What tools do ConTEXt users have?

Abstract
This article describes the tools that are available for running ConTEXt, for testing, finding, analysing files, installing fonts, and more.

A bit of history

When we started working on ConTEXt, MS Windows (and before that 4Dos) was our main platform; it still is for development (we use Unix on the web and file servers and the Mac for fun). So, when ConTEXt was integrated into the TEX distributions we faced the problem of portability. Since one needs auxiliary programs[1] *e.g.* for sorting an index, we had written TEXutil, and the lack of a command line handler made us come up with TEXexec. Both were written in Modula but were rewritten in Perl in order to be usable on platforms other than MS Windows. It was easier to maintain a Perl version than to deal with low-level platform issues indefinitely.

As both our own and user demands grew, we wrote more tools and found out that they could best be written in Ruby. In the meantime TEXexec has been rewritten in Ruby, and relevant parts of TEXutil have been merged into it.

Launching scripts

Starting a script on an MS Windows box can be done using a so-called stub, a small program or command file with the same name that locates a similarly named script. On Unix some shell magic can be used to do the same or one can fall back on a magic preamble (a Bash/Perl mixture) fooling the operating system into locating and spawning the script using the right interpreter. By now, MS Windows has a convenient file association mechanism (but one has to activate it first) while Unix needs a (nowadays less path sensitive) shebang line and a suffix-less copy of the script.

Nevertheless we decided to come up with a less sensitive approach which also gave us the opportunity to accomplish a few more things: TEXMFstart. This script locates and executes a script (or program) in the TEX tree and executes it.

```
texmfstart texexec somefile.tex
```

When you incorporate TEX in work flows, calling TEXexec this way is rather future safe. Actually, because of this method, we could make the transition from TEXexec being a Perl script to being a Ruby program without too much trouble. A side effect of this way of launching scripts is that nested calls are faster because some information is passed on to child runs.

The script is also able to sort out a couple of things, for instance where files reside. Nowadays one will seldom use TEX alone and not all text processing (or related) programs have a clear concept of resource management and/or can work well with a TDS conforming tree.

```
texmfstart bin:xsltproc --output=new.xml \
  kpse:how.xsl old.xml
```

This[2] will locate the file `how.xsl` in the TEX tree and expand the file name to the full path. That way one can keep XSLT scripts organized as well. There are a few more such prefixes.

Other features are locating and showing documentation and launching editors with files located in the tree. The following call will open the `texmf.cnf` file that is currently used.

```
texmfstart --edit kpse:texmf.cnf
```

The script can initialize a tree so one can effectively run multiple trees in parallel. It does so by loading (when present) a file with variable specifications (more about that later).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

We often use a different tree for each project because commercial fonts may be project related and this way we can move a tree around without running into copyright problems (read: installing all fonts on each box).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

Another handy feature is conditional processing. In the following case the test file will only be processed when it has changed.

```
texmfstart --ifchanged=test.r --direct R \
  "-q --save --restore < test.r"
```

In a similar fashion one can make running dependent on time stamp comparison. More details can be found in the manual.

Managing ConTEXt runs

The TEXexec script manages a user's TEX run. There are many factors that influence such a run:

- Since ConTEXt uses the same format for all back ends, it depends on loading the relevant back end driver modules. Although one has complete control, life can be made easier when this is done automatically.
- A first pass may generate data needed in a successive pass. There may be references, tables of contents, indexes, etc. so we need a way to manage multiple runs. We have to make sure that neither fewer nor more runs than needed take place.
- A run may demand further action between runs, like graphic manipulations or delayed MetaPost execution.
- We may want to run different TEX engines, apply different back ends, use different user interfaces. Also, the name and way of calling TEX may change over time, something that we don't want users to be bothered with.
- We may want to process a TEX or XML file under different style regimes or enable style-specific modes.
- The document may need an additional page imposition pass, managed in such a way that no auxiliary data gets messed up.
- We may want to close and open the result in a viewer after the run is done.

This and a bit more is handled by TEXexec. When dealing with ConTEXt files the script will do a few things users are normally not aware of, like making sure that the random seed is frozen for a run, bugs in programs are caught (as long as needed) and that omissions in the `texmf.cnf` settings are compensated for. In addition TEXexec provides a few features for combining and manipulating PDF files.

The latest versions of TEXexec also support socalled ctx files. These are files in XML format that describe a process, additional pre- and post processing needed, styles and modules to be used, etc.[3] This means that one can easily configure projects without the need to tweak source files or editor setups or give explicit commands. Think of situations where an XML file (or bunch of files) has to be converted to another variant in order to be processed. TEXexec will only do that conversion when needed. In Figure 1 we show the file that is used in the MathAdore project.[4] The source file contains OpenMath and what we call 'shortcut math' and after normalizing this to OpenMath (first conversion) we convert the math to content MATHML (second conversion).

