SciTE
IN CONTEXT

1 About SCITE

SciTE is a source code editor written by Neil Hodgson. After playing with several editors we decided that this editor was quite configurable and extendible. At Pragma ADE we use TeXedit, an editor we wrote long ago in Modula as well as a platform independent reimplementation called TeXwork written in Perl/Tk. Although these editors provide some functionality not present in SciTE, we've decided to use SciTE because it frees us from maintaining the other ones. In due time we will write some extensions for SciTE to fill in the gaps.

2 Installing SCITE

Installing SciTE is straightforward. We assume that you use 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.

3 The TeX lexer

SciTE provides several so called lexers. A lexer is responsible for highlighting the syntax of your document. The way a TeX 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 TeX 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 that 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, like:

```
cont-nl-scite.properties
cont-en-scite.properties
latex-scite.properties
```

The CONTEXT distribution comes with a file:

```
context.properties
```

as well as the interface specific files. There are two way to make sure that the extra keywords are loaded. One way is to copy the following files to your SCITE properties path.

```
cont-*-scite.properties
latex-scite.properties
```

The `*-scite` files define the keywords, while the later configures SCITE for CONTEXT. If you are no CONTEXT user and want another brand of TEX to be processed, you can tweak the properties, or better: redefine some of them in your `SciTEUser.properties`. For example plain TEX users may like:

```
file.patterns.tex=*.tex;*.sty;
filter.tex=TeX|$(file.patterns.tex)|
lexer.$(file.patterns.tex)=tex
command.compile.$(file.patterns.tex)=
command.build.$(file.patterns.tex)=tex $(FileNameExt)
command.go.$(file.patterns.tex)=gv $(FileName).pdf
```

On the other hand, LATEX users may want:

```
file.patterns.latex=*.tex;*.sty;*.aux;*.toc;*.idx;
filter.latex=LaTeX|$(file.patterns.latex)|
lexer.$(file.patterns.latex)=tex
command.compile.$(file.patterns.latex)=
command.build.$(file.patterns.latex)=pdflatex $(FileNameExt)
command.go.$(file.patterns.latex)=gv $(FileName).pdf
```

But, since you are a ConTEXt user, you will need:

```
file.patterns.context=*.tex;*.tui;*.tuo;*.sty;
filter.context=ConTeXt|$(file.patterns.context)|
lexer.$(file.patterns.context)=tex
command.compile.$(file.patterns.context)=
command.build.$(file.patterns.context)=texexec --pdf $(FileNameExt)
command.go.$(file.patterns.context)=gv $(FileName).pdf
```

The good news is that ConTEXt users don't have to mess around with the properties files! They have:

```
context.properties
```

You can best put this file in the same path as `SciTEUser.properties`. In this user file, you can add the following line:

```
import context
```

Now you have much more commands available (it makes sense to take a look into this file). Beware: this setup assumes that you have the Latin Modern Typewriter font on your system and that your operating system is aware of that. If needed, you can add options (or local changed) after the line that loads `context.properties`.

If you didn't copy the `cont-*.properties` files to the SciTE properties path, you can put them in the same path as `SciTEUser.properties`, in which you then have to add:

```
import cont-cz-scite
import cont-de-scite
import cont-en-scite
import cont-nl-scite
import cont-ro-scite
import cont-xx-scite
import latex-scite
import context
```

It may sound complicated but if you have done it once, you get the picture quite well, and find out that SciTE can easily be tuned to your local preferences.

The ConTEXt related properties files are part of the ConTEXt distribution and can be found in one of the following places:

```
../tex/texmf/context/data
../tex/texmf-local/context/data
```

We generate the interface specific property files automatically from the ConTEXt interface definition files, while the xx file is hand–crafted and contains missing or very special bits and pieces.

Back to the properties in `tex.properties`. You can disable keyword coloring alltogether 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 TEX 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`.

4 The METAPOST lexer

The MetaPost lexer is set up slightly different from its TEX counterpart, first of all because MetaPost is more a language that TEX. As with the TEX 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 TEX 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 TEX, where documentation is often coded in TEX).

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.

5 The XML exporter

The exporter will be descibed as soon as there are styles for processing the XML *code. For the moment we stick to showing the schema.*

```
<?xml version="1.0" ?>

<!--

    filename : scite.rng
    comment  : scite xml export specification
    version  : 1.0 - September 2003
    author   : Hans Hagen
    company  : PRAGMA ADE - Hasselt NL
    url      : www.pragma-ade.com

-->

<grammar xmlns="http://relaxng.org/ns/structure/1.0">
```

```
<start>
    <ref name="document"/>
</start>

<define name="document">
    <element name="document">
        <optional>
            <attribute name="filename"/>
            <attribute name="type"/>
            <attribute name="version"/>
        </optional>
        <optional>
            <ref name="data"/>
        </optional>
        <optional>
            <ref name="text"/>
        </optional>
    </element>
</define>

<define name="data">
    <!-- reserved for future usage -->
    <element name="data">
        <optional>
            <attribute name="comment"/>
        </optional>
        <empty/>
    </element>
</define>

<define name="text">
    <element name="text">
        <optional>
            <attribute name="comment"/>
        </optional>
        <zeroOrMore>
            <ref name="line"/>
        </zeroOrMore>
    </element>
</define>

<define name="line">
    <element name="line">
```

```
        <zeroOrMore>
            <choice>
                <group>
                    <optional>
                        <!-- number of empty lines -->
                        <attribute name="n"/>
                    </optional>
                    <empty/>
                </group>
                <group>
                    <zeroOrMore>
                        <choice>
                            <ref name="space"/>
                            <ref name="tagged"/>
                        </choice>
                    </zeroOrMore>
                </group>
            </choice>
        </zeroOrMore>
    </element>
</define>

<define name="space">
    <element name="s">
        <optional>
            <!-- number of spaces -->
            <attribute name="n"/>
        </optional>
        <empty/>
    </element>
</define>

<define name="tagged">
    <element name="t">
        <optional>
            <!-- style number -->
            <attribute name="n"/>
        </optional>
        <zeroOrMore>
            <choice>
                <text/>
                <ref name="space"/>
                <element name="l">
```

```
                                <!-- less token -->
                                <empty/>
                            </element>
                            <element name="g">
                                <!-- greater token -->
                                <empty/>
                            </element>
                            <element name="a">
                                <!-- ampersand token -->
                                <empty/>
                            </element>
                            <element name="h">
                                <!-- hash token -->
                                <empty/>
                            </element>
                        </choice>
                    </zeroOrMore>
                </element>
            </define>

        </grammar>
```

6 Using ConTEXt

As soon as they are stable, we will describe the ConTEXt *specific menu commands and key-bindings here.*

7 Using SCITE

Here we will put a couple of tables with key bindings and alike.

8 Affiliation

author	Hans Hagen
copyright	PRAGMA ADE, Hasselt NL
more info	`www.pragma-ade.com`