



SciTiTE
IN CONTEXT MkIV & LMTX

About SCiTE

For a long time at Pragma ADE we used $\text{T}_{\text{E}}\text{X}$ edit, an editor we'd written in Modula. It had some project management features and recognized the project structure in $\text{ConT}_{\text{E}}\text{Xt}$ documents. Later we rewrote this to a platform independent reimplementaion called $\text{T}_{\text{E}}\text{X}$ work written in Perl/Tk (not to be confused with the editor with the plural name) that when I checked last still works okay, which is proof that Perl/Tk has been stable for decades.

In the beginning of the century I can into SciTE, written by Neil Hodgson. Although the mentioned editors provide some functionality not present in SciTE we decided to use that editor because it frees us from maintaining our own. I ported our $\text{T}_{\text{E}}\text{X}$ and MetaPost (line based) syntax highlighting to SciTE and got a lot of others for free.

After a while I found out that there was an extension interface written in Lua. I played with it and wrote a few extensions too. This pleasant experience later triggered the $\text{LuaT}_{\text{E}}\text{X}$ project.

A decade into the century SciTE got another new feature: you can write dynamic external lexers in Lua using lpeg. As in the meantime $\text{ConT}_{\text{E}}\text{Xt}$ has evolved in a $\text{T}_{\text{E}}\text{X}$ /Lua hybrid, it made sense to look into this. The result is a couple of lexers that suit $\text{T}_{\text{E}}\text{X}$, MetaPost and Lua usage in $\text{ConT}_{\text{E}}\text{Xt}$ MkIV. As we also use xml as input and output format a lexer for xml is also provided. And because pdf is one of the backend formats lexing of pdf is also implemented. In the process some of the general lexing framework were adapted to suit our demands for speed. For a long time we shipped these files as well but at point I decided that it made no sense to keep adapting to the relatively frequent changes in the api. The last version in the 3.* series worked okay, in the 4.* series things failed but we didn't adapt, and when series 5.* showed up I decided to drop the old lexer compatibility. I assume that a version of SciTE is run that has lpeg available in the main Lua instance and that also supports copying text fragments using the editor object. (Till that is the case, we provide binaries with the $\text{ConT}_{\text{E}}\text{Xt}$ distribution.)

In the $\text{ConT}_{\text{E}}\text{Xt}$ (standalone) distribution you will find the relevant files under:

```
<texroot>/tex/texmf-context/context/data/scite
```

Normally a user will not have to dive into the implementation details but in principle you can tweak the properties files to suit your purpose.

The look and feel

The color scheme that we use is consistent over the lexers but we use more colors than in the traditional lexing. For instance, $\text{T}_{\text{E}}\text{X}$ primitives, low level $\text{T}_{\text{E}}\text{X}$ commands, $\text{T}_{\text{E}}\text{X}$ constants, basic file structure related commands, and user commands all get a different treatment. When spell checking is turned on, we indicate unknown words, but also words that are known but might need checking, for instance because they have an uppercase character. In figure 1 we see some of that in practice.

Installing SCiTE

Installing SciTE is straightforward. We are most familiar with MS Windows but for other operating systems installation is not much different. First you need to fetch the archive from:

www.scintilla.org

```

t:\scite\data\scite-context-visual.tex - ScITE
File Edit Search View Tools Options Language Buffers Help
1 tebooks.tex | 2 d-buildtool.lua | 3 s-methods.lua | 4 s-jobcontrol.lua | 5 s-sessions.lua | 6 d-runtool.lua | 7 mtex-example.lua | 8 demoticket-m4all.xml | 9 m4all.lua | 0 d-dispatchers.lua | 1 crap.tex | scite-context-readme.tex | scite-context-visual.tex
>mtxrun --autogenerate --script context --autopdf scite-context-visual.tex
mtx-context | run 1: luatex --fmt="c:/data/develop/tex-context/tex/texmf-
This is LuaTeX, Version beta-0.71.0-2011062811 (rev 4315)
\write18 enabled.
(scite-context-visual.tex)
ConTeXt ver: 2011.11.08 19:35 MKIV fmt: 2011.11.8 int: english/english
system > cont-new.mkiv loaded
(c:/data/develop/context/sources/cont-new.mkiv)
system > Beware: some patches loaded from cont-new.mkiv
)
system > cont-loc.mkiv loaded
(c:/data/develop/context/sources/cont-loc.mkiv)
system > Beware: some patches loaded from cont-loc.mkiv
)
system > cont-exp.mkiv loaded
(c:/data/develop/context/sources/cont-exp.mkiv)
system > Beware: some patches loaded from cont-exp.mkiv
)
system > scite-context-visual.top loaded
(scite-context-visual.top)
languages > language en is active
> latin modern fonts are not preloaded
{c:/data/develop/tex-context/tex/texmf-context/fonts/map/pdf/tex/context/mkiv-
fonts} > preloading latin modern fonts (second stage)
(c:/data/develop/context/sources/type-siz.mkiv) (c:/data/develop/context/sour
fonts > virtual math > unable to resolve name mapstrfromchar
structure > sectioning > Chapter @ level 2 : 0.1 -> Some fancy title
metapost > initializing instance 'metafun' using format 'metafun'
metapost > loading 'metafun': c:/data/develop/context/metapost/context
backend > xmp > using file 'c:/data/develop/context/sources/pdf-pdx.
pages > flushing realpage 1, userpage 1, subpage 1
) < c:/data/develop/tex-context/tex/texmf/fonts/opentype/public/lm/lmroman12-r
mkiv lua stats > used config file
mkiv lua stats > used cache path
mkiv lua stats > resource resolver
mkiv lua stats > stored bytecode data
mkiv lua stats > cleaned up reserved nodes - 39 nodes, 9 lists of 427
mkiv lua stats > node memory usage
mkiv lua stats > node list callback tasks - 2 glue, 2 penalty, 12 attribute
mkiv lua stats > used backend
mkiv lua stats > used backend
mkiv lua stats > loaded patterns
mkiv lua stats > callbacks
mkiv lua stats > randomized
mkiv lua stats > lxm preparation time
mkiv lua stats > result saved in file
mkiv lua stats > loaded fonts
mkiv lua stats > fonts load time
mkiv lua stats > metapost processing time - 0.016 seconds, loading: 0.078 s
mkiv lua stats > luatex banner
mkiv lua stats > control sequences
mkiv lua stats > current memory usage
mkiv lua stats > runtime
mkiv lua stats >
mtx-context | pdfview methods: Acrobat default okular, current method: ac
system | total runtime: 2.264sExit code: 0

```

```

1 % language=uk
2
3 \defineframedtext
4 [entry]
5
6 \starttext
7
8 \startchapter[title=Some fancy title]
9
10 \startluacode
11 local entries = { -- there can be more
12 { text = "The third entry!" },
13 { text = "The fourth entry!" },
14 }
15
16 for i=1,#entries do
17 context.startentry()
18 context.entries[i].text
19 context.stopentry()
20 end
21 \stopluacode
22
23 This is just some text to demonstrate the realtime spellchecker
24 in combination with the embedded lua and metapost lexers and
25 inline as well as display \ctxlua{context("lua code")}.
26
27 \startlinecorrection
28 \startMPcode
29 for i=1 upto 100 :
30 draw fullcircle scaled (i*mm) ;
31 endfor ;
32 \stopMPcode
33 \stoplinecorrection
34
35 \iftrue \def\crap{some text} % who cares
36 \else \def\crap{some crap} % about this
37 \fi
38
39
40
41 \blank[2*big]
42
43 \crap
44
45 \stopchapter
46
47 \stoptext

```

```

scite-context-visual.tex | 5-10-2011 23:12:25 | line 23 column 31 | mode INS | eol CR+LF | 5-10-2011 23:12:25

```

Figure 1 Nested lexers in action.

