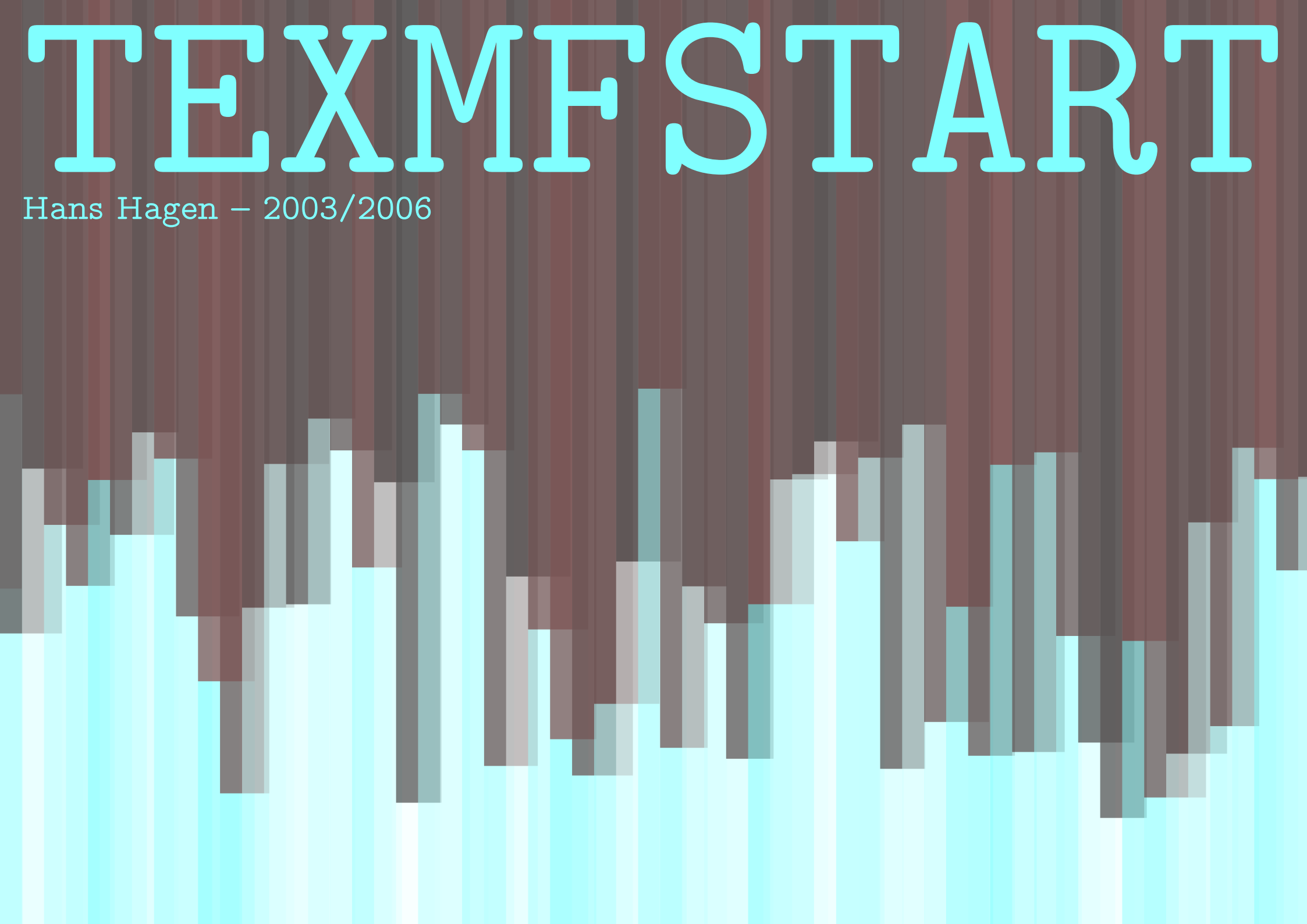
TEXMFSTART

Hans Hagen – 2003/2006



Introduction

This manual is about a small (RUBY) script that can be used to run a script or open a document which is located somewhere in the `texmf` tree. This scripts evolved out of earlier experiments and is related to scripts and programs like `runperl`, `runruby` and `irun`.

