Idris Samawi Hamid
Luigi Scarso

Notepad++

FOR CONTEXT MKIV

Version 0.98

Table of Contents

1 Background

1.1 Motivation

A continuing desideratum for CONTEXT is a user-friendly writing and editing environment, where the range of application of the category "user-friendly" includes especially non-experts in programming or software development. The lack of such an environment is one factor that inhibits the wider use of CONTEXT. Despite its immense power, precision, and flexibility: At present it is not generally feasible for instructors and researchers in, e.g., the humanities to assign the use of CONTEXT to students, or to use it to collaborate on projects.

The first author of this manual, Idris Samawi Hamid, is a professor who has felt the acuteness of this lacuna. In the course of an ongoing effort to address it, in 2017 a project to develop a set of utilities for the Windows editor Notepad++, including a dedicated CONTEXT *lexer* plugin, was launched.[1] The software development plan was developed and supervised by Hamid, who also wrote the color-scheme and themes. The initial C++ code and Python scripts were written by Jason Wu (a research assistant at Colorado State University); currently the code and scripts are being written by and maintained with coauthor Luigi Scarso. This manual documents a major release of that project: For the moment we call the project, simply, **Notepad++ for CONTEXT MkIV**: Lexer and Command Utilities for Editing ConTeXt Documents.

1.2 History

Prior to his move to CONTEXT, Hamid was using the shareware editor WinEdt. At that time WinEdt was (and surely still is) a very polished environment for writing and processing documents written in TEX. However, configuring WinEdt for CONTEXT was critically impeded, due in major part to the fact that much of its graphical user interface was hardcoded for a certain famous document preparation system. Around the same time, lexers and tools were being developed for SciTE, which eventually became the semi-official text-editor for CONTEXT. Despite its CONTEXT-friendly tools, Hamid continued to miss many of the configuration and interface options of WinEdt that made editing and processing TEX documents so efficient and user-friendly for non-programmers. After trying virtually every available option – explicitly TEX-friendly or other – he finally settled upon Notepad++.[2] Its look, feel, and extensive configuration options allowed Hamid to quickly achieve a setup analogous to WinEdt. A few characteristics of WinEdt were still missed; on the other hand, Notepad++ brought to the table other features missing in WinEdt; these helped to make the transition worth it. For example, Notepad++ supports global bidirectional text editing essential for the Arabic script – WinEdt had no such support.[3]

Eventually, over a decade ago, a basic package for Notepad++ was released to the CONTEXT community by Hamid. It consisted of a number of configuration files, including, among other things,

---

[1] For explanation of what a lexer is, see Section 2.3.1.

[2] First author Hamid was introduced to Notepad++ by Hans Hagen, the founder and primary developer of CONTEXT. Hagen also developed the TEX and MetaPost interface used by **SciTE in CONTEXT MkIV**:
http://www.pragma-ade.nl/general/manuals/scite-context-readme.pdf.

[3] Apparently WinEdt finally implemented support for bidirectional text editing in 2016. The first author also considered Emacs, which finally released a version with bidirectional support in 2012 (Version 24.1):
http://lists.gnu.org/archive/html/info-gnu-emacs/2012-06/msg00000.html.
On the other hand, editors such as Emacs (or Vim) are far too esoteric for the target audience of this project. It should also be mentioned that, in the Spring of 2017, the first author once again made an inventory of the available options and paradigms for writing and editing content, including new editors such as Atom and Sublime Text; none could replace Notepad++ for the target purposes and audience.

- a UDL (User-Defined Language) file for code highlighting of different classes of TEX-commands and other keywords;

- an autocompletion "API" file; and

- some console scripts, many of which appear under the submenu item `Macros`. These provided, among other things, a functionality largely identical to that provided by the corresponding SciTE scripts for CONTEXT – found under the submenmenu item `Tools`.[4]

Although remarkably versatile, the UDL system was still too restrictive. Other Notepad++ mechanisms, such as autocompletion of control sequences, were not designed with TEX-type languages in mind;this resulted in certain limitations or annoyances. Among other issues: As CONTEXT MkIV has continued to develop in the direction of a pure markup language, its syntax has

- become considerably more verbose; and

- demanded a mechanism for easy tagging of text with, e.g., braces or a set of `\start|\stop<command>` sequences.

Mere autocompletion of commands was no longer sufficient for efficient content writing and editing. Fortunately WebEdit, a Notepad++ plugin designed for XML-type tagging and related function completion, came to the rescue. Unfortunately it also had certain limitations which inhibited a satisfactory solution (such as a limit to the number of markup tags and no way to organize them by submenus).

In the wake of these and other limitations: What we needed was a dedicated CONTEXT lexer and plugin to assist content writing and editing. In combination with other mechanisms and plugins, the result would be a complete Notepad++ system for writing, editing, and processing CONTEXT documents. Hence **Notepad++ for CONTEXT MkIV**.

2 Introduction to NOTEPAD++

2.1 Features

Developed by Don Ho, Notepad++ is a very popular text editor for the Windows platform. Although geared towards programmers and web designers, it has a number of features that make it exceptionally appropriate for non-programmers. Notepad++ features, among other things

- A user-friendly configuration system, via graphical dialogs and settings saved to editable XML files;

- both multiple and single-document splitting;

- translation of its display interface into multiple languages;

- the toolset TextFX, which provides a plethora of useful functions that would normally involve script-writing on the part of the user;

---

[4] That package, now obsolete, remains available here: http://wiki.contextgarden.net/File:Npp_ConTeXt-Uni.zip.

- a plugin system and a vast catalogue of over 100 available plugins which immensely extend the capabilities of Notepad++ in a user-friendly manner; and

- the User-Defined Language (UDL) system, which allows the user to easily define folding rules and syntax highlighting for a coding language that does not already come with Notepad++. It is especially useful for simple scripting languages or text-file formats.[5]

2.2 NOTEPAD++ and SCITE

As mentioned earlier, CONTEXT already comes with SciTE. Both SciTE and Notepad++ are based on the same text-editing component, Scintilla. Thus a user switching between the two editors can expect a similar typing and editing experience. A fundamental difference between the two is that Notepad++'s preferences, thematic styles, and shortcuts are all extensively configurable via a system of menus and dialogs, whose style is mostly common to mainstream programs that use a GUI. For non-programmers and the like, this is more comfortable than, e.g., editing the `.properties` files used by SciTE.

One of the most important features of Notepad++ is its support for global bidirectional editing. Some background: Unfortunately Scintilla never implemented bidirectional editing, and the developer of Scintilla apparently has little interest in pursuing it. Visually, basic mixed right-to-left (RTL) and left-to-right (LTR) text *may* look normal, but selection of text whose direction is opposite to that of the global direction of the editor will *generally* not copy and paste correctly. For SciTE the global direction is, naturally, LTR; hence RTL will *generally* not copy and paste correctly.[6] Notepad++ provides a mechanism that mirrors, i.e., flips Scintilla behavior so that it can be used for RTL editing, except that LTR will now generally not copy and paste correctly. So for proper RTL or LTR editing one must switch the global direction to match the immediately desired editing direction.

SciTE in CONTEXT features a set of commonly used scripts that may be found under the Tools menu. In Notepad++ for CONTEXT a similar set of tools – with identical shortcuts wherever convenient – may be found under the Macro menu.

The core of Notepad++ is explicitly designed for speed. On Windows, Notepad++ generally starts up fast, even faster than SciTE. A few plugins will slow Notepad++ down, however.

2.3 Lexers and NOTEPAD++

2.3.1 Lexers: Internal, User-Defined, and External

A lexer is a program (or subroutine of a program) that performs *lexical analysis* of a stream of text (such as a TEX or Lua file). This means that it analyzes the entire text stream into discrete strings, each of which belongs to some class with a specific meaning. For example: In our CONTEXT lexer `\emph` is a 5-character string that belongs to the class STYLE. Given a lexer class, each of its members is assigned a particular highlighting convention, such as a specific color, typographical appearance, or font. Notepad++ ships with highlighting and theme support for over 50 code languages (via native *internal lexers*); the UDL system allows a user (even with little-to-no coding experience) to comfortably configure and add more.[7] For maximum flexibility and control, Notepad++ also supports *external*

---

[5] For example, one may edit tables in an OpenType font editor, then save those tables to a text file with an associated syntax. One may then choose to work with the text file instead of the Graphical User Interface (GUI). Notepad++ for CONTEXT MkIV also includes a basic UDL for bibTEX.

[6] The use of 'may' and 'generally' are meant to indicate that there may be some important exceptions and subtleties: See Section 7.

[7] To configure a private UDL: From the Notepad++ menu, go to

```
Language ▸ Define your language…
```

lexers, development of which requires some C++ programming skill: Each external lexer will appear under the Language menu and in the associated dialogs. An external lexer can add support for a previously unsupported language, or it can be used to provide an alternative to a currently supported language. For example, one can use the Lua lexer and highlighting that comes with Notepad++, or one can download the external lexer Gmod Lua, then configure that to be the default lexer for the Lua language. An external lexer can also be augmented by other features, which will then appear under the Plugins submenu.

2.3.2 Recommended Plugins

For use as a complete environment for writing and editing documents, a number of plugins complement the Notepad++ for CONTEXT system. The following are highly recommended:

- **NppExec**

  This is the console, and is an integral component of Notepad++ for CONTEXT; see Section 5. Although one can have Notepad++ launch the command prompt or other console of one's choosing, NppExec is also needed to show a set of select scripts under the Macro menu. A standard installation usually gives the option of installing the console. Or one can use Plugin Manager (see below).

- **Explorer**

  Notepad++ can launch the normal Windows Explorer. But there is also the Explorer plugin which can be docked inside of the editor or detached; it has some useful features such as a filter which allows one to view only files of a selected type.

- **DSpellCheck**

  This spell checker works well, although it could be improved. Currently it doesn't make exceptions for words that begin with a backslash; this means that most TeX control sequences are treated as misspelled. We hope to see this fixed in the short term.

- **Compare**

  This is a plugin for comparing files; it launches a double-pane view and a dockable applet.

- **XBrackets Lite**

  This plugin provides automatic completion of different types of brackets and is configurable. Notepad++ comes with some native facility for bracket control, but XBrackets Lite is more useful.

- **Plugin Manager**

  This plugin maintains a list of i) all registered plugins, ii) installed plugins, and iii) installed plugins for which updates are available. One can choose to install, update, or delete any given plugin as desired.