The MS Windows binaries are zipped in `wscite.zip`, and you can unzip this in any directory you want as long as you make sure that the binary ends up in your path or as shortcut on your desktop. So, say that you install SciTE in:

```
c:\data\system\scite\wscite
```

You need to add this path to your local path definition. Installing SciTE to some known place has the advantage that you can move it around. There are no special dependencies on the operating system.

On MS Windows you can for instance install SciTE in:

```
c:\data\system\scite
```

and then end up with:

```
c:\data\system\scite\wscite
```

and that is the path you need to add to your environment `PATH` variable.

On linux the files end up in:

```
/usr/bin
/usr/share/scite
```

Where the second path is the path we will put more files.

Binaries

When you compile binaries yourself or get them from somewhere you need to make sure that they end up in the right place (see previous section): When you're on MS Windows they fly to:

```
wscite/scite.exe
wscite/scilexer.dll
```

And on linux then end up in:

```
/usr/bin/SciTE
/usr/bin/libscintilla.so
/usr/bin/liblexilla.so
```

Because we only use the official Lua interface methods the lexers should just work, assuming that you have imported the `context/scite-context-user` properties file.

Installing the CONTEXT lexers

If you want to use ConTEXT, you need to copy the relevant files from

```
<texroot>/tex/texmf-context/context/data/scite
```

to the path where SciTE keeps its property files (`*.properties`). This is the path we already mentioned. There should be a file there called `SciteGlobal.properties`.

So, in the end you get on MS Windows new files in:

```
c:\data\system\scite\wscite
c:\data\system\scite\wscite\context
c:\data\system\scite\wscite\context\lexer
c:\data\system\scite\wscite\context\lexer\themes
c:\data\system\scite\wscite\context\lexer\data
c:\data\system\scite\wscite\context\documents
```

while on linux you get:

```
/usr/bin/share/
/usr/bin/share/context
/usr/bin/share/context/lexer
/usr/bin/share/context/lexer/themes
/usr/bin/share/context/lexer/data
/usr/bin/share/context/documents
```

At the end of the `SciteGlobal.properties` you need to add the following line:

```
import context/scite-context-user
```

After this, things should run as expected (given that $\text{T}_{\text{E}}\text{X}$ runs at the console as well).

Fonts

The configuration file defaults to the DejaVu fonts. These free fonts are part of the Con $\text{T}_{\text{E}}\text{X}$ t suite (also known as the standalone distribution). Of course you can fetch them from <http://dejavu-fonts.org> as well. You have to copy them to where your operating system expects them. In the suite they are available in:

```
<contextroot>/tex/texmf/fonts/truetype/public/dejavu
```

Extensions

Just a quick note to some extensions. If you select a part of the text (normally you do this with the shift key pressed) and you hit `Shift-F11`, you get a menu with some options. More (robust) ones will be provided at some point.

Spell checking

If you want to have spell checking, you need have files with correct words on each line. The first line of a file determines the language:

```
% language=uk
```

When you use the external lexers, you need to provide some files. Given that you have a text file with valid words only, you can run the following script:

```
mtxrun --script scite --words nl uk
```

This will convert files with names like `spell-nl.txt` into Lua files that you need to copy to the

`lexers/data` path. Spell checking happens realtime when you have the language directive (just add a bogus character to disable it). Wrong words are colored red, and words that might have a case problem are colored orange. Recognized words are greyed and words with less than three characters are ignored.

A spell checking file has to be put in the `lexers/data` directory and looks as follows (e.g. `spell-uk.lua`):

```
return {
  ["max"]=40,
  ["min"]=3,
  ["n"]=151493,
  ["words"]={
    ["aardvark"]="aardvark",
    ["aardvarks"]="aardvarks",
    ["aardwolf"]="aardwolf",
    ["aardwolves"]="aardwolves",
    ...
  }
}
```

The keys are words that get checked for the given value (which can have uppercase characters). The word files are not distributed (but they might be at some point).

