
: Basic ConTeXt Syntax Checking

  Occasionally useful on big projects, but be warned that it does not actually run any TEX engine, so it is not 100% reliable.

`colors`
: ConTeXt Color Management

  This displays icc color tables by name

`convert`
: ConTeXT Graphic Conversion Helpers

  A wrapper around ghostscript and imagemagick that offers some extra (batch processing) functionality.

`dvi`
: ConTeXt DVI Helpers

`epub`
: ConTeXt EPUB Helpers

  The EPUB manual (`epub-mkiv.pdf`) explains how to use this script.

`evohome`
: Evohome Fetcher

Evohome is a domotica system that controls your central heating

`fcd`
: Fast Directory Change

`flac`
: ConTeXt Flac Helpers

Extracts information from `.flac` audio files into an xml index.

`fonts`
: ConTeXt Font Database Management

`grep`
: Simple Grepper

`interface`
: ConTeXt Interface Related Goodies

`metapost`
: MetaPost to PDF processor

`metatex`
: MetaTeX Process Management (obsolete)

`modules`
: ConTeXt Module Documentation Generators

`package`
: Distribution Related Goodies

This script is used to create the generic ConTEXt code used in LuaLATEX c.s.

`patterns`
: ConTeXt Pattern File Management

Hyphenation patterns, that is . . .

`pdf`
ConTeXt PDF Helpers

`plain`
Plain TeX Runner

`profile`
ConTeXt MkIV LuaTeX Profiler

`rsync`
Rsync Helpers

`scite`
Scite Helper Script

`server`
Simple Webserver For Helpers

There are some subscripts associated with this.

`synctex`
ConTeXt SyncTeX Checker

`texworks`
TeXworks Startup Script

`timing`
ConTeXt Timing Tools

`tools`
Some File Related Goodies

`unicode`
Checker for `char-def.lua`

`unzip`
Simple Unzipper

`update`
ConTeXt Minimals Updater

`watch`
: ConTeXt Request Watchdog

`youless`
: YouLess Fetcher

  YouLess is a domotica system that tracks your home energy use.

11.9 Writing your own

A well-written script has some required internal structure. It should start with a module definition block. This gives some information about the module, but more importantly, it prevents double-loading.

Here is an example:

```
if not modules then modules = { } end

modules ['mtx-envtest'] = {
    version   = 0.100,
    comment   = "companion to mtxrun.lua",
    author    = "Taco Hoekwater",
    copyright = "Taco Hoekwater",
    license   = "bsd"
}
```

Next up is a variable containing the help information. The help information is actually a bit of xml stored in Lua string. In the full example listing at the end of this article, you can see what the internal structure is supposed to be like.

```
local helpinfo = [[
<?xml version="1.0"?>
<application>
  ....
</application>
]]
```

And this help information is then used to create an instance of an `application` table.

```
local application = logs.application {
    name     = "envtest",
    banner   = "Mtxrun environment demo",
```

```
    helpinfo = helpinfo,
}
```

After this call, the `application` table contains (amongst some other things) three functions that are very useful:

`identify()`
: Prints out a banner identifying the current script to the user.

`report(str)`
: For printing information to the terminal with the script name as prefix.

`export()`
: Prints the `helpinfo` to the terminal, so it can easily be used for documentation purposes.

Next up, it is good to define your scripts' functionality in functions in a private table. This prevents namespace pollution, and looks like this:

```
scripts          = scripts         or { }
scripts.envtest  = scripts.envtest or { }

function scripts.envtest.runtest()
    application.report("script name is " .. environment.ownname)
end
```

An finally, identify the current script, followed by handling the provided options (usually with an `if`–`else` statement).

```
if environment.argument("exporthelp") then
   application.export()
elseif environment.argument('test') then
   scripts.envtest.runtest()
else
    application.help()
end
```

11.10 Script environment

`mtxrun` includes lots of the internal Lua helper libraries from ConTEXt. We actually maintains a version of the script without all those libraries included, and before every (beta) ConTEXt release, an amalgamated version of `mtxrun` is added to the distribution. In the merging process, most all comments are stripped from the embedded

libraries, so if you want to know details, it is better to look in the original Lua source file.