  In addition to the recommended set above, there are many other plugins available, e.g., NppDocShare for collaborative editing, MarkdownViewer++ for previewing markdown output, and XMLTools. With a little research and some tweaking, it is not hard to turn Notepad++ into a writing, editing, and development environment that will suit most of one's needs.

2.4 Installing NOTEPAD++

2.4.1 Basic Installation

A basic Notepad++ installation may be setup via the standard installer or as a standalone package. Both are available from here:

```
https://notepad-plus-plus.org/download/
```

For the average user, it is best to use the standard Notepad++ installer. Currently Notepad++ for CONTEXT supports (i.e., has been tested to work under) Notepad++ Version 7.5.4 (32-bit) and NppExec Version 0.5.3.[8] 64-bit support is planned for the near future; see also Section 3.2. There are two directories the user should make note of:

```
C:\Program Files (x86)\Notepad++
C:\Users\<username>\AppData\Roaming\Notepad++
```

The core files of Notepad++ for CONTEXT will be installed in these two directories.

Some expert users may prefer to install a standalone Notepad++ as well as all associated local files into this single directory:

```
/ConTeXt/data/notepad++
```

This is analogous to the standalone directory for SciTE:

```
/ConTeXt/data/wscite
```

2.4.2 Recommended Settings

- Notepad++ does not support automatic *hard wrap* of lines that surpass a certain length (as does WinEdt). That is, Notepad++ doesn't insert new-line breaks when a line reaches the maximum width of the editing window (or some user-specified width). Notepad++ does support *soft wrap* of lines: This means that a line longer than the editing-window length will appear as though it is split into multiple lines, but is treated by Notepad++ as a single line. Soft wrap can be toggled on and off:

  ```
  View — Word wrap (Ctrl+Alt+W)
  ```

  Except for special cases such as the editing of wide tables, it is recommended to keep Word Wrap on.

  One may manually break long-editor or soft-wrap lines into distinct paragraphs of multiple editor lines, each with a select maximum width of characters, by selecting the text and choosing

  ```
  TextFX — Edit — ReWrap Text to (Clipboard or 72) width (Ctrl+Alt+Shift+INS)
  ```

  Eventually the first author came to the conclusion that automatic text breaking is not as helpful as first thought. For example, a search for a string of text longer than one word will miss instances if a line break occurs within that string.

- Based on extensive writing and editing work with Notepad++, the first author has found it helpful, for purposes of organization and productivity, to maintain the following setting:

  ```
  Settings — Preferences — Editing — Line Wrap — Indent
  ```

  Indented wrapping happens with respect to the beginning of the line. So if a given line is tabbed or padded with spaces prior to its first character, any soft-wrapped text will always be indented with

---

[8] It has been noticed that, when NppExec is updated to a newer version, the shortcuts in the Macro submenu sometimes fall out of sync and have to be reset. See also Section 5.3.

respect to the column where the line begins. This allows for, e.g., hierarchical organization or nesting of paragraph levels if desired.

- Notepad++ has a toolbar, but its style is quite dated. The first author has found it more productive to leave it turned off:

```
Settings — Preferences — General — Toolbar — Hide
```

3 The NOTEPAD++ for CONT$_E$XT Package

3.1 Components

Notepad++ for ConT$_E$Xt is organized as follows:

```
/Npp-for-ConTeXt/doc
/Npp-for-ConTeXt/Program Files (x86)
/Npp-for-ConTeXt/Roaming
/Npp-for-ConTeXt/scripts

/Npp-for-ConTeXt/Program Files (x86)/Notepad++

/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins
/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/ConTeXt.dll

/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/APIs
/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/APIs/context.xml
/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/APIs/context-user.xml

/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/Config
/Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/Config/ConTeXt.xml

/Npp-for-ConTeXt/Roaming/Notepad++
/Npp-for-ConTeXt/Roaming/Notepad++/contextMenu.xml
/Npp-for-ConTeXt/Roaming/Notepad++/shortcuts.xml
/Npp-for-ConTeXt/Roaming/Notepad++/userDefineLang.xml
/Npp-for-ConTeXt/Roaming/Notepad++/stylers.xml

/Npp-for-ConTeXt/Roaming/Notepad++/plugins

/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/config.xml
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/ConTeXt.ini
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/NppExec.ini
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/npes_saved.txt

/Npp-for-ConTeXt/doc
/Npp-for-ConTeXt/doc/npp-context-manual.pdf
/Npp-for-ConTeXt/doc/npp-context-manual.tex
/Npp-for-ConTeXt/doc/README.md
```

```
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/themes
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/themes/Silver Twilight Hi.xml
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/themes/Silver Twilight Lo.xml

/Npp-for-ConTeXt/scripts/command_primitives_api_new.py
/Npp-for-ConTeXt/scripts/update-ConTeXt.py
```

What follows is a brief description of each component of this system:

1. **CONTEXT Lexer and Plugin**

   `ConTeXt.dll` is the heart of the system. It manages the classes specified for content highlighting, autocompletion and calltips, as well as the content-markup and templates system.

   ```
   /Npp-for-ConTeXt/Program Files/Notepad++/plugins/ConTeXt.dll
   ```

2. **Initialize Plugin**

   `ConTeXt.ini` allows the user to add, remove, configure, and organize commands for content markup into menus and submenus, as well as to specify a shortcut that can be replaced by a template in running text.

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/ConTeXt.ini
   ```

3. **Right-Click Menu**

   Notepad++ features a right-click menu mechanism, whose settings are managed via the configuration file `contextMenu.xml`. The full set of markup menus in the plugin can be added to this file, then edited manually as desired. Note that, despite appearances, the name `contextMenu.xml` has nothing to do with CONTEXT; it is native to Notepad++.

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/contextMenu.xml
   ```

4. **Autocompletion API**

   The so-called "API" `context.xml` features (what aims to be) a complete list of official CONTEXT commands, organized alphabetically for autocompletion purposes.[9] For a subset of this list, each is also tagged with information about usage; when typed and followed by a left bracket '[', this information will appear as a *calltip* (also called a *tooltip*).

   If autocompletion is desired for user-defined macros, then they should be placed in `context-user.xml`. Its structure follows that of `context.xml`, and and one may configure calltips for user macros as well. See Section 6.3.

   ```
   /Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/APIs/context.xml
   /Npp-for-ConTeXt/Program Files (x86)/Notepad++/plugins/APIs/context-user.xml
   ```

5. **Content-Highlighting Classes**

   `ConTeXt.xml` includes the same list of official CONTEXT commands, this time organized into semantic *classes*. These and other classes are configured for content highlighting through Notepad++'s Style Configurator.

   ---

   [9] The list of CONTEXT commands is currently generated from the CONTEXT sources by a Python script; see Heading 11 below. There is still a minor residue of commands that are missed in the sources for the list, and thus by the script as well. We hope to see that gap closed in the near future.

```
/Npp-for-ConTeXt/Program Files/Notepad++/plugins/Config/ConTeXt.xml
```

6. **Highlighting: Silver Twilight Hi and Silver Twilight Lo**

   Two general themes for content highlighting have been developed especially for this project: the first and default theme (Hi) is light, the second (Lo) is dark. Each may be accessed and tweaked via Style Configurator, or copied to a new name and modified to make a new theme. See Section 4.4.

   Silver Twilight themes apply to one degree or other throughout the default languages that come with Notepad++ (there remains some work to do in that respect).

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/plugins/themes/Silver Twilight Hi.xml
   /Npp-for-ConTeXt/Roaming/Notepad++/plugins/themes/Silver Twilight Lo.xml
   ```

   The file `stylers.xml` is optional: It is identical to Silver Twilight Hi, and is a starting point for the user to make one's own changes to the theme. This file will appear in Style Configurator labeled as `Default (stylers.xml)`.

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/stylers.xml
   ```

7. **NppExec Scripts**

   A number of scripts commonly used for CONTEXT productivity are saved in `npes_saved.txt`. Normally one configures these through the dialog that appears when the console is executed (by typing `F6`).

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/npes_saved.txt
   ```

8. **Initialize NppExec and Configure Macro Menu**

   Default settings for the appearance of NppExec, consistent with the Silver Twilight themes, are saved in `NppExec.ini`. This file also maintains a list of console scripts that are to appear under the Macro menu; this is normally edited via the NppExec Advanced Options dialog.

   ```
   /Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/NppExec.ini
   ```

9. **Users Manual**

   The user's manual (this document) and its source are named, respectively, `npp-context-manual.pdf` and `npp-context-manual.tex`.

   ```
   /Npp-for-ConTeXt/doc/npp-context-manual.pdf
   /Npp-for-ConTeXt/doc/npp-context-manual.tex
   ```

10. **Shortcuts**

    Virtually all menu commands can be assigned a keyboard shortcut, and each shortcut is configurable. A basic system of shortcuts, consistent across a number of recommended or useful plugins, is provided by `shortcuts.xml`.

    Notepad++ has a `Run…` command that allows the user to execute a script that will call an external programs; that script can be saved. Saved scripts appear under the `Run` menu; these are also saved in `shortcuts.xml`. The user will almost certainly want to edit the `Run` menu at some point.

    ```
    /Npp-for-ConTeXt/Roaming/Notepad++/shortcuts.xml
    ```

11. **Python Scripts**

New versions of CONTEXT are released often; the addition of new control sequences (= commands) or an update of the parameters of some already existing command is not uncommon. For those who update often: The lists of official commands in `ConTeXt.xml` and `context.xml` are generated from the sources via the Python script `command_primitives_api_new.py`; `update-ConTeXt.py` makes sure that local changes to the `ConTeXt.xml` configuration are saved and not overridden.

```
/Npp-for-ConTeXt/scripts/command_primitives_api_new.py
/Npp-for-ConTeXt/scripts/update-ConTeXt.py
```

For more detail about these scripts and their usage, see

```
/Npp-for-ConTeXt/doc/README.md
```

12. **BibTEX**

Finally, there is a UDL (user-defined language) file configured for content highlighting of `.bib` files; it is consistent with the Silver Twilight themes. This file may be considered optional. Any additional UDL's defined or imported by the user will also be saved to the file `userDefineLang.xml`.