In the case of internal lexers, the following file is needed:

```
spell-uk.txt
```

If you use the traditional lexer, this file is taken from the path determined by the environment variable:

```
CTXSPELLPATH
```

As already mentioned, the lpeg lexer expects them in the data path. This is because the Lua instance that does the lexing is rather minimalistic and lacks some libraries as well as cannot access the main SciTE state.

Spell checking in `txt` files is enabled by adding a first line:

```
[#!-%] language=uk
```

The first character on that line is one of the four mentioned between square brackets. So,

```
# language=uk
```

should work. For xml files there are two methods. You can use the following (at the start of the file):

```
<?xml ... language="uk" ?>
```

But probably better is to use the next directive just below the usual xml marker line:

```
<?context-directive editor language uk ?>
```

Interface selection

In a similar fashion you can drive the interface checking:

```
% interface=nl
```

Property files

The internal lexers are controlled by the property files while the external ones are steered with themes. Unfortunately there is hardly any access to properties from the external lexer code nor can we consult the file system and/or run programs like `mtxrun`. This means that we cannot use configuration files in the ConT_EXt distribution directly. Hopefully this changes with future releases.

The external lexers

These are the more advanced lexers. They provide more detail and the ConT_EXt lexer also supports nested MetaPost and Lua. Currently there is no detailed configuration but this might change once they are stable.

The external lexers operate on documents while the internal ones operate on lines. This can make the external lexers slow on large documents. We've optimized the code somewhat for speed and memory consumption but there's only so much one can do. While lexing each change in style needs a small table but allocating and garbage collecting many small tables comes at a price. Of course in practice this probably gets unnoticed.¹

The external lpeg lexers work okay with the MS Windows and linux versions of SciTE, but unfortunately at the time of writing this, the Lua library that is needed is not available for the MacOSX version of SciTE. Also, due to the fact that the lexing framework is rather isolated, there are some issues that cannot be addressed in the properly, at least not currently.

In addition to ConT_EXt and MetaFun lexing a Lua lexer is also provided so that we can handle ConT_EXt Lua Document (cld) files too. There is also an xml lexer. This one also provides spell checking. The pdf lexer tries to do a good job on pdf files, but it has some limitations. There is also a simple text file lexer that does spell checking. Finally there is a lexer for cweb files.

Don't worry if you see an orange rectangle in your T_EX or xml document. This indicates that there is a special space character there, for instance `0xA0`, the nonbreakable space. Of course we assume that you use utf8 as input encoding.

The internal lexers

SciTE has quite some built in lexers. A lexer is responsible for highlighting the syntax of your document. The way a T_EX file is treated is configured in the file:

```
tex.properties
```

You can edit this file to your needs using the menu entry under `options` in the top bar. In this file, the following settings apply to the T_EX lexer:

```
lexer.tex.interface.default=0
```

¹ I wrote the code in 2011 on a more than 5 years old Dell M90 laptop, so I suppose that speed is less an issue now.

```
lexer.tex.use.keywords=1
lexer.tex.comment.process=0
lexer.tex.auto.if=1
```

The option `lexer.tex.interface.default` determines the way keywords are highlighted. You can control the interface from your document as well, which makes more sense than editing the configuration file each time.

```
% interface=all|tex|nl|en|de|cz|it|ro|latex
```

The values in the properties file and the keywords in the preamble line have the following meaning:

```
0 all      all commands (preceded by a backslash)
1 tex      TEX, ε-TEX, pdfTEX, Omega primitives (and macros)
2 nl       the dutch ConTEXt interface
3 en       the english ConTEXt interface
4 de       the german ConTEXt interface
5 cz       the czech ConTEXt interface
6 it       the italian ConTEXt interface
7 ro       the romanian ConTEXt interface
8 latex    LATEX (apart from packages)
```

The configuration file is set up in such a way that you can easily add more keywords to the lists. The keywords for the second and higher interfaces are defined in their own properties files. If you're curious about the way this is configured, you can peek into the property files that start with `scite-context`. When you have ConT_EXt installed you can generate configuration files with

```
mtxrun --script interface --scite
```

You need to make sure that you move the result to the right place so best not mess around with this command and use the files from the distribution.