One of the main reasons for `texmfstart` to exist is that it enables us to be downward compatible when using a TEX based environment. TEX itself is pretty stable, but this is not true for the whole collection of files that comes with a distribution and the way they are organized. We will see some other reasons for using this script as well.

We can also use this script for lanching applications that need access to resources in the TEX tree but that lack the features to locate them.

The script has a few dependencies on libraries. This means that relocating the script to a bin path may give problems. One can make a self–contained version by saying:

```
texmfstart --selfmerge
```

One can undo this with the `--selfclean` option. Normally users don't have to worry about this because in the CONTEXT distribution the merged version is shipped. A MS WINDOWS (pseudo) binary can be made with `exerb` or one can simply associate the `.rb` suffix with the RUBY program.

```
FTYPE RubyScript=c:\data\system\ruby\bin\ruby.exe %%1 %%*

ASSOC .rb=RubyScript
ASSOC .rbw=RubyScript
```

On UNIX one can make a copy without suffix:

```
cp texmfstart.rb /path/to/bin/texmfstart
chmod +x texmfstart
```

Alternative approaches have been discussed on the CONTEXT and TEXLive mailing lists and can be found in their archives.

Launching programs

The primary usage of `texmfstart` is to launch programs and scripts. We can start the `texexec` PERL script with:

```
texmfstart texexec.pl --pdf somefile
```

We can also start the `pstopdf` RUBY script:

```
texmfstart pstopdf.rb --method=3 cow.eps
```

However, we can omit the suffix:

```
texmfstart texexec --pdf somefile
texmfstart pstopdf --method=3 cow.eps
```

The suffixless method is slower unless the scripts are known. For familiar CONTEXT scripts it's best not to use the suffix since this permits us to change the scripting language. CONTEXT related scripts are known. Because in the meantime `texexec` has become a RUBY script, users who use the suffixless method automatically will get the right version.

You can also say:

```
texmfstart --file=pstopdf --method=3 cow.eps
```

When locating a file to run, several methods are applied, one being `kpsewhich`. You can control the path searching by providing a program space, which by default happens to be `context`.

```
texmfstart --program=context --file=pstopdf --method=3 cow.eps
```

The general pattern is:

```
texmfstart switches filename arguments
```

Here `switches` control `texmfstart`'s behaviour, and `arguments` are passed to the program identified by `filename`.

Sometimes the operating system will spoil our little game of passing arguments. In the following case we want the output of `texexec` to be written to a log file. By using quotes, we can pass the redirection without problems.

```
texmfstart texexec "somefile.tex > whatever.log"
```

Generating stubs

One of the reasons for writing `texmfstart` is that it permits us to write upward compatible scripts (batch files), so instead of

```
texexec --pdf somefile
texexec --pdf anotherfile
```

We prefer to use:

```
texmfstart texexec --pdf somefile
texmfstart texexec --pdf anotherfile
```

Instead of using `texmfstart` directly you can also use it in a stub file. For MS WINDOWS such a file looks like:

```
@echo off
texmfstart texexec %*
```

In this case, the file itself is named `texexec.cmd`. Now, given that no new functionality of `texmfstart` itself is needed, one will automatically use the version of `texexec` that is present in the (latest) installed CONTEXT tree.