```
/Npp-for-ConTeXt/Roaming/Notepad++/userDefineLang.xml
```

13. **SumatraPDF**

Notepad++ uses as the default pdf viewer. It's fundamental advantage over Adobe Reader or Acrobat is that it does not lock the pdf file. This means that one can continue to view the output `.pdf` while LuaTEX processes its source `.tex` file. See Section 5.2 for examples of its use in console scripts.[10]

If SyncTEX is turned on: Given a PDF file, SumatraPDF has the ability to i) read the SyncTEX file associated with that file, if any; ii) open the source TEX file in Notepad++; and iii) go to the line specified via double-clicking a position in the PDF file that carries some synchronization info.[11]

In the relevant console scripts – see Section 5.2 – SumatraPDF is invoked via a batch file, `sumatra.bat`.

```
/Npp-for-ConTeXt/scripts/sumatra.bat
```

14. **GUI and Editor Settings**

Configuration options for Notepad++ are set through

```
Settings — Preferences
```

User preferences are saved in `config.xml`. The (optional) version that ships with Notepad++ for CONTEXT contains a recommended set of configuration options.

```
/Npp-for-ConTeXt/Roaming/Notepad++/plugins/config/config.xml
```

3.2 Installation

In a future version, this plugin package should become available through the official Notepad++ Plugin Manager. Currently it must be installed manually. The Notepad++ for CONTEXT package, `Npp-for-ConTeXt.zip`, may be obtained here:

---

[10] There are alternatives to SumatraPDF; an example is Okular (which requires KDE for Windows).

[11] Unfortunately there are too many structural elements of CONTEXT whose text do not carry the needed synchronization info; SyncTEX works best with normal paragraphed text.

```
https://github.com/luigiScarso/context-npp
https://github.com/luigiScarso/context-npp/blob/master/install/Npp-for-ConTeXt.zip
```

Once again, the directory structure is as follows:

```
/Npp-for-ConTeXt/doc
/Npp-for-ConTeXt/Program Files (x86)/Notepad++
/Npp-for-ConTeXt/Roaming/Notepad++
/Npp-for-ConTeXt/scripts
```

Be sure that no instances of Notepad++ are running. Copy `/doc` and `/scripts` to

```
C:<ConTeXt root directory>\data\notepad++
```

Copy `/Program Files (x86)/Notepad++` onto

```
C:\Program Files (x86)\Notepad++
```

Copy `/Roaming/Notepad++` onto[12]

```
C:\Users\<username>\AppData\Roaming\Notepad++
```

From `/ConTeXt/data/notepad++/scripts`, move `sumatra.bat` into your CONTEXT root directory:

```
C:\<ConTeXt root directory>
```

For a standalone installation in `C:<ConTeXt root directory>\data\notepad++`, first install a standalone Notepad++; see Section 2.4. Basically you will merge

```
/Roaming/Notepad++
/Program Files (x86)/Notepad++
```

into this directory:

```
C:<ConTeXt root directory>\data\notepad++
```

Now open Notepad++: If all has gone well, you will see the following submenus:

```
Language — ConTeXt
Language — BibTeX <under "Define your own language">
Plugins  — ConTeXt
```

You should also see CONTEXT-related commands under the Macro menu, e.g.,

```
End TeX Control Sequence
<contd.>
Scratch ConTeXt File
Check ConTeXt File
Process ConTeXt File
<contd.>
```

---

[12] For older versions of Windows such as Vista, the local configuration folder (`%APPDATA%`) may have a different name. If necessary, type '`%APPDATA%`' into the address field of Windows Explorer to determine the official location of the local Notepad++ directory.

4 Highlighting and Themes

4.1 Solarized++: Screen Contrast and Color Scheme

Writing and editing content via a digital display for many hours on end can cause severe strain on the eyes. One way to ameliorate this is to use a comfortable color scheme for one's editor. The individual colors provide the building blocks for themes and for distinguishing the various types of written content involved in one's editing.

Color-scheme preferences will naturally differ from person to person to one degree or other. However, a couple of general rules appear to stand out for long periods of editing:

- Maintain a *medium-to-high* balance of contrast between the main text and background tones; i.e., strong contrast, but not too high.

- Choose *soft* colors to distinguish classes of content; not too bright, not too dim.

One of the most thought out and successful color schemes is Solarized, by Ethan Schoonover.[13] It features two series: a series of eight *background tones* and another series of eight *accent colors*. As excellent as it is, the first author found the Solarized background tones to exude something of a murky or "swampy" aesthetic. The light theme is too bright for continuous full-screen use (see Section 4.4). The content colors are more successful: They are both soft and distinct, although Solarized green is perhaps better called *yellowgreen*.

In Notepad++ for CONTEXT the first author has developed a modification of Solarized; the resultant color scheme is called, perhaps appropriately, **Solarized++**. There are nine background tones and ten accent colors. The background tones are entirely different from the original Solarized; the accent colors are largely the same. However, Solarized green has been replaced with Solarized++ green, Solarized green has become Solarized++ yellowgreen, and an additional color, Solarized++ maroon, has been added. See Figure 1.

In addition, Solarized++ currently features a series of five supplementary *anti-base* tones for purposes of contrast when needed. As the name suggests, these five are meant to complement the base tones; see Figure 2.

4.2 On Syntax and Semantic Highlighting

Syntax highlighting has been shown to have a positive impact on the comprehension of computer programs.[14] In the experience of the authors, the same is true for highlighting of structural and stylistic markup in CONTEXT. There is a (perhaps pedantic) difference: Although the high-level, *basic* user interface of CONTEXT is expressed in terms of control sequences that take the form of TEX commands, TEX per se closely exemplifies the paradigm of a *programming* language in a strict sense; whereas the user interface of CONTEXT has developed towards exemplifying the paradigm of a *markup* language. Technically speaking, even if one writes a basic CONTEXT document with pure markup and no deeper commands, one still has to run that document through a compiler which will interpret the input and convert it to some output, normally a PDF document. We might describe the basic CONTEXT interface as a hybrid: markup language in appearance and programming language in reality.

Markup is focused more on meaning, i.e., *semantics*, and less on grammar, i.e., *syntax*. Programming involves syntax to a high degree, and also semantics. Because syntax is often subtle and slippery

---

[13] See http://ethanschoonover.com/solarized.
[14] See, e.g., Sarkar (2015).

Name	Hex	Sample	Name	Hex	Sample
Accent Colors			**Base Tones**		
Yellow	#B58900		Base04	#1E2D2E	
Orange	#CB4B16		Base03	#324140	
Red	#DC322F		Base02	#475652	
Magenta	#D33682		Base01	#5C6B64	
Violet	#6C71C4		Base0	#718076	
Blue	#268BD2		Base1	#899589	
Cyan	#2AA198		Base2	#A2AA9D	
Green	#399900		Base3	#BABFB1	
Maroon	#A12A33		Base4	#D3D5C5	
Yellowgreen	#859900				

Figure 1   Solarized++: Base Tones and Accent Colors

Anti-Base Tones		
Antibase0	#73606D	
Antibase1	#897781	
Antibase2	#9F8E96	
Antibase3	#B5A5AA	
Antibase4	#CCBDBF	

Figure 2   Solarized++:
Anti-Base Tones

for the programmer, code highlighting for programming languages generally takes the form of *syntax highlighting*, so much so that 'code highlighting' and 'syntax highlighting' are often treated as synonymous. In recent years, some coders have begun to emphasize a distinction between syntax highlighting and *semantic highlighting*.[15] Because the interpreting of structural and stylistic markup pertains much more to matters of meaning than to grammar, highlighting of CONTEXT code is best contextualized in terms of semantic highlighting. Of course, there is syntax to CONTEXT as well: The different mechanisms between the earlier `Table` and the now standard `TABLE` environments (for typesetting of tabular data) exhibit stark differences in syntax. Considering possible models and implementations of

---

[15] For detailed discussion of the distinction between syntax and semantic highlighting, see
Semantic Code Highlighting (https://goo.gl/dB5d9u – accessed February 28, 2018); and
C++ IDE Evolution: From Syntax Highlighting to Semantic Highlighting (https://goo.gl/jTPwD1 – accessed February 28, 2018).

code highlighting specific to the clarification of CONTEXT syntax is a matter for future research.[16]

4.3 Highlighting and the CONTEXT Lexer

Settings for semantic highlighting of CONTEXT keywords in Notepad++ are saved in the configuration file `ConTeXt.xml`, mentioned earlier. In particular, there are 14 classes of keywords; members of each class are given a specific color; these may be viewed (and edited) in Style Configurator, under `Language:ConTeXt`.[17]

Following is a brief description of each keyword class supported in the CONTEXT lexer. See also Figure 3.
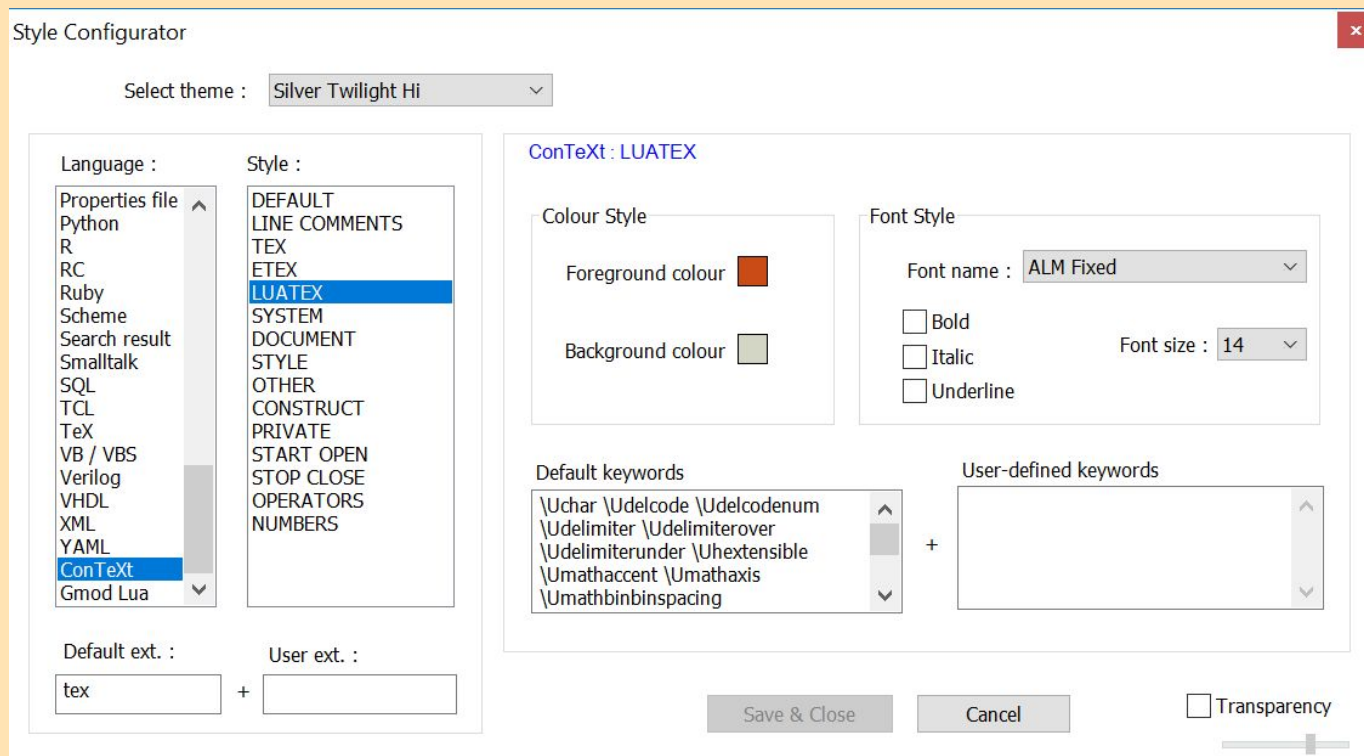
Figure 3 CONTEXT Lexer and Notepad++ Style Configurator

1. **DEFAULT**

   This is the default keyword class, applied to strings which involve no other semantics. Normal text will generally belong to the default class. As default, there are no other keywords specified for this class.

2. **LINE COMMENTS**

   This class includes the percent sign and all text on the same line that comes after it. Of general scope, there are no keywords specified for this class.

---

[16] Technical note: ConTeXt resembles SGML – derivatives include XML and HTML – in that it can used to define new semantics. For example, one can create a verbatim environment, which simulates typing or editing. Ideally, then, the lexer should be coded to manage a user-defined semantics, defined itself at runtime. But this is not enough because, furthermore, the semantics of a macro can temporarily change. For example:

```
\begingroup \let\oldvbox\vbox \let\vbox\relax .\vbox{..} \endgroup
```

Here `\vbox` means nothing inside the group so semantics are context sensitive.

[17] Because the CONTEXT language comes in the form of a lexer plugin, it will generally appear near the bottom of the language list on the left side of the dialog, after the natively supported languages, and along with other lexer plugins, if installed. Some classes allow the user to add one's own keywords to the class as well

3. **TEX ETEX**

   Primitive commands of TeX and ε-TeX are treated as one class.

   Allows user-defined keywords: **No**

4. **LUATEX**

   LuaTeX has its own class. Although not often, new primitives can appear, and LuaTeXperts can define their own.

   Allows user-defined keywords: **Yes**

5. **SYSTEM**

   This is an official CONTEXT keyword class. It includes system-level commands, i.e., those which are not meant for general typesetting and which the average user will rarely or never see.

   Allows user-defined keywords: **Yes**

6. **DOCUMENT**

   This is an official CONTEXT keyword class. It includes commands that are generally meant to *produce* a stream of text within a document.

   Allows user-defined keywords: **Yes**

7. **STYLE**

   This is an official CONTEXT keyword class. It includes commands that are generally meant to *style* a stream of text within a document.

   Allows user-defined keywords: **Yes**

8. **CHARDEF (formerly OTHER)**

   This is an official CONTEXT keyword class. It consists of commands that translate to certain Unicode characters that are needed but normally inconvenient to typeset directly.

   Allows user-defined keywords: **Yes**

9. **CONSTRUCT**

   This class includes keywords used to constitute prefixes to other keywords, such as `\place` and `\set`. The prefix and any immediately following string connected to that prefix is treated as a keyword. Words in other classes that already contain one of these prefixes are not effected.

   Allows user-defined keywords: **Yes**

10. **PRIVATE**

    These are for keywords defined by the user. A few *highlight* definitions are given for illustration, and the user can add more.

    Allows user-defined keywords: **Yes**

11. **START OPEN**

    These are opening commands that begin a folding environment; each must have an associated closing keyword in the STOP CLOSE class. A small symbol will appear in the margin next to the opening keyword, with a bright line leading to the closing symbol.

    Allows user-defined keywords: **Yes**

12. **STOP CLOSE**

    These are closing commands that end a folding environment; each must have an associated opening keyword in the START OPEN class.

    Allows user-defined keywords: **Yes**

13. **OPERATORS**

    This class includes punctuation and related symbols.

    Allows user-defined keywords: **No**

14. **NUMBERS**

    This includes numerals and related symbols.

    Allows user-defined keywords: **No**

4.4 Silver Twilight Hi and Silver Twilight Lo

The Solarized++ color scheme and lexer keyword classes for semantic highlighting together constitute the components which go into Silver Twilight. Silver Twilight consists of two closely related themes which are designed for writing and editing for long hours, usually on a monitor in portrait mode. Portrait mode is generally more efficient than landscape mode for writing and editing productivity: It allows for the editor to comfortably fill most or all of the width of the screen, depending on the monitor resolution. The maximum width of the editor window should correspond to a maximum of between 77 to 105 characters per line within the typing area of the editor (average 91), depending on the zoom level and the choice of fixed-width font.[18] This leaves a generous full length of the rest of the screen available for writing or editing with a minimum need for scrolling.

On the other hand, staring at such a large area of writing space for long periods needs to be ameliorated, as discussed earlier. The Silver Twilight themes are designed to address and meet that need. Silver Twilight Hi is a light theme, perhaps best for daylight hours, but works for nighttime as well. Silver Twilight Lo is a dark theme, perhaps best for nighttime, but works for daylight as well. At the time of writing this manual, the first author is somewhat more satisfied with Silver Twilight Hi than with Silver Twilight Lo; your mileage may vary. Both could benefit from improvement in future versions; suggestions from the CONTEXT community are welcome!

---

[18] Typographers recommend a length of 45 to 75 characters per line (average 60); see Bringhurst (2008). However, writing and editing in a fixed-width font is not the same as reading the final output in a book or on a web page. Restricting the typing area of an editor to 45 to 75 characters per line feels forced. That said, Notepad++ can display a vertical edge and the user can choose a value for "number of columns", i.e., number of characters per line (we set it to 91). It would be nice if Notepad++ could automatically soft wrap (i.e., wrap without line breaks – see Section 2.4.2) long lines at the vertical edge instead of at the border of the edge of the typing area.

Hi					
Style	**Color**	**Sample**	**Style**	**Color**	**Sample**
Global Override Background (B)	Base4		Global Override Foreground (F)	Base04	
Line Number Margin B	Base3		Line Number Margin F	Antibase0	
Current Line Background	Base3		Comment	Base0	
Inactive Tabs	Base2		Smart Highlighting	Cyan	
Selected Text Color	Base1		Fold Active	Cyan (NPP)	
Fold Margin B	Antibase4		Fold Margin F	Base0	

Figure 4   Global Style: Silver Twilight Hi

Hi					
Style	**Color**	**Sample**	**Style**	**Color**	**Sample**
Global Override Background (B)	Base04		Global Override Foreground (F)	Base4	
Line Number Margin B	Base03		Line Number Margin F	Antibase0	
Current Line Background	Base03		Comment	Base0	
Inactive Tabs	Base02		Smart Highlighting	Cyan	
Selected Text Color	Base01		Fold Active	Cyan (NPP)	
Fold Margin B	Antibase4		Fold Margin F	Base0	

Figure 5   Global Style: Silver Twilight Lo

In Notepad++ Style Configurator, a *global style* may be configured to set the general appearance of the editor. See `Language: Global Styles`: Individual elements for configuration are listed to the right under `Language: Style: <element>`. A *lexer style* involves setting the code highlighting rules for each keyword class of a given lexer. See `Language:<language>`: Individual keyword classes for each lexer are also listed under `Language: Style: <keyword class>`. See also Figure 3.

Each Silver Twilight theme consists of a global and a lexer style. See Figures 4 and 5 for the global style of Silver Twilight High and of Silver Twilight Lo respectively.

Note that the lexer styles for Silver Twilight Hi and Lo for CONTEXT are almost identical: The only

Keyword Class	Hi		Lo	
	Color	Sample	Color	Sample
DEFAULT (F)	Base03		Base03	
LINE COMMENTS	Base0		Base0	
TEX/ETEX	Maroon		Maroon	
LUATEX	Orange		Orange	
SYSTEM	Yellowgreen		Yellowgreen	
DOCUMENT	Green		Green	
STYLE	Yellow		Yellow	
CHARDEF	Magenta		Magenta	
CONSTRUCT	Violet		Violet	
PRIVATE	Blue		Blue	
STOP OPEN	Cyan		Cyan	
STOP CLOSE	Cyan		Cyan	
OPERATORS	Maroon		Maroon	
NUMBERS	Cyan		Cyan	

Figure 6 CONTEXT Lexer Style: Silver Twilight

difference is that the foreground and background colors for the DEFAULT keyword class are reversed; see Figure 6. This is intentional: the two themes are intended to form a single system. In order for a common lexer style to work well between light and dark themes, the color scheme has to be well thought out.[19] Again, there is always room for improvement.

4.5 ALM Fixed

The default font for Silver Twilight is Arabic-Latin Modern Fixed, a derivation from Latin-Modern Mono developed by the first author, Idris Samawi Hamid. Designed for extensive use of Arabic script and its diacritics, it has a larger than usual interline spacing. For those who desire tighter interline spacing or just another default typeface: Instead of tediously replacing the font in every dialog of

---

[19] The developer of Solarized had this ideal in mind: A single color scheme should work across nearly all keyword classes for each of a pair of light and dark themes. Note that a pair of Solarized themes is available for Notepad++ (the user will have to change any background tones used by the CONTEXT lexer style, as they are not compatible).

Style Configurator, one can open `ConTeXt.xml` and `stylers.xml` and make a global substitution of the name 'ALM Fixed' with that of another font (preferably fixed-width) of one's choosing, e.g., 'Dejavu Sans Mono'.

5 NPPEXEC, the Macro Submenu, and Shortcut Mapper

5.1 NPPEXEC and Console Scripts
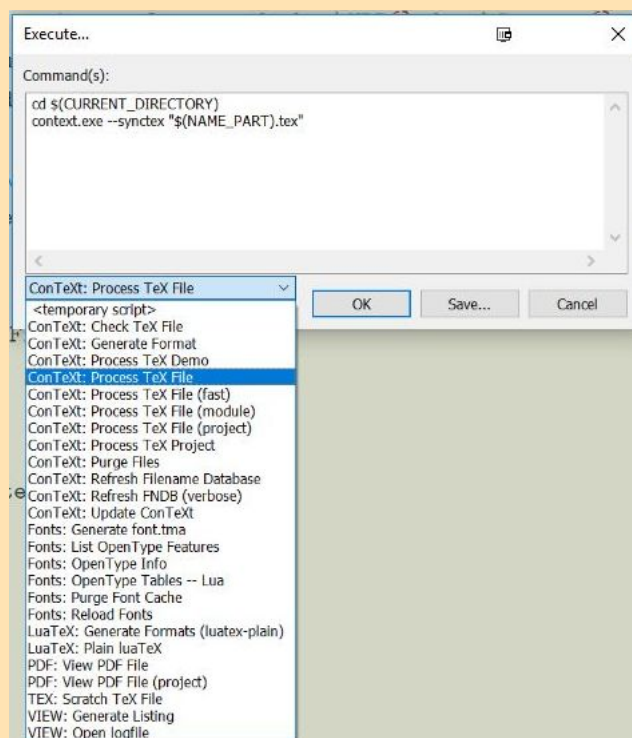
Figure 7   NppExec and
CONTEXT-related Scripts

The NppExec console emulator is an integral part of the Notepad++ for CONTEXT system.[20] When invoked (`F6` is the default shortcut), NppExec opens a dialog which features a typing area for one to write a script, an option to save it, as well as a drop-down list of saved scripts; see Figure 7. There are 22 scripts that come with Notepad++ for CONTEXT; one can add and remove these or one's own private scripts.

One can install multiple copies of NppExec; just copy `NppExec.dll` to, e.g., `NppExec2.dll`, etc. Then one will be able to, e.g., run two instances of LuaTEX simultaneously.

NppExec has considerable documentation, including its own manual; see