Back to the properties in `tex.properties`. You can disable keyword coloring altogether with:

```
lexer.tex.use.keywords=0
```

but this is only handy for testing purposes. More interesting is that you can influence the way comment is treated:

```
lexer.tex.comment.process=0
```

When set to zero, comment is not interpreted as T_EX code and it will come out in a uniform color. But, when set to one, you will get as much colors as a T_EX source. It's a matter of taste what you choose.

The lexer tries to cope with the T_EX syntax as good as possible and takes for instance care of the funny `^^` notation. A special treatment is applied to so called `\if`'s:

```
lexer.tex.auto.if=1
```

This is the default setting. When set to one, all `\ifwhatever`'s will be seen as a command. When set to zero, only the primitive `\if`'s will be treated. In order not to confuse you, when this property is set to one, the lexer will not color an `\ifwhatever` that follows an `\newif`.

The MetaPost lexer

The MetaPost lexer is set up slightly different from its T_EX counterpart, first of all because MetaPost is more a language than T_EX. As with the T_EX lexer, we can control the interpretation of identifiers. The MetaPost specific configuration file is:

```
metapost.properties
```

Here you can find properties like:

```
lexer.metapost.interface.default=1
```

Instead of editing the configuration file you can control the lexer with the first line in your document:

```
% interface=none|metapost|mp|metafun
```

The numbers and keywords have the following meaning:

| | | |
|---|----------------|--------------------------------|
| 0 | none | no highlighting of identifiers |
| 1 | metapost or mp | MetaPost primitives and macros |
| 2 | metafun | MetaFun macros |

Similar to the T_EX lexer, you can influence the way comments are handled:

```
lexer.metapost.comment.process=1
```

This will interpret comment as MetaPost code, which is not that useful (opposite to T_EX, where documentation is often coded in T_EX).

The lexer will color the MetaPost keywords, and, when enabled also additional keywords (like those of MetaFun). The additional keywords are colored and shown in a slanted font.

The MetaFun keywords are defined in a separate file:

```
metafun-scite.properties
```

You can either copy this file to the path where your global properties files live, or put a copy in the path of your user properties file. In that case you need to add an entry to the file `SciTEUser.properties`:

```
import metafun-scite
```

The lexer is able to recognize `btex-etex` and will treat anything in between as just text. The same happens with strings (between `"`). Both act on a per line basis.

Using ConT_EXt

When `mtxrun` is in your path, ConT_EXt should run out of the box. You can find `mtxrun` in:

```
<contextroot>/tex/texmf-mswin/bin
```

or in a similar path that suits the operating system that you use.

When you hit **CTRL-12** your document will be processed. Take a look at the **Tools** menu to see what more is provided.

Extensions (using LUA)

When the Lua extensions are loaded, you will see a message in the log pane that looks like:

- see `scite-ctx.properties` for configuring info
- `ctx.spellcheck.wordpath` set to `ENV(CTXSPELLPATH)`
- `ctxspellpath` set to `c:\data\develop\context\spell`
- `ctx.spellcheck.wordpath` expands to `c:\data\develop\context\spell`
- `ctx.wraptext.length` is set to 65
- key bindings:

`Shift + F11` pop up menu with ctx options

`Ctrl + B` check spelling
`Ctrl + M` wrap text (auto indent)
`Ctrl + R` reset spelling results
`Ctrl + I` insert template
`Ctrl + E` open log file
`Ctrl + +` show language character strip (key might change)

- recognized first lines:

```
xml <?xml version='1.0' language='nl'
tex % language=nl
```

This message tells you what extras are available. The language character strip feature is relatively new and displays buttons at the bottom of the screen for the characters in a (chosen) language. This is handy when you occasionally have to key in (snippets) of a language you're not familiar with. More alphabets will be added (we take data from some ConTeXt language relates files).