It is possible to generate stubs automatically. You can provide a path where the stub will be written. This permits tricks like the following. Say that on a CDROM we have the following structure:

```
tex/texmf-mswin/bin/texexec.bat
tex/texmf-linux/bin/texexec
tex/texmf-local/scripts/context/ruby/texexec.rb
```

If we are on the main `tex` path, we can run `texmfstart` as follows:

```
texmfstart --make --windows --stubpath=tex/texmf-mswin/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
texmfstart --make --unix    --stubpath=tex/texmf-linux/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
```

This will generate start up scripts that point directly to the PERL script. Such a link may fail when files get relocated. In that case you can use the `--indirect` directive, which will force the `texmfstart` into the stub file.

```
texmfstart --make --windows --indirect --stubpath=tex/texmf-mswin/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
texmfstart --make --unix    --indirect --stubpath=tex/texmf-linux/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
```

However, the prefered way and most simple way to generate the stubs for the scripts that come with CONTEXT is:

```
texmfstart --make all
```

This will generate stubs suitable for the current operating system in the current path.

Documents

You can use `texmfstart` to open a document.

```
texmfstart showcase.pdf
```

This will open the document `showcase.pdf`, when found. The chance is minimal that such a document can be located by `kpsewhich`. In that case, `texmfstart` will search the tree itself.

Given that it is supported on your platform, you can also open a PDF file on a given page.

```
texmfstart --page=2 showcase.pdf
```

On MS WINDOWS the following command will open the PDF file in a web browser. This is needed when you want support for form submission.

```
texmfstart --browser examplap.pdf
```

Search strategy

In a first attempt, `kpsewhich` will be used to locate a file. When `kpsewhich` cannot locate the file, the following environment variables will be used:

`RUBYINPUTS`	ruby scripts with suffix `rb`
`PERLINPUTS`	perl scripts with suffix `pl`
`PYTHONINPUTS`	python scripts with suffix `py`
`JAVAINPUTS`	java archives with suffix `jar`
`PDFINPUTS`	pdf documents with suffix `pdf`

It using them fails as well, the whole tree is searched, which will take some time.

When a file found, its location is remembered and passed on to nested runs. So, in general, a nested run will start faster.

Directives

The script accepts a few directives. Some are rather general:

`--verbose`	report some status and progress information
`--arguments`	an alternative for providing the arguments to be passed
`--clear`	don't pass info about locations to child processes

Directives that concern starting an application are:

`--program=str`	the program space where `kpsewhich` will search
`--locate`	report the call as it should happen (no newline)
`--report`	report the call as it should happen (simulated)
`--browser`	start the document in a web browser
`--file`	an alternative for providing the file
`--direct`	run a program without searching for it's location
`--execute`	use RUBY's 'exec' instead of 'system'
`--batch`	*not yet implemented*

You can create startup scripts by providing one of the following switches in combination with a filename.

`--make`	create a start script or batch file for the given program
`--windows`	when making a startup file, create a windows batch file
`--linux`	when making a startup file, create a unix script
`--stubpath`	destination of the startup file
`--indirect`	always use `texmfstart` in a stub file

Some directives can be accompanied by specifications, like:

`--page=n`	open the document at this page
`--path=str`	change from the current path to the given path
`--before=str`	*not yet implemented*
`--after=str`	*not yet implemented*
`--tree=str`	use the given TEX tree
`--autotree`	automatically determine the TEX tree to use
`--environment=str`	use the given tmf environment file

Conditional directives are:

`--iftouched=str,str`	only run when the given files have different time stamps
`--ifchanged=str`	only run when the given file has changed (md5 check)

Special features:

`--showenv`	show the environment variables known at runtime
`--edit`	open the given file in an editor

In addition, there are prefixes for filenames:

`bin:filename`	expanded name, based on `PATH` environment variable
`kpse:filename`	expanded name, based on `kpsewhich` result
`rel:filename`	expanded name, backtracking on current path (`. .. ../..`)
`env:name`	expanded name, based on environment variable `name`
`path:filename`	pathpart of `filename` as located by `kpsewhich`

Performance

The performance of the indirect call is of course less than a direct call. You can gain some time by setting the environment variables or by using a small TEX tree.

