



SciTE
IN CONTEXT MkIV

About SCiTE

This manual is under (re)construction.

For a long time at Pragma ADE we used T_EXedit, an editor we'd written in Modula. It had some project management features and recognized the project structure in ConT_EXt documents. Later we rewrote this to a platform independent reimplementation called T_EXwork written in Perl/Tk (not to be confused with the editor with the plural name).

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 T_EX 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 LuaT_EX 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 ConT_EXt has evolved in a T_EX/Lua hybrid, it made sense to look into this. The result is a couple of lexers that suit T_EX, MetaPost and Lua usage in ConT_EXt 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.¹

In the ConT_EXt (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.

¹ In the process some of the general lexing framework was adapted to suit our demands for speed. We ship these files as well.

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, \TeX primitives, low level \TeX commands, \TeX 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

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.

Next you need to install the lpeg lexers.² These can be fetched from:

code.google.com/p/scintilla

On windows you need to copy the `lexers` subfolder to the `wscite` folder. For Linux the place depends on the distribution and I just copy them in the same path as where the regular properties files live.

For Unix, one can take a precompiled version as well. Here we need to split the set of files into:

² Versions later than 2.11 will not run on Windows 2K. In that case you need to comment the external lexer import.

```

t:\scite\data\scite-context-visual.tex - SCITE
File Edit Search View Tools Options Language Buffers Help
1 texboot.tex | 2 d-buildtool.lua | 3 s-methods.lua | 4 s-jobcontrol.lua | 5 s-sessions.lua | 6 d-runtool.lua | 7 mtr-example.lua | 8 demoticket-mall.xml | 9 mhall.lua | 0 d-dispatchers.lua | 1 scrap.tex | scite-context-readmetex | scite-context-visual.tex
% language=uk
\defineframedtext
[entry]
\starttext
\startchapter[title=Some fancy title]
\startluacode
local entries = { -- there can be more
  { text = "The third entry!" },
  { text = "The fourth entry!" },
}
for i=1,#entries do
  context.startentry(i)
  context.startentry([i].text)
  context.stopentry(i)
end
\stoptluacode
This is just some text to demonstrate the realtime spellchecker
in combination with the embedded lua and metapost lexers and
inline as well as display \ctxlua{context("lua code")}.
\startlinecorrection
\startMPcode
for i=1 upto 100 :
  draw fullcircle scaled (i*mm) ;
endfor ;
\stopMPcode
\stoplinecorrection
\iftrue
\def\crap{some text} % who cares
\else
\def\crap{some crap} % about this
\fi
\blank[2*big]
\crap
\stopchapter
\stoptext
mtx-context
system
| pdfview methods: Acrobat default okular, current method: ac
| total runtime: 2.264sExit code: 0
mtx-context
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
system
> latin/modern fonts are not preloaded
fonts
> language en is active
languages
> 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 mapsfromchar
fontstructure
> fallback modern rm 12pt is loaded
fontstructure
> 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
backendl
> xmp > using file 'c:/data/develop/context/sources/lpdf.pdx
pages
> flushing realpage 1, userpage 1, subpage 1
)
(c:/data/develop/context/txmf/txmf/fonts/opentype/public/lm/lmroman12-r
)
mkiv lua stats > used config file - selfautoparent:txmf-local/web2
mkiv lua stats > used cache path - c:/data/develop/context/txmf
mkiv lua stats > resource resolver - loadtime 0.016 seconds, 1 scans
mkiv lua stats > stored bytecode data - 382 modules, 63 tables, 365 chu
mkiv lua stats > cleaned up reserved nodes - 39 nodes, 9 lists of 427
mkiv lua stats > node memory usage - 2 glue, 2 penalty, 12 attribute
mkiv lua stats > node list callback tasks - 6 unique task lists, 5 instance
mkiv lua stats > used backend - pdf (backend for directly gener
mkiv lua stats > loaded patterns - en::2
mkiv lua stats > callbacks - 2888 direct, 3573 indirect, 638
mkiv lua stats > randomizer - resumed with value 0.682328348
mkiv lua stats > XML preparation time - 0.000 seconds, 0 nodes, 15 'pat
mkiv lua stats > result saved in file - scite-context-visual.pdf
mkiv lua stats > loaded fonts - 33 files: stmary10-arm 1mmono12
mkiv lua stats > fonts load time - 0.361 seconds
mkiv lua stats > metapost processing time - 0.016 seconds, loading: 0.076 s
mkiv lua stats > luaTeX banner - this is luaTeX, version beta-0.
mkiv lua stats > control sequences - 3152 of 82535 (3.8%)
mkiv lua stats > current memory usage - 34.98 (cck:35 MB)
mkiv lua stats > runtime - 1.188 seconds, 1 processed page
mtx-context
system

```

Figure 1 Nested lexers in action.

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

The second path is hard coded in the binary.

If you want to use ConT_EXt, 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). There is a file called `\type{SciteGlobal.properties}`. At the end of that file on windows it is in the path where the Scite binary) you then add a line to the end:

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

You need to restart SciTE in order to see if things work out as expected.

Disabling the external lexer in a recent SciTE is somewhat tricky. In that case the end of that file looks like:

```
imports.exclude=scite-context-external
import *
import scite-context-user
```

In any case you need to make sure that the user file is loaded last.

After this, things should run as expected (given that T_EX runs at the console as well).