```
Plugins – NppExec – Help/Docs
Plugins – NppExec – Help/Manual.
```

For help using common DOS commands such as `copy`, `move`, and `mkdir` – these are actually part of the system interpreter `cmd.exe` – see Section 4.4. For an example, see Heading 6 in Section 5.2.

Let's take a look at

```
Plugins – NppExec – Advanced Options...
```

---

[20] Of course, a user can choose to configure an external console for use with Notepad++ via the `Run…` menu.
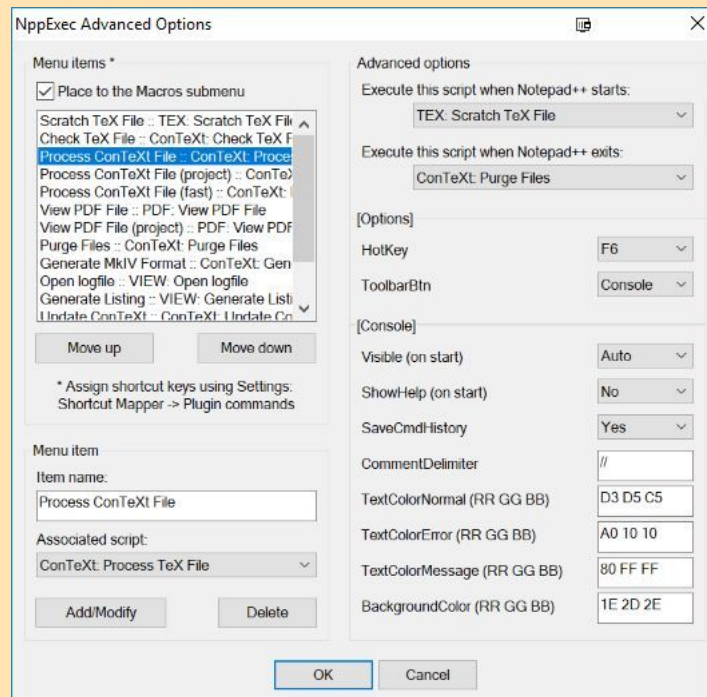
Figure 8 NppExec Advanced Options

See Figure 8. At the top right you will notice that a script can be executed when Notepad++ starts or exits. By default, the `Scratch TeX File` script is executed when Notepad++ starts: The user will have to edit that script and point it to the directory where one's scratch file is located; see Heading 13 in Section 5.2. The `Purge Files Keep SyncTEX` script is executed when Notepad++ exits; it then applies only to the temporary files associated with the leftmost tab (see also Heading 6 in Section 5.2). Of course one can disable the execution of any start or exit script.

The NppExec console emulator is configured to use a Solarized++ color scheme that closely resembles Silver Twilight Lo; see Section 9. This is also set in Advanced Options. The authors find the combination of Silver Twilight Lo for the console and Silver Twilight Hi for the main editing area to be quite pleasing.

5.2 NPPEXEC and CONTEXT

A brief description of each console script that ships with Notepad++ for CONTEXT follows. For each command, mention is made of whether it appears in the Macro submenu (see Section 5.3). If it has a default shortcut, that is also mentioned. Note that the last script executed by NppExec can be invoked by a shortcut, `F7` by default.

Most of the script names below should be self-explanatory; many are annotated. For an explanation of the environment variables, see the NppExec documentation. Remember that everything below – the number of scripts, the content of each script, the macro submenu, and shortcuts – can be easily configured to suit the user's needs.

1. **Check ConTEXt File (Ctrl+Alt+0)**

This script quickly checks the TEX file for errors without running MkIV.

Appears in Macro submenu: **Yes**

Figure 9 NppExec Console

```
mtxrun --autogenerate --script check
```

2. **ConTEXt: Process ConTEXt File (Ctrl+1)**

This script runs MkIV on the file visible under the active tab.

Appears in Macro submenu: **Yes**

```
context.exe "$(NAME_PART).tex"
```

3. **ConTEXt: Process ConTEXt Project (Ctrl+Shift+1)**

A main, project file often involves a multiple of subsidiary input files. One often wants to run the main file while working on a different file. This script runs MkIV on the main file, regardless of which tab is active. To view a project independent of the active tab, see Heading 15.

Appears in Macro submenu: **Yes**

The user will have to fill in the directory path and project name; see below:

```
cd "C:\<path to your directory>"
context.exe "<your project>.tex"
```

4. **ConTEXt: Process ConTEXt File (luajittex) (Ctrl+Alt+Shift+1)**

LuajitTEX is an experimental version of LuaTEX. It is not currently advertised for production purposes and may be ignored by the average user.

Appears in Macro submenu: **Yes**

```
mtxrunjit --autogenerate --script context test.tex "$(NAME_PART).tex"
```

5. **ConTEXt: Purge Files (Ctrl+3)**

   This script purges temporary files (ending in, e.g., `.tuc`, `.log`) that can clutter the user directory, take up cloud-storage space, etc.

   Appears in Macro submenu: **Yes**

   ```
   context --purge
   ```

6. **ConTEXt: Purge Files Keep SyncTEX**

   This script purges all temporary files except the SyncTEX file. This script is executed by default when Notepad++ exits (in which case, it applies only to the temporary files associated with the leftmost tab). One may then open Notepad++ at a particular line via SumatraPDF and SyncTEX; see Heading 13 in Section 3.1.

   Note the the use of `cmd /c ren` instead of the usual `ren.exe`. As mentioned earlier, see Section 4.4 of the NppExec manual.

   Appears in Macro submenu: **No**

   ```
   cmd /c ren *.synctex *.synctext
   context --purge
   cmd /c ren *.synctext *.synctex
   ```

7. **ConTEXt: Generate MkIV Format (Ctrl+4)**

   Appears in Macro submenu: **Yes**

   ```
   luatools.exe --generate
   context.exe --make
   ```

8. **ConTEXt: Update ConTeXt**

   Appears in Macro submenu: **Yes**

   ```
   first-setup.bat --engine=luatex
   ```

9. **ConTEXt: Refresh Filename Database**

   Appears in Macro submenu: **Yes**

   ```
   mktexlsr
   luatools.exe --generate
   ```

10. **ConTEXt: Refresh FNDB (verbose)**

    Appears in Macro submenu: **No**

    ```
    luatools.exe --verbose --generate
    ```

11. **LuaTEX: Generate Formats (luatex-plain)**

    This is for generating the LuaTEX version of the Plain TEXformat. Hardly needed by the average CONTEXT user.

Appears in Macro submenu: **No**

```
luatex --ini luatex-plain.tex
```

12. **LuaTEX: Plain LuaTEX**

The main use of Plain LuaTEX is to test possible bugs in LuaTEX. Hardly needed by the average CONTEXT user.

Appears in Macro submenu: **No**

```
luatex --ini luatex-plain.tex
```

13. **TEX: Scratch ConTEXt File (Ctrl+9)**

It's always good to have a scratch file on hand to, e.g., test individual elements and components of a larger project, explore or practice usage of some command, or make a minimal working example when encountering difficulty with some set of commands. This script is executed by default when Notepad++ starts; thus every session opens with a scratch file tab. See also Section 5.1.

Appears in Macro submenu: **Yes**

The user will have to fill in the directory path; see below:

```
notepad++.exe "C:\<path to your directory>\scratch.tex"
```

14. **PDF: View PDF File (Ctrl+2)**

We use SumatraPDF by default; see Heading 13 in Section 3.1.

Appears in Macro submenu: **Yes**

```
sumatra.bat "$(CURRENT_DIRECTORY)\$(NAME_PART).pdf"
```

15. **PDF: View PDF File (project) (Ctrl+Shift+2)**

This script will show the PDF file of the main file or project, independent of active tab; see Heading 3.

Appears in Macro submenu: **Yes**

The user will have to fill in the directory path and project name; see below:

```
cd "C:\<path to your directory>
sumatra.bat "<your project>.pdf"
```

16. **VIEW: Open logfile (Ctrl+5)**

Appears in Macro submenu: **Yes**

```
notepad++.exe  "$(CURRENT_DIRECTORY)\$(NAME_PART).log"
```

17. **VIEW: Generate Listing (Ctrl+7)**

Appears in Macro submenu: **Yes**

This script prints a pdf copy of the source file. In order to not override the normal pdf output, the script copies the source to a temporary TEX file with the suffix `-listing`; this file is deleted after the pdf is created.