The script tries to be clever. First it tries to honor a given path, and if that fails it will strip the path part and look on the current path. When this fails, it will consult the environment variables. Then it will use `kpsewhich` and when that fails as well, it will start searching the TEX trees. This may take a while, especially when you have a complete tree, like the one on TEX Live.[1]

If you want, you can use the built in `kpsewhich` functionality (written in RUBY) by setting the environment variable `KPSEFAST` to `yes`. The built in handler is a bit faster and maintains its own file database. Such a database is generated with:

```
tmftools --reload
```

[1] On my computer I use multiple trees parallel to the latest TEX Live tree. This results in a not that intuitively and predictable search process. The cover of this manual reflects state of those trees.

Using prefixes

You can also use `texmfstart` to launch other programs that need files in one of the TEX trees:

```
texmfstart --direct xsltproc kpse:somescript.xsl somefile.xml
```

or shorter:

```
texmfstart bin:xsltproc kpse:somescript.xsl somefile.xml
```

In both cases `somescript.xsl` will be resolved and in the second case `bin:` will be stripped. The `--direct` switch and `bin:` prefix tell `texmfstart` not to search for the program, but to assume that it is a binary. The `kpse:` prefix also works for previously mentioned usage.

A convenient way to edit your local context system setup file is the following; we don't need to go to the path where the file resides.

```
texmfstart bin:scite kpse:cont-sys.tex
```

Because editing is happening a lot, you can also say:

```
texmfstart --edit kpse:cont-sys.tex
```

You can set the environment variable `TEXMFSTART_EDITOR` to your favourite editor.

Conditional processing

A bit obscure feature is triggered with `--iftouched`, for instance:

```
texmfstart --iftouched=normal.pdf,lowres.pdf \
    downsample.rb --verylow normal.pdf lowres.pdf
```

Here, `downsample.rb` is only executed when `normal.pdf` and `lowres.pdf` have a different modification time. After execution, the times are synchronized. This feature is rather handy when you want to minimize runtime. We use it in the resource library tools.

```
texmfstart --iftouched=foo.bar,bar.foo convert_foo_to_bar.rb
```

A similar option is `ifchanged`:

```
texmfstart --ifchanged=whatever.mp texexec --mpgraphic whatever.mp
```

This time we look at the MD5 checksum, when the sum is changed, `texexec` will be run, otherwise we continue.

TEX trees

There are a few more handy features built in. The reason for putting those into this launching program is that the sooner they are executed, the less runtime is needed later in the process.

Imagine that you have installed your tree on a network attached storage device. In that case you can say:

```
texmfstart --tree=//nas-1/tex texexec --pdf yourfile
```

There should be a file `setuptex.tmf` in the root of the tree. An example of such a file is part of the CONTEXT distribution (minimal trees). This feature permits you to have several trees alongside and run specific ones. You can also specify additional environments, using `--environment`.

Such an environment file is platform independent and looks as follows. The `%VAR%` variables will be replaced by their meaning, while the `$VAR` variables are left untouched. The = sets a value, while > and < prepend and append the given value to the current value.

```
# 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
TEXMFEXTRA   = %TEXPATH%/texmf-extra
TEXMFPROJECT = %TEXPATH%/texmf-project
VARTEXMF     = %TMP%/texmf-var
HOMETEXMF    =

TEXMFOS      = %TEXPATH%/%TEXOS%
# OSFONTDIR  = %SYSTEMROOT%/fonts

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

```
RUBYLIB      > %TEXMFLOCAL%/scripts/ruby/context

TEXINPUTS    =
MPINPUTS     =
MFINPUTS     =
```

When you only want to set a variable that has no value yet, you can use an `?`. These symbols have alternatives as well:

`=`	`<<`	assign a value to the variable
`?`	`??`	only assign a valuehen the variable is unset
`<`	`+=`	append a value to the current value of the variable
`>`	`=+`	prepend a value to the current value of the variable