The source file contains a reference to this ctx file and when TEXexec is applied to the source file, it will take the appropriate actions. Such a reference looks like:

```
<?ctx-dir job ctxfile ../mathadore.ctx ?>
```

Here "`ctx-dir`" denotes a ConTEXt directive.

When dealing with a TEX file, TEXexec will scan the first line for comments that serve a similar purpose.

Handling the utility file

For a long time TEXutil was called from within TEXexec to handle the utility file that collects the index entries, tables of contents, references, etc. Nowadays this functionality is integrated in TEXexec which is more efficient. We also took the opportunity to enhance the sorting features so that one can mix language specific sorting rules.

The original TEXutil is also responsible for some other manipulations, like analyzing graphics. That kind of functionality has been moved to other scripts and more modern ways of dealing with such issues. Because we were in a transition stage to Ruby scripting, it was a good moment to say goodbye to TEXutil and concentrate on building a more extensive set of tools.

The tools collection

Instead of expanding TEXutil, we decided to spread functionality over multiple scripts. These can be recognized by their name: they all end with `tools`. If you call them using TEXMFstart there is not much opportunity for conflicts with existing tools.

Each tool comes with a manual, so we will not discuss details here.

ctxtools. This tool provides ConTEXt related features, like generating generic pattern files (so that we are independent), providing editor syntax checking files derived from the generic ConTEXt interface definition (handy for lexers), generating documentation (from the ConTEXt source code), updating ConTEXt (by downloading an archive and regenerating formats), etc.

rlxtools. The 'r' represents resources, normally graphics, the 'l' stands for libraries, and the 'x' (indeed) for XML. This tool can analyze graphic files and manipulate resources using other programs. For instance it can be used to down sample files at run time, to handle special color conversion, and to convert graphics to formats acceptable for TEX. By using the run time converters one can build work flows without the need to rely on additional scripting. There is a dedicated manual on this topic so we will not bore you here with yet another blob of XML.

```
<?xml version='1.0' standalone='yes'?>
<ctx:job>
    <ctx:message>mathadore</ctx:message>
    <ctx:preprocess suffix='prep'>
        <ctx:processors>
            <ctx:processor name='openmath' suffix='om'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-sm2om.xsl <ctx:value name='old'/>
            </ctx:processor>
            <ctx:processor name='mathadore' suffix='prep'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-openmath.xsl
                <ctx:value name='old'/>.om
            </ctx:processor>
        </ctx:processors>
        <ctx:files>
            <ctx:file processor='openmath,mathadore'>v*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>h*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>openmath*.xml</ctx:file>
        </ctx:files>
    </ctx:preprocess>
    <ctx:process>
        <ctx:resources>
            <ctx:environment>o-m4all.tex</ctx:environment>
        </ctx:resources>
    </ctx:process>
    <ctx:postprocess>
    </ctx:postprocess>
</ctx:job>
```

Figure 1. A ctx file used in the MathAdore project

xmltools. You can use this tool to do a simple analysis on an XML file. Another option is to generate a directory listing in XML format. In both cases, the result can be fed into ConTEXt and used in the process. A more obscure option is to generate images from MATHML snippets. This script will without doubt include more features in the future.

pdftools. This is work in progress. One can for instance roughly analyze PDF files. It also provides a way to manipulate colors in PDF images but that feature is now supported in ConTEXt directly.

textools. Users will seldom need this tool. It can fix things in a TDS compliant tree (for instance when the standard has changed), it deals with a few cross platform issues, it can help you to create so-called TPM archives (and is meant for ConTEXt module writers) and it can merge updates into your tree.

mpstools. In the future this tool will host the now standalone MetaPost to PDF wrapper (mptopdf) as well as the cropper (both are still Perl scripts).

tmftools. This script encapsulates some of the functionality of the Ruby based kpsewhich functionality that we use. In the future we may completely move away from the binary because the script is just as fast or faster when it serializes the database. The script can act as a kpsewhich server. The script can also analyze the tree for duplicates.

runtools. Because TEX is multi platform and because we (need to) run services on multiple platforms, we use this script to do things normally done at the console (shell). It just loads the given Ruby scripts with the appropriate library. We also use this tool to generate the ConTEXt distribution.

exatools. This is a more obscure tool. It provides some features related to form based style control and web driven TEX processing that we use in projects.