```
mtxrun --autogenerate --script context --autopdf --extra=listing --compact --pretty
--result="$(NAME_PART)-listing" "$(NAME_PART).tex"
```

18. **Fonts: Purge Font Cache (Ctrl+Shift+3)**

MkIV generates a cache of OpenType and related tables for each font the first time it is used. The cache contains, e.g., an abbreviated version of the OpenType tables. When, e.g., a new version of a given font is installed, when that font is otherwise changed, or in certain other instances (e.g., bug fixes), the cache needs to be emptied and regenerated on a fresh run of MkIV.

Appears in Macro submenu: **Yes**

```
mtxrun.exe --script cache --erase
mtxrun --generate
```

19. **Fonts: List Font Info**

This produces a list of all the individual fonts belonging to a typeface family. The script below uses a sample font; replace `lmroman` with the font-family name of interest.

Appears in Macro submenu: **No**

```
mtxrun --script font --list --all lmroman
```

20. **Fonts: List OpenType Features**

This lists all of the GSUB and GPOS feature tags in the OpenType tables of a given font. The script below uses a sample font; replace `lmroman12-regular` with your font of interest.

Appears in Macro submenu: **No**

```
mtxrun.exe --script font --list --info lmroman12-regular
```

21. **Fonts: OpenType Tables – Verbose**

This generates the verbose version (ending in `.lua`) of the OpenType tables. The script below uses a sample font; replace `lmroman12-regular` with your font of interest.

Appears in Macro submenu: **No**

```
mtxrun --script font --list --info lmroman12-regular
```

For typeface families you may prefer to specify a pattern, e.g.,

```
mtxrun --script font --list --info --pattern=lmroman12
```

22. **Fonts: Reload Font Database**

This script comes in handy when the font database is incorrect for some reason, or when the script under Heading 18 above doesn't appear to work.

Appears in Macro submenu: **No**

```
mtxrun --script fonts --reload --force
```

5.3 The Macro and Run Submenus

From the NppExec Advanced Options dialog, any console script can be made to appear at the bottom of the submenu `Macro`. We have configured 14 NppExec scripts to appear there; see Figure 10. Note the submenu identifier need not be the same as the source console script name!
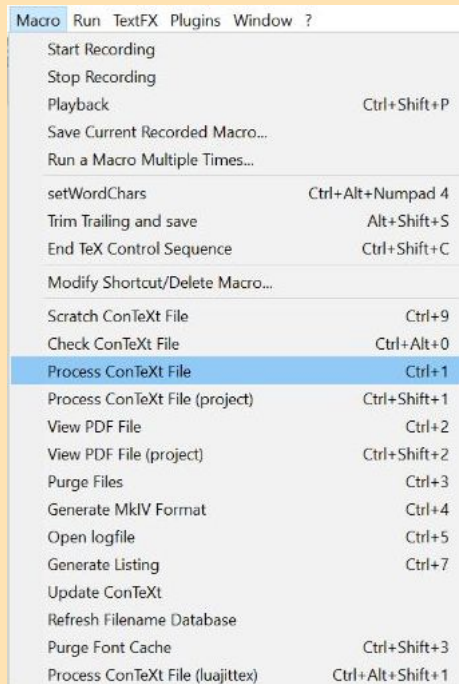
Figure 10 NppExec Scripts and the Macro Submenu

There is one submenu command under `Macro` that doesn't come from a console script, but was made via the normal macro-recording mechanism:

- **End T$_E$X Control Sequence (Ctrl+Shift+C)**

When invoked at the end of a given T$_E$X control sequence, this macro adds the string '`{}` ', including a single word space, to that sequence. At least for paragraph writing, to use a brace pair after a control sequence is generally better practice than to use a backslash.

Note also that the shortcut for invoking the Command Prompt (default shortcut `Ctrl+6`) occurs under the `Run` submenu.

5.4 Configuring Shortcut Mapper

Nearly all shortcuts for Notepad++ are configurable via

```
Settings — Shortcut Mapper
```

Shortcuts for those NppExec commands that appear in the `Macro` submenu will be found under the tab

```
Shortcut Mapper — Plugin Commands
```

which is organized by plugin. Shortcut Mapper can handle only two levels of submenus. For a third level, special treatment is needed; this brings us to the heart of the Notepad++ for CONTEXT system: the CONTEXT lexer and plugin.

6 The CONTEXT Plugin

6.1 Components of the Plugin

The CONTEXT plugin features the following three mechanisms:

1. **The Lexer**

2. **Autocompletion and Calltips**

3. **Tags, Templates, and Keys**

We now go over these in some detail.

6.2 Configuring Keyword Classes: Style Configurator and `ConTeXt.xml`

In the course of Section 4.3, we already discussed the keyword classes of the CONTEXT lexer, as well as their role in semantic highlighting. There are 14 classes, each of whose entries are currently generated via a Python script. The highlighting settings for each class may be managed through Style Configurator; some classes also feature a field where one may add user-defined keywords to the desired class as needed.

ConTeXt.xml is organized as follows:

1. **Language**

This follows the structure of the standard langs.xml that comes with Notepad++. A snippet:

```
<Languages>
    <Language commentEnd="" commentLine="%" commentStart="" ext="tex" name="ConTeXt">
        <!-- luatex -->
        <Keywords name="0">
        \Uchar [etc.]
        </Keywords>
    </Language>
</Languages>
```

2. **Style**

This follows the structure of the standard stylers.xml that comes with Notepad++. A snippet:

```
<LexerStyles>
    <LexerType desc="ConTeXt" excluded="no" ext="" name="ConTeXt">
    <WordsStyle bgColor="D3D5C5" fgColor="324140" fontName="ALM Fixed" fontSize="14"
  fontStyle="0" name="DEFAULT" nesting="0" styleID="0" />
    [etc.]
    </LexerType>
</LexerStyles>
```

User-defined keywords set in Style Configurator are saved within `<WordsStyle>`. Here is an example (much abbreviated):

```
<WordsStyle name="STYLE">\emph</WordsStyle>
```

This adds the control sequence `\emph` (at the time of this writing, missing from the CONTEXT autocompletion sources – see Section 3.1, Heading 11) to the keyword class STYLE.

We have tried to make the order of classes as useful as possible. Note that the syntax coloring of each class takes precedence over the one that is next down the list. Suppose one defines the following `\start|\stop<command>` environment:

```
\STARTTEST \STOPTEST
```

If used often or in multiple documents, one should place these into the `PRIVATE` keyword class: either by directly editing `ConTeXt.xml` – in which case they will appear as `Default Keywords` – or by using the `User-defined keywords` field in Style Configurator. Then one can place `\STARTTEST` in the `START OPEN` class, and `\STOPTEST` in the `STOP CLOSE` class. The result is that, because `PRIVATE` is prior to the two folding classes, you will get keyword folding with the semantic highlighting of `PRIVATE`.

In the default `ConTeXt.xml` that ships with Notepad++ for CONTEXT, the commands listed for the class `PRIVATE` are *highlights* that the first author has found useful for defining style elements: Given some text tagged with such an element, its style will be converted to an xml tag.[21] The macro definitions of these highlights are included in `ConTeXt.xml` for illustration purposes and have been commented.[22]

All lexer keywords and styles are specified in

```
/Program Files (x86)/Notepad++/plugins/Config/ConTeXt.xml
```

6.2.1 A Trick

If one takes a look under

```
Settings – Style Configurator – Language:
```

you will see that `ConTeXt` and other external lexers appear at the bottom of the `Language:` list.

If you wish to spend significant time to develop your own theme, or add lots of user-defined keywords to the relevant classes (see Section 4.3), it *may* be convenient (but is by no means necessary) to have it appear alphabetically in that list (to avoid having to scroll to the bottom of the `Language:` list too often). In that case you can copy the `<LexerType>` tag from `ConTeXt.xml` to its natural alphabetical location in

```
/Roaming/Notepad++/stylers.xml
```

Now a second presentation of the `ConTeXt` lexer will appear alphabetically in the `Language:` list. Notepad++ will use this as default, so CONTEXT styles will now be governed by

---

[21] Note that normal styling options in CONTEXT are ignored when CONTEXT output is set to xml. For example: `\emph{text}` will not be tagged and will show no emphasis in xhtml output. Using `\emphasis{text}`, on the other hand, will generate a tag for emphasis, which may then be configured for, e.g., css.

[22] A good practice is to make a file such as `highlights.tex` and place it in the private branch of your CONTEXT tree, e.g.,

```
/ConTeXt/tex/texmf-project/tex/aliases
```

or wherever else you choose. (Always remember to refresh the filename database!)

```
Style Configurator – Default theme: (stylers.xml)
```

When one is done with development, it *may* be convenient (but is by no means necessary) to copy `<LexerType>` for CONTEXT from `stylers.xml` back to `ConTeXt.xml`, then delete it from `stylers.xml`.

6.3 Configuring Autocompletion and Calltips: `context.xml`

Notepad++ has native autocompletion capabilities. However, it was not designed with TEX-style languages in mind; working with the backslash \ has limitations. Therefore a decision was made to reimplement autocompletion for the CONTEXT plugin. Both function and word completion are supported. Native Notepad++ offers a choice of function completion, word completion, or both; however, the plugin currently supports only both. Since functions in CONTEXT all begin with a backslash, and normal words generally do not, there is little-to-no ambiguity between words and functions.[23]

After typing the first three characters of a CONTEXT command or a previously used word, a popup will appear that will give a list of possible completions, from a subset of the list of official control sequences (in the case of function completion), or from a list of previously used words that begin with those thr characters (in the case of word completion). See Figure 6.3.
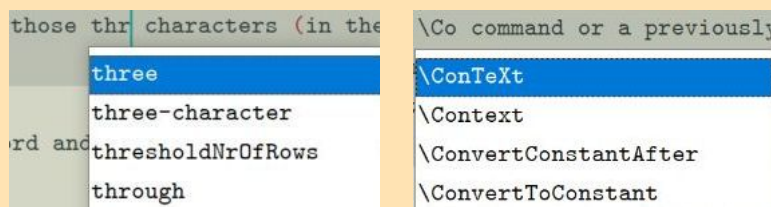
Figure 11   Word and Function Autocompletion

Many CONTEXT control sequences come with options specified within a pair of brackets, e.g., `big` in `\blank[big]`, or `offset=none` in `\framed[offset=none]`. It is virtually impossible to remember the bewildering array of options that come with thousands of control sequences, and having to consult some online documentation or the commands manual `setup-en.pdf` takes time. Oftentimes a post-autocompletion indicator, a so-called "calltip", is enough to give a hint or a reminder about the available options to a command.

When an official command is typed with the aid of autocompletion, and immediately followed by a left bracket `[` (the "trigger"), a calltip appears. The native Notepad++ calltip mechanism is one-dimensional: Once the calltip pertaining to some language appears following autocompletion and the relevant trigger, it shows a single line of information about the function. Any additional line of information must be selected by scrolling with a mouse. For viewing what is often a wide array of CONTEXT options this can be very inefficient. The CONTEXT-plugin implementation is two-dimensional: After the calltip triggered, it appears as a pane in columns; see Figure 12.

The list of official control sequences is found, and the calltip settings are configured, in the file