Fonts

The configuration file defaults to the DejaVu fonts. These free fonts are part of the ConT_EXt 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
```

An alternative approach

If for some reason you prefer not to mess with property files in the main SciTE path, you can follow a different route and selectively copy files to places.

The following files are needed for the lpeg based lexer:

```
lexers/scite-context-lexer.lua
lexers/scite-context-lexer-tex.lua
lexers/scite-context-lexer-mps.lua
lexers/scite-context-lexer-lua.lua
lexers/scite-context-lexer-cld.lua
lexers/scite-context-lexer-txt.lua
lexers/scite-context-lexer-xml*.lua
lexers/scite-context-lexer-pdf*.lua
```

```
lexers/context/data/scite-context-data-tex.lua
lexers/context/data/scite-context-data-context.lua
lexers/context/data/scite-context-data-interfaces.lua
lexers/context/data/scite-context-data-metapost.lua
lexers/context/data/scite-context-data-metafun.lua
```

```
lexers/themes/scite-context-theme.lua
```

The data files are needed because we cannot access property files from within the lexer. If we could open a file we could use the property files instead.

These files go to the `lexers` subpath in your SciTE installation. Normally this sits in the binary path. The following files provide some extensions. On windows you can copy these files to the path where the SciTE binary lives.

```
scite-ctx.lua
```

Because property files can only be loaded from the same path where the (user) file loads them you need to copy the following files to the same path where the loading is defined:

```
scite-context.properties
```

```
scite-context-internal.properties  
scite-context-external.properties
```

```
scite-pragma.properties
```

```
scite-tex.properties  
scite-metapost.properties
```

```
scite-context-data-tex.properties  
scite-context-data-context.properties  
scite-context-data-interfaces.properties  
scite-context-data-metapost.properties  
scite-context-data-metafun.properties
```

```
scite-ctx.properties  
scite-ctx-context.properties  
scite-ctx-example.properties
```

On Windows these go to:

```
c:/Users/YourName
```

Next you need to add this to:

```
import scite-context  
import scite-context-internal  
import scite-context-external  
import scite-pragma
```

to the file:

```
SciTEUser.properties
```

Of course the pragma import is optional. You can comment either the internal or external variant but there is no reason not to keep them both.

Extensions

Just a quick not 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.

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-xml-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. 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.³

³ 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.

In principle the external lexers can be used with `textadept` which also uses `scintilla`. Actually, support for lpeg lexing originates in `textadept`. Currently `textadept` lacks a couple of features I like about SciTE (for instance it has no realtime logpane) and it's also still changing. At some point the ConT_EXt distribution might ship with files for `textadept` as well.

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.

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
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	<code>all</code>	all commands (preceded by a backslash)
1	<code>tex</code>	\TeX , ϵ - \TeX , pdf \TeX , Omega primitives (and macros)
2	<code>nl</code>	the dutch Con \TeX t interface
3	<code>en</code>	the english Con \TeX t interface
4	<code>de</code>	the german Con \TeX t interface
5	<code>cz</code>	the czech Con \TeX t interface
6	<code>it</code>	the italian Con \TeX t interface
7	<code>ro</code>	the romanian Con \TeX t interface
8	<code>latex</code>	L \AA \TeX (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 Con \TeX t 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 \TeX code and it will come out in a uniform color. But, when set to one, you will get as much colors as a \TeX source. It's a matter of taste what you choose.

The lexer tries to cope with the $\text{T}_{\text{E}}\text{X}$ syntax as good as possible and takes for instance care of the funny $\wedge\wedge$ notation. A special treatment is applied to so called $\backslash\text{if}$'s:

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

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

The MetaPost lexer

The MetaPost lexer is set up slightly different from its $\text{T}_{\text{E}}\text{X}$ counterpart, first of all because MetaPost is more a language than $\text{T}_{\text{E}}\text{X}$. As with the $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$, where documentation is often coded in $\text{T}_{\text{E}}\text{X}$).

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 you global properties files lives, 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

- recognized first lines:

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

This message tells you what extras are available.

Templates

There is an experimental template mechanism. One option is to define templates in a properties file. The property file `scite-ctx-context` contains definitions like:

```
command.25.$(file.patterns.context)=insert_template \
$(ctx.template.list.context)
```

```
ctx.template.list.context=\
  itemize=structure.itemize.context|\
  tabulate=structure.tabulate.context|\
  natural TABLE=structure.TABLE.context|\
  use MP graphic=graphics.usemp.context|\
  reuse MP graphic=graphics.reusemp.context|\
  typeface definition=fonts.typeface.context
```

```
ctx.template.structure.itemize.context=\
\startitemize\n\
\item ?\n\
\item ?\n\
```

```
\item ?\n\
\stopitemize\n
```

The file `scite-ctx-example` defines xml variants:

```
command.25.$(file.patterns.example)=insert_template \
$(ctx.template.list.example)
```

```
ctx.template.list.example=\
  bold=font.bold.example|\
  emphasized=font.emphasized.example|\
  |\
  inline math=math.inline.example|\
  display math=math.display.example|\
  |\
  itemize=structure.itemize.example
```

```
ctx.template.structure.itemize.example=\
<itemize>\n\
<item>?</item>\n\
<item>?</item>\n\
<item>?</item>\n\
</itemize>\n
```

For larger projects it makes sense to keep templates with the project. In one of our projects we have a directory in the path where the project files are kept which holds template files:

```
....../ctx-templates/achtergronden.xml
....../ctx-templates/bewijs.xml
```

One could define a template menu like we did previously:

```
ctx.templatelist.example=\
  achtergronden=mathadore.achtergronden|\
  bewijs=mathadore.bewijs|\
```

```
ctx.template.mathadore.achtergronden.file=smt-achtergronden.xml
ctx.template.mathadore.bewijs.file=smt-bewijs.xml
```

However, when no such menu is defined, we will automatically scan the directory and build the menu without user intervention.

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
<hr/>	
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
<hr/>	
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
<hr/>	

Affiliation

author Hans Hagen
 copyright PRAGMA ADE, Hasselt NL
 more info www.pragma-ade.com
www.contextgarden.net
 version November 11, 2011