pstopdf. This last one is not a collection like the previous tools. It started long ago as a wrapper for Ghostscript. It still provides this function and over the years we've added quite a bit of filtering to it (we just filter the things that Ghostscript fails on or gets confused from). In the meantime we cheat on the name since it also manages the conversion of bitmap images, especially cached down sampling, using ImageMagick as

```
# file   : setuptex.tmf (the less generic version have suffixes like cmd, sh, csh etc)
# author : Hans Hagen - PRAGMA ADE - Hasselt NL - www.pragma-ade.com
# usage  : texmfstart --tree=f:/minimal/tex ...
#
# this assumes that calling script sets TEXPATH without a trailing
# slash; %VARNAME% expands to the environment variable, $VARNAME
# is left untouched; we also assume that TEXOS is set.

TEXMFMAIN    = %TEXPATH%/texmf
TEXMFLOCAL   = %TEXPATH%/texmf-local
TEXMFFONTS   = %TEXPATH%/texmf-fonts
TEXMFPROJECT = %TEXPATH%/texmf-project
VARTEXMF     = %TMP%/texmf-var
HOMETEXMF    =

TEXMFOS      = %TEXPATH%/%TEXOS%

TEXMFCNF     = %TEXPATH%/texmf{-local,}/web2c
TEXMF        = {$TEXMFOS,$TEXMFPROJECT,$TEXMFFONTS,$TEXMFLOCAL,!!$TEXMFMAIN}
TEXMFDBS     = $TEXMF

TEXFORMATS   = %TEXMFOS%/web2c/{$engine,}
MPMEMS       = %TEXFORMATS%
TEXPOOL      = %TEXFORMATS%
MPPOOL       = %TEXPOOL%

PATH         > %TEXMFOS%/bin
PATH         > %TEXMFLOCAL%/scripts/perl/context
PATH         > %TEXMFLOCAL%/scripts/ruby/context

TEXINPUTS    =
MPINPUTS     =
MFINPUTS     =
```

Figure 2. Example `texmf.cnf` file

well as conversion from SVG to PDF using Inkscape.

texfont. This script has been around for a while now and is used to install (commercial) fonts. It generates metric files, map files, and a demo file so that one can see if things went right. ConTEXt does not depend on (ever changing) map file methods and loads map files on demand. You can generate map files for dvipdfmx with the previously mentioned `ctxtools`.

More

There are a few more scripts, like `concheck` (simple syntax checking) and `texsync` (synchronizing minimal distributions) but we will not discuss them here.

Integration

When setting up multiple TEX trees, the trick is in isolating them as much as possible. Because one can never be sure how distributions set things up, we revert to setting environment variables, which will then take precedence over the settings in a regular `texmf.cnf` file. In the TEXMFstart manual you can find more details on how we take care of this; here we only show an example of such an file in Figure 2.

When the tree flag is given, TEXMFstart will read this file and set the environment variables accordingly before it launches the program it is supposed to start. In fact, a tree specification can specify a file, but by default the `setyptex` one is taken.

```
texmfstart \
  --tree=f:/minimal/tex/setuptex.tmf \
  texexec test.tex
```

Since TEXMFstart can load several such files, we can also use this method to preset more environment variables, for instance pointers to resources like graphics. This is what the `--env` or `--environment` option is for, as in:

```
texmfstart --tree=f:/minimal/tex \
  --env=xyz.tmf texexec test.tex
```

The advantage of this variable setting game is that instead of cooking up scripts with statements like:

```
thread.new do
 ENV["something"] = "nothing"
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

we can put the variable definition in a file and say:

```
thread.new do
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

This has not only a big advantage in terms of isolation (and maintenance) but is also more robust since one can never be sure if another thread is not setting the same variable too, thereby creating much confusion for all the other threads that use the same variable. Since TEXMFstart runs as a separate process, it can set its variables independently.

Whenever (on the ConTEXt mailing list) you see mentioning of something named `setuptex`, you can be sure that it relates to initializing a TEX tree (probably a minimal ConTEXt tree) in an isolated way.

Conclusion

In this short article we have tried to give you an impression of what is needed in order to make TEX usable in a diversity of today's environments. It was not our intention to be complete, because for that purpose we have manuals. One thing should be made clear: although TEX itself is pretty stable, the same cannot be said for the environment that it is used in. Just telling TEX to process a file is not enough nowadays. This also means that ConTEXt and its tools, in order to keep up, need to be adapted to current needs. On the other hand, by organizing the functionality in tools, and by using a modern and reliable scripting language like Ruby, users don't pay a high price for this. Most nasty details can be hidden from them.

Notes

1. We will use the terms 'scripts' and 'programs' interchangeably.
2. The backslash at the end of line denotes a continued line.
3. Although one can use the `ctx` suffix for ConTEXt related TEX files, this is normally a bad idea.
4. This project will provide highly interactive math to schools and is conducted in cooperation with the University of Eindhoven.

Hans Hagen