```
/Program Files (x86)/Notepad++/plugins/APIs/context.xml
```

The file `context.xml` is generated automatically from the latest CONTEXT sources via a Python script; see Section 3.1, Heading 11. One generally does not need to edit the autocompletion information. However the `<Environment>` tag at the beginning of the file does contain information that one may edit to configure the appearance of the calltips. Here are the default settings:

---

[23] That said, a final decision as to whether or not to provide the ability to turn off function or word autocompletion has yet to be made.
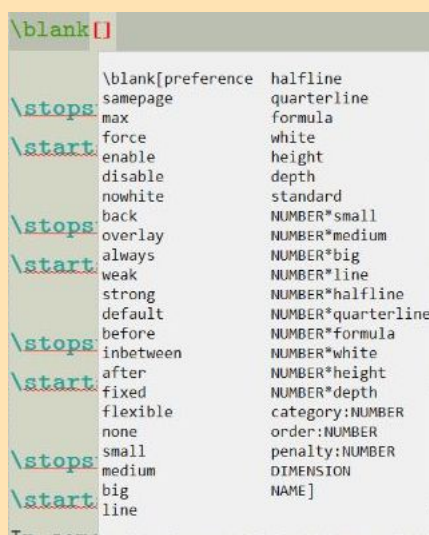
Figure 12 Calltips

```
<Environment additionalWordChar="" calltipBackColor="0xF0F0F0"
calltipFontName="consolas" calltipFontSize="9" calltipForeColor="0x101010"
columns="2" ignoreCase="no" macroValueOnSingleLine="yes" maxLineLength="70"
sortMacroValues="no" startFunc="" startFunc1="[" startFunc2="{"
startFunc3="(" stopFunc="" stopFunc1="]" stopFunc2="}" stopFunc3=")"
thresholdNrOfRows="20" toplines="1" widthColumnSep="2"/>
```

The current selection of available information about any given command reflects the limitations of the sources: As detail is added to the CONTEXT sources for autocompletion, more calltip information will become available to the user in future versions of `context.xml`.[24]

Private commands and macros are not included `context.xml`; hence there is no function completion available for them. They may, however, be included in `context-user.xml`. Here, advanced users may also configure calltip information for private macros. This will be useful for, e.g., writers of add-on modules for CONTEXT.

6.4 Configuring Markup Tags, Template Tags, and Keys: `ConTeXt.ini`

As mentioned in Section 1.2, the CONTEXT MkIV interface has developed considerably in the direction of a pure markup syntax, which demands the tagging of text with either two commands – these usually take the form of a pair of `\start|\stop<command>` sequences – or a pair of braces. Autocompletion of one command at a time for purposes of tagging is still tedious. The CONTEXT plugin features a user-configurable markup, template, and key system to make the handling of markup more accessible and efficient.[25]

Configuration of markup tags, templates, and keys are setup in

   Roaming/Notepad++/plugins/config/ConTeXt.ini

Let us now look at how each of these three subsystems is configured.

---

[24] In a future version we intend to add the ability to select parameters for each supported command.
[25] The authors again acknowledge the developer of WebEdit: His plugin was a major inspiration for the approach taken by the CONTEXT plugin; see Section 1.2.

6.4.1 Markup

The CONTEXT plugin displays the organization of markup tags in `ConTeXt.ini` as a system of submenus with support for two levels before `ConTeXt` (Level 1). For example:

```
Plugins — ConTeXt — <Level 2 Name 1> — <Level 3 Name 1>
                  — <Level 2 Name 1> — <Level 3 Name 2>
                  — <Level 2 Name 2> — <Level 3 Name 1>
                  — <Level 2 Name 2> — <Level 3 Name 2>
                  — <Level 2 Name 2> — <Level 3 Name 3>
```

In `ConTeXt.ini` we have the following default structure:

```
[Markup]
[Project]
[Document]
[Style]
[Highlights]
[Private]
[XML/HTML]
[MarkupEnd]
```

This specifies six `Level-2` menus (of course you can add, subtract, or reconfigure). The markup tags will be specified at Level-3. For example:

```
[Project]
 Project=\startproject%n|%n\stopproject%n
 Product=\startproduct%n|%n\stopproduct
 Component=\startcomponent%n|%n\stopcomponent
 Environment=\startenvironment%n|%n\stopenvironment
 TeXpage=\startTEXpage%n|%n\stopTEXpage
 Text=\starttext%n|%n\stoptext
```

The result is what you see in Figure 13.
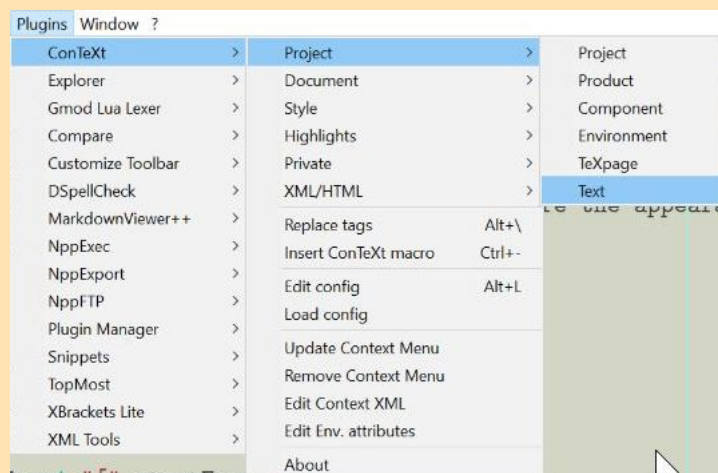
Figure 13   Organization of Markup-Tag Submenus

Now type some text:

```
Here is some text.
```

Select the above text, then go to and click on

```
Plugins – ConTeXt – Project – Text
```

This results in the following:

```
\starttext
Here is some text.
\stoptext
```

The above example illustrates the basic principle governing the markup-tags subsystem. The syntax of the markup tags is as follows:

```
; Syntax: <Item name>=<Left text>|<Right text>
; %n insert new line % n insert %n
```

The vertical bar | marks the position of the content to be wrapped by the `<Left text>` and `<Right text>`. In our example:

```
Project=\startproject%n|%n\stopproject%n
```

To open the configuration file `ConTeXt.ini` for editing, go to

```
ConTeXt – Plugins – Edit config (Alt+Shift+L)
```

(Like most shortcuts, this one can be configured via Shortcut Mapper; see Section 5.4.)

After editing, load the configuration file to update the plugin menus and invoked markup tags:

```
ConTeXt – Plugins – Load config (Alt+L)
```

For fine organization of markup tags, basic subdivision of Level-3 submenus is supported: Just place three dashes `---` where you want a dividing line to appear in the menu, e.g.,

```
Single Quotes=\quote{|}
---
Align Middle=\startalignment[middle]%n|%n\stopalignment
```

See Figure 14. There are three dividing lines:

- one above `Block Quote`;
- one above `Align Middle` and below `Single Quotes`; and
- one above `Text Braces` and below `Align Left`.
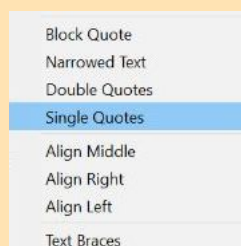
Figure 14   Dividing Lines in Submenus

In the default `ConTeXt.ini` that ships with Notepad++ for CONTEXT, the commands listed under `[Private]` are private macros by the first author; the definitions for these are double commented. These macros

are included for illustration purposes only; the user should replace them with one's own private commands.

6.4.2 Configuring the Right-Click Menu

Although one may go to `Plugins — ConTeXt` each time a markup tag is desired, it is often convenient to skip this step and to instead use the Notepad++ right-click menu mechanism. Settings for this mechanism are configured in a file entitled, unhappily for CONTEXT users, `contextMenu.xml`. In most cases a user will manually add the names of menu items to the configuration file, for example:[26]

```
<Item PluginEntryName="TextFX Characters"
      PluginCommandItemName="Proper Case"
      ItemNameAs="Proper Case" />
```

However, for the CONTEXT plugin things are not so simple; this is, in part, because the right-click mechanism wasn't designed for things like our Level-3 submenus. So the CONTEXT plugin has been designed to, upon request, populate `contextMenu.xml` using id codes internally generated to Notepad++, for example:

```
<Item FolderName="Project"
      id="23004"
      ItemNameAs="Text"
      User="" />
```

To copy the Level-2 and Level-3 submenus to the right-click menu, go to