Inside `mtxrun`, the full list of embedded libraries can be queried via the array `own.libs`:

l-lua.lua l-macro.lua l-sandbox.lua l-package.lua l-lpeg.lua l-function.lua l-string.lua l-table.lua l-io.lua l-number.lua l-set.lua l-os.lua l-file.lua l-gzip.lua l-md5.lua l-url.lua l-dir.lua l-boolean.lua l-unicode.lua l-math.lua util-str.lua util-tab.lua util-fil.lua util-sac.lua util-sto.lua util-prs.lua util-fmt.lua trac-set.lua trac-log.lua trac-inf.lua trac-pro.lua util-lua.lua util-deb.lua util-tpl.lua util-sbx.lua util-mrg.lua util-env.lua luat-env.lua lxml-tab.lua lxml-lpt.lua lxml-mis.lua lxml-aux.lua lxml-xml.lua trac-xml.lua data-ini.lua data-exp.lua data-env.lua data-tmp.lua data-met.lua data-res.lua data-pre.lua data-inp.lua data-out.lua data-fil.lua data-con.lua data-use.lua data-zip.lua data-tre.lua data-sch.lua data-lua.lua data-aux.lua data-tmf.lua data -lst.lua util-lib.lua luat-sta.lua luat-fmt.lua

In fact, the Lua table `own` contains some other useful stuff like the script's actual disk name and location (`own.name` and `own.path`) and some internal variables like a list of all the locations it searches for its embedded libraries (`own.list`), which is used by the `--selfmerge` option and also allows the non–amalgamated version to run (since otherwise `--selfmerge` could not be bootstrapped).

`mtxrun` offers a programming environment that makes it easy to write Lua a scripts. The most important element of that environment is a Lua table that is conveniently called `environment` (`util-env` does the actual work of setting that up).

The bulk of `environment` consists of functions and variables that deal with the command–line given by the user as `mtxrun` does quite a bit of work on the given command–line. The goal is to safely tuck all the given options into the `arguments` and `files` tables. This work is done by two functions called `initializearguments()` and `splitarguments()`. These functions are part of the `environment` table, but you should not need them as they have been called already once control is passed on to your script.

`arguments`
: These are the processed options to the current script. The keys are option names (without the leading dashes) and the value is either `true` or a string with one level of shell quotes removed.

`files`
: This array holds all the non-option arguments to the current script. Typically, those are supposed to be files, but they could be any text, really.

`getargument(name,partial)`
: Queries the `arguments` table using a function. Its main reason for existence is the `partial` argument, which allows scripts to accept shortened command-line options (alias: `argument()`).

`setargument(name,value)`
: Sets a value in the `arguments` table. This can be useful in complicated scripts with default options.

In case you need access to the full command-line, there are some possibilities:

`arguments_after`
: These are the unquoted but otherwise unprocessed arguments to your script as an array.

`arguments_before`
: These are the unquoted but otherwise unprocessed arguments to `mtxrun` before your scripts' name (so the last entry is usually `--script`).

`rawarguments`
: This is the whole unprocessed command-line as an array.

`originalarguments`
: Like `rawarguments`, but with some top-level quotes removed.

`reconstructcommandline(arg,noquote)`
: Tries to reconstruct a command-line from its arguments. It uses `originalarguments` if no `arg` is given. Take care: due to the vagaries of shell command-line processing, this may or may not work when quoting is involved.

`environment` also stores various bits of information you may find useful:

`validengines`
: This table contains keys for `luatex` and `luajittex`. This is only relevant when `mtxrun` itself is called via LuaTeX's `luaonly` option.

`basicengines`
: This table maps executable names to `validengines` entries.

`default_texmfcnf`
: This is the `texmfcnf` value from `kpathsea`, processed for use with MkIV in the unlikely event this is needed.

`homedir`
: The user's home directory.

`ownbin`
: The name of the binary used to call `mtxrun`.

`ownmain`
: The mapped version of `ownbin`.

`ownname`
: Full name of this instance of `mtxrun`.

`ownpath`
: The path this instance of `mtxrun` resides in.

`texmfos`
: Operating system root directory path.

`texos`
: Operating system root directory name.

`texroot`
: ConTEXt root directory path.

As well as some functions:

`texfile(filename)`
: Locates a TEX file.

`luafile(filename)`
: Locates a Lua file.

`loadluafile(filename,version)`
: Locates, compiles and loads a Lua file, possibly in compressed `.luc` format. In the compressed case, it uses the `version` to make sure the compressed form is up--to--date.

`luafilechunk(filename,silent,macros)`
Locates and compiles a Lua file, returning its contents as data.

`make_format(name,arguments)`
Creates a format file and stores in in the ConTEXt cache, used by `mtxrun --make`.

`relativepath(path,root)`
Returns a modified version of `root` based on the relative path in `path`.

`run_format(name,data,more)`
Run a TEX format file.

11.11 Shell return values

As explained earlier, the shell return value of `mtxrun` normally indicates whether the script was found. If you are running a ConTEXt release newer than September 2018 and want to modify the shell return value from within your script, you can use `os.exitcode`. Whatever value you assign to that variable will be the shell return value of your script.

Colofon

author Hans Hagen, PRAGMA ADE, Hasselt NL
Taco Hoekwater, extra `mtxrun` section
version January 1, 2019
website www.pragma-ade.nl – www.contextgarden.net