Templates

It is possible to define (and use) templates. There is a demo file in the distribution called `scite-ctx-templates`. You can put a similar file in your working path or one or two levels up from there. If not found, the default (demo) file will be used. a manu is called up with `ctrl-i`.

A template file is a Lua file and looks like this:

```
-- this is just an example

return {
  xml = {
    {
      name      = "bold",
      nature    = "inline",
      template  = "<b>?</b>",
    },
    {
```

```

        name      = "emphasized",
        nature    = "inline",
        template  = "<em>?</em>",
    },
    {
        name      = "inline",
        nature    = "inline",
        template  = "<m>?</m>",
    },
    {
        name      = "display",
        nature    = "display",
        template  = "<math>?</math>",
    },
    {
        name      = "itemize",
        nature    = "display",
        template  = "<itemize>\n    <item>?</item>\n    <item>?</item>\n    <item>?</it
    },
}

```

In xml sources you can add a line:

```
<?context-directive job ctxtemplate mytemplates.lua ?>
```

The file will be searched for in the current direct and upto two levels higher. When no file is found the T_EX distribution is checked.

The files `scite-ctx-example` and `scite-ctx-context` define the menu commands, like:

```
command.25.$(file.patterns.example)=insert_template
```

Using SCITE

The following keybindings are available in SciTE. Most of this list is taken from the on-line help pages.

| keybinding | meaning (taken from the SciTE help file) |
|----------------------|--|
| Ctrl+Keypad+ | magnify text size |
| Ctrl+Keypad- | reduce text size |
| Ctrl+Keypad/ | restore text size to normal |
| Ctrl+Keypad* | expand or contract a fold point |
| Ctrl+Tab | cycle through recent files |
| Tab | indent block |
| Shift+Tab | dedent block |
| Ctrl+BackSpace | delete to start of word |
| Ctrl+Delete | delete to end of word |
| Ctrl+Shift+BackSpace | delete to start of line |

| | |
|--------------------|--|
| Ctrl+Shift+Delete | delete to end of line |
| Ctrl+Home | go to start of document; Shift extends selection |
| Ctrl+End | go to end of document; Shift extends selection |
| Alt+Home | go to start of display line; Shift extends selection |
| Alt+End | go to end of display line; Shift extends selection |
| Ctrl+F2 | create or delete a bookmark |
| F2 | go to next bookmark |
| Ctrl+F3 | find selection |
| Ctrl+Shift+F3 | find selection backwards |
| Ctrl+Up | scroll up |
| Ctrl+Down | scroll down |
| Ctrl+C | copy selection to buffer |
| Ctrl+V | insert content of buffer |
| Ctrl+X | copy selection to buffer and delete selection |
| Ctrl+L | line cut |
| Ctrl+Shift+T | line copy |
| Ctrl+Shift+L | line delete |
| Ctrl+T | line transpose with previous |
| Ctrl+D | line duplicate |
| Ctrl+K | find matching preprocessor conditional, skipping nested ones |
| Ctrl+Shift+K | select to matching preprocessor conditional |
| Ctrl+J | find matching preprocessor conditional backwards, skipping nested ones |
| Ctrl+Shift+J | select to matching preprocessor conditional backwards |
| Ctrl+[| previous paragraph; Shift extends selection |
| Ctrl+] | next paragraph; Shift extends selection |
| Ctrl+Left | previous word; Shift extends selection |
| Ctrl+Right | next word; Shift extends selection |
| Ctrl+/ | previous word part; Shift extends selection |
| Ctrl+\ | next word part; Shift extends selection |
| F12 / Ctrl+F7 | check (or process) |
| Ctrl+F12 / Ctrl+F7 | process (run) |
| Alt+F12 / Ctrl+F7 | process (run) using the luajit vm (if applicable) |

Affiliation

author Hans Hagen
copyright PRAGMA ADE, Hasselt NL
more info www.pragma-ade.com
www.contextgarden.net
version October 6, 2021