```
Plugins — ConTeXt — Update right-click menu
```

Unlike the case with the loading of `ConTeXt.ini` after it is edited (`Load config`), one has to restart Notepad++ in order to complete update of the right-click menu. Once updated, the submenu subsystem will appear upon any right click (`Shift+F10` is the usual system shortcut); see Figure 15.
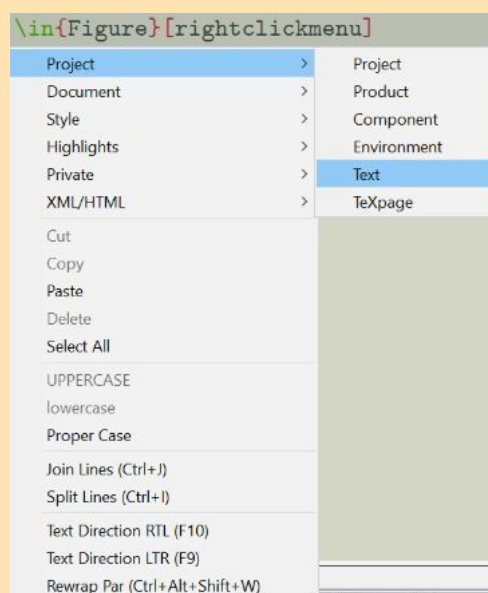
Figure 15   The Right-Click Menu

Once setup, select some text, right click, then choose the desired markup tag with which to wrap that text.

The `User` attribute (empty in the above snippet) allows the user to give any right-click item a name different from the default name featured by the `Plugins` menu. Hence there is the option to edit `contextMenu.xml` manually. But update through the plugin first! Then make your changes, save, and restart Notepad++.

6.4.3 Keys

The default submenu configuration that ships with Notepad++ for CONTEXT features over 80 markup tags, and many more could be added. Some tags will be used frequently; it would be convenient to have shortcuts for them. Unfortunately, Shortcut Mapper is not designed to handle Level-3 submenus. Misfortune in this case gave way to the opportunity to do something better: Second author Luigi Scarso has implemented a *key-based* shortcut system for our CONTEXT plugin.[27]

In `ConTeXt.ini` one sets a one-, two-, or three-character key within a pair of parenthesis and immediately prior to the name of the markup tag of interest, for example:

```
[Style]
(em)Emphasize=\emph{|}
(ty)Type=\type{|}
```

To see the available keys, go to

```
Plugins — ConTeXt — Insert ConTeXt macro (Ctrl+-)
```

One will normally invoke this via the shortcut `Ctrl+-`. A popup window will appear with the full list of markup tags. A list of keys previously set in `ConTeXt.ini` is displayed in the left column; see Figure 16.
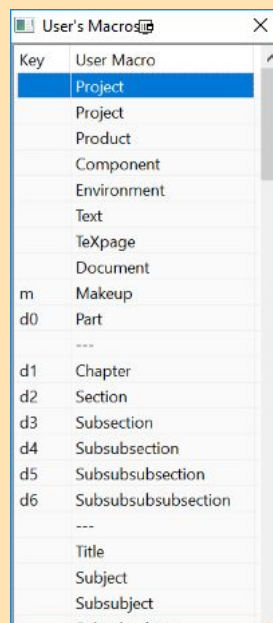
Figure 16 The User's Macros and Keys Popup

Opening the key window triggers a timer: To wrap some selected `<text>` in a markup tag, now quickly type the key that has been set for that macro, e.g., `em` for `\emph{<text>}`. The available settings for

---

[27] This is inspired, in part, by the "mnemonics" system used by Emacs.

the timer and the popup window may be configured under `[CommandsSetup]`, located at the first line of `ConTeXt.ini`:

```
[CommandsSetup]
usermacro:elapse=400
um:elapse_shift=500
um:display_rows=40
```

The two elapse variables are in milliseconds. The second variable, `um:elapse_shift` gives the time lapse for keys that require a `Shift`, e.g., capital letters. One can fine tune the values of the elapse variables until they are in sync with one's typing speed. The third variable controls the number of rows displayed in the popup: The current maximum value is 100.

Finally, one can invoke a macro directly from the popup window: Just double click on any row that corresponds to a markup tag and it will wrap the selected text with that tag.

6.4.4 Templates

The CONTEXT plugin supports one more way to enter markup or other code: *templates*. In `ConTeXt.ini` these are configured after `[Templates]`. For example:

```
[Templates]
item2=\startitemize%n\startitem[]%n|%n\stopitem %n
        \startitem[]%n%n\stopitem %n\stopitemize
```

In one's document, go to a new line and type the keyword `item2`. Then choose

```
Plugins — ConTeXt — Replace tags (Alt+\)
```

As usual, one normally just invokes the shortcut. This will result in

```
\startitemize
\startitem[]

\stopitem
\startitem[]

\stopitem
\stopitemize
```

with the cursor placed on the line between the first `\startitem[]`-`\stopitem` pair. Replacing the keyword `item7` will produce seven `\start|\stopitem` tags.

The template system is perfect for things like tables. For example, if one types and replaces the keyword `TABLE22` – See `ConTeXt.ini` – the result will be

```
\placetable{}
{\bTABLE
\bTR \bTD  \eTD \bTD  \eTD \eTR
\bTR \bTD  \eTD \bTD  \eTD \eTR
\eTABLE}
```

with the cursor placed between the first `\bTD`-`\eTD` pair.

7 Note on Bidirectional Editing and Scintilla

As mentioned in Section 2.2, one of the distinguishing characteristics of Notepad++ in comparison with other Scintilla-based editors is its support of global bidirectional editing. This means that one can set the global direction of the editor from LTR to RTL and back; see

```
View — Text Direction RTL
View — Text Direction LTR
```

When RTL reading order is chosen, one can type and edit, e.g., Arabic and Hebrew text normally.[28] Note the following significant but non-critical limitations:

• Notepad++ support for global directionality applies to the entire editing area and to all editing tabs. When applied, every open document will appear as either global RTL or global LTR.

• For each tab, global directionality applies to the entire document, not line by line.[29] That is, every line or paragraph will obey the global direction.

Moving along: As long as one works with some script whose directionality matches the *global* direction, global RTL or LTR reading order ('order' for short) each behaves as expected. When one mixes *local* RTL and LTR text within the same line, however, then challenges arise. See Figure 17: Visually, the mixed-direction text in global LTR order, and the same text in global RTL order, look exactly as they would in any editor that fully supports bidi (such as MS Notepad). But looks are deceiving. In general: When in global LTR order, typed RTL text will look normal but one cannot visually edit it; when in global RTL order, typed LTR text will look normal but one cannot visually edit it. For example, if one visually selects, copies, and pastes some RTL text while in global LTR order, the pasted text will generally not look as one expects. If one starts typing in the middle of an RTL word while in global LTR order, what one types will generally not appear where one expects it to. See Figure 18: If, from global LTR order, one double-clicks in the middle of the word 'هو' to select it, one sees the selection background color doing something odd: What the editor actually selects is the word 'امتحان'. The RTL text that appears on the screen does not match what the editor is doing because Scintilla can only process the text in one direction at a time. If one switches to global RTL and visually selects the same word, Scintilla behaves as expected.

Figure 17 Appearance of Bidi in the CONTEXT Lexer: Global LTR (left) Global RTL (right)

Figure 18 Selecting an RTL Word in the CONTEXT Lexer: Global LTR (left) Global RTL (right)

But this is not the whole story. An oddity that one finds, not only in Notepad++ but in SciTE as well, is that not all lexers behave exactly the same. There are actually two *modes* of Scintilla bidirectional behavior. The first (and apparently the most common among Notepad++ lexers) is the one described just above: Visually, mixed-direction text looks correct with either global RTL or LTR reading order

---

[28] By default (Shortcut Mapper, `shortcuts.xml`) we use `Alt+X` to switch to RTL, `Alt+Z` to switch to LTR. These direction options are also supported in the default right-click menu (`contextMenu.xml`). In Microsoft Notepad, what Notepad++ calls *Text Direction RTL* is called *Right to left Reading order*.

[29] Note that we are using 'line' in the editor sense. Thus a typeset paragraph may be represented as a single line in the editor, separated from other such lines by whitespace. See also Section 2.4.2.

activated, but the only text that can be edited naturally is that whose local directionality matches the global direction of Notepad++. But if one switches the language to, e.g., Lua, things look significantly different. See Figure 19: In global LTR, each individual continuous RTL string (such as a word) looks and reads RTL, but a concatenation of continuous RTL strings reads LTR. Similarly: In global RTL, each individual continuous LTR string looks and reads LTR, but a concatenation of continuous RTL strings reads RTL.

Figure 19   Appearance of Bidi in the Lua Lexer: Global LTR (left) Global RTL (right)

This apparent oddity brings a significant benefit: Visual selection of any complete word or continuous text string by double-click now matches the editor selection! See Figure 20: A double-click to select the RTL string 'هو' now works as expected in global LTR order. Similarly, a double-click to select an LTR string now works as expected in global RTL order; see Figure 21, which compares the behavior of the CONTEXT lexer (left) and that of the Lua lexer (right). Note that this does not affect bidirectionality *within* a word or continuous string: Within this mode of Scintilla, one can select an entire word regardless of whether it is RTL or LTR, but one cannot naturally select, type, or edit individual characters within that word.

Figure 20   Selecting an RTL Word in the Lua Lexer: Global LTR (left) Global RTL (right)

Figure 21   Selecting an LTR Word in CONTEXT (left) and Lua (right): Global RTL

Despite this persistent limitation: From the perspective of efficient editing, it is arguable that the second mode of Scintilla behavior – where a concatenation of continuous RTL (LTR) strings reads LTR (RTL) while each individual string apears RTL (LTR) – is more useful than the first mode – where a pure visual bidirectionality is maintained without the ability to select individual words whose directionality is opposite to the global reading order. At this stage, the authors have yet to identify the exact "trigger" which governs whether a lexer will adopt the first Scintilla mode or the second. Once that is identified, we may configure the lexer to adopt the second behavior, provide a choice to the user, or embark upon some other, creative, protocol.[30]

A related question is whether or not Notepad++ or the CONTEXT lexer can be extended via, e.g., a Lua extension, to fully support bidirectional editing in the proper manner. Of course, the best solution would be for the Scintilla development team to add bidirectional support to Scintilla directly.

References

Bringhurst, R. (2008). *The Elements of Typographic Style, Version 3.2*. Hartley & Marks, Publishers.
(p. 16)

Sarkar, A. (2015). The impact of syntax colouring on program comprehension. In The impact of syntax colouring on program comprehension. *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group (PPIG 2015)*. Author.   (p. 12)

---

[30] The original CONTEXT UDL – see Section 1.2 – uses the second mode, whereas our current lexer uses the first. In SciTE (LTR global text direction only), the TEX lexer also uses the second mode, whereas the MetaPost lexer uses the first.

The Authors

author   Idris Samawi Hamid, Professor
email   ishamid@colostate.edu
affiliation Department of Philosophy
            Colorado State University
            The Oriental TƐX Project

author   Luigi Scarso
email   luigi.scarso@gmail.com
affiliation The ConTƐXt Development Team
            The LuaTƐX Team

version   February 28, 2018