KeenWrite

User Manual

Version: 3.6.3
Date: 08-SEP-2025

Contents

1 Introduction

KeenWrite is a free desktop editor that transforms Markdown documents into beautifully typeset PDF files. Whether you're crafting technical documentation, reports, academic papers, or creative content, KeenWrite combines the simplicity and readability of Markdown with high-quality publishing capabilities.

1.1 Rethink

The core philosophy behind KeenWrite may require shifting how you think about writing documents. Unlike traditional word processors that mix content and styling, KeenWrite aims to completely separate what is written from how it appears. You write in plain text and instruct the computer to typset the document into print-ready PDF files. This approach allows for:

- **Rebranding** – Corporations can redesign documents in weeks, not years
- **Consistency** – Ensure uniform, high-quality output across documents
- **Future-proof** – Plain-text files will be readable long after proprietary formats go obsolete
- **Version control** – Plain text works seamlessly with version control systems
- **Reproducibility** – Generate the same output using different systems
- **Focus** – Write without distraction from formatting concerns

1.2 Reflow

Learning how KeenWrite converts plain text into PDF can help understand its usage.

The original intent of Markdown was to simplify writing web pages. As such, there are numerous tools to convert Markdown to HTML. KeenWrite goes one step further and converts Markdown to XHTML, which is a *well-formed* version of HTML and also a type of XML. The typesetting system can transform XML documents, and therefore XHTML documents, into typesetting instructions. Those instructions add styling (colours, fonts, layout, etc.) to the final output document.

In a nutshell: **Markdown → XHTML → TEX → PDF**

This chain allows separating content (the Markdown source) from presentation (a TEX typesetting system), allowing the two to vary independently. Incidentally, it also allows publishing the same content to the web, styled using cascading stylesheets.

1.3 Features

KeenWrite has numerous tools to assist with document creation.

1.3.1 Writing and editing

Built-in features to help write and edit documents include:

- **User-defined variables** for consistency, with autocomplete
- **Spell checking** to catch typos as you type
- **Document outline** for easy navigation in long documents
- **File manager** to view related files within the editor
- **Word count** and document statistics

1.3.2 Export formats

KeenWrite provides multiple export formats from Markdown:

- **PDF files** typeset using the ConTEXt typesetting system
- **XHTML files** for publishing to the web
- **Markdown** with variables interpolated and substituted

1.3.3 Advanced capabilities

Beyond basic document creation, KeenWrite includes:

- **Data visualization** through R integration
- **Diagram generation** supporting PlantUML, Mermaid, and other text formats
- **Variable interpolation** enables a single source of truth for content
- **Internationalization** with font handling for multiple languages
- **Mathematical** equations and formulas rendered beautifully

1.3.4 User interface

The user interface has:

- **Dark mode** and other skin customizations
- **Multi-document editing** with detachable tabs
- **Live preview** showing formatted output as you write
- **Distraction-free** settings when you need to focus

1.4 Target users

KeenWrite is designed for anyone who needs to produce high-quality documents while maintaining control over content and presentation:

- **Technical writers** creating documentation, user manuals, and specifications
- **Researchers and academics** writing papers with mathematical content
- **Authors** producing books and long-form content

1.5 Quick start

The easiest way to begin using KeenWrite:

1. **Download** the application from **keenwrite.com**.
2. **Run** the application (no installation necessary).
3. **Create** your first document by clicking **File → New**.
4. **Start writing** in Markdown format.
5. **Preview** your work in the preview tab.
6. **Export** to PDF when ready.

To generate PDF files, you'll also need:

- The theme pack to apply a look and feel
- ConTEXt typesetting system (can be installed automatically)

If you have opted to download the Java version of KeenWrite, see **Section 2.2** for installation instructions.

1.6 Document structure

This manual is organized to support both newcomers and experienced users seeking specific information. You can read the manual sequentially to build expertise gradually, or jump to specific chapters as needed.

1.7 Getting help

When you need assistance or encounter challenges, multiple resources are available to help. If you encounter issues or need assistance:

- Check the **Troubleshooting** chapter for common problems and solutions
- Visit the **Screenshots** chapter to see the interface in action
- Consult the **Command-line reference** for automation tasks
- Review the **Architecture** chapter if you're interested in extending KeenWrite

The remaining chapters will guide you through installation, basic usage, and gradually introduce advanced features.

2 Installation

This chapter covers installing the application and its dependencies, including the typesetting software required for PDF generation. Generating PDF files requires the following components:

- KeenWrite binary
- KeenWrite theme pack
- ConTEXt typesetting system

The easiest way to get started is to download a KeenWrite binary for your operating system, run it, then launch the guided wizard as per **Section 2.1**. The wizard will download and install all necessary components.

KeenWrite binaries are self-installing, except for the Java archive (`.jar`) version. Installing the Java archive version is useful for platforms that don't have prebult KeenWrite binaries. The Java archive version is described in **Section 2.2**.

2.1 Wizard

Launch the wizard as follows:

1. **Download** KeenWrite.
2. Start KeenWrite.
3. Click **File → Export As → PDF**.

The installation wizard installs a container manager and typesetter container image. The container image is a self-contained operating system that includes the typesetting software, theme pack, and fonts. Using a container reduces the number of manual installation steps.

2.2 Java Runtime Environment

Using the Java archive version of KeenWrite requires the following:

- `keenwrite.jar` file.
- Java Runtime Environment (JRE) **24.0.2+12** with JavaFX.
- KeenWrite launch script.

Software developers may choose to use the Java Development Kit (JDK) **24.0.2+12**.

2.2.1 Download KeenWrite

Download the Java archive version of KeenWrite.

On Windows:

1. Create `C:\Users\%USERNAME%\keenwrite`.
2. Save `keenwrite.jar` into the new directory.

2.2.2 JRE with JavaFX

Install a JRE **24.0.2+12** version that bundles JavaFX.

1. Visit the BellSoft's Liberica JDK **download page**.
2. Select the JDK corresponding to **24.0.2+12**.
3. Scroll down to the desired operating system.
4. Optionally, change the target CPU architecture.
5. Set **Package** to: `Full JRE`.
6. Download the `.zip` or `.tar.gz` file.

After downloading the *full* JRE, install it and make sure Java can be found when run. On Unix systems, my preferred installation method follows:

1. Edit `$HOME/.bashrc`.
2. Add `export JAVA_HOME="/opt/jre"`.
3. Add `export PATH="${PATH}:${HOME}/.local/bin:${JAVA_HOME}/bin"`.
4. Save the file.
5. Install into `/opt/jre-24.0.2+12-full` and link to `/opt/jre`:

```
sudo su -
cd /opt
JAVA_DIR="jre-24.0.2+12-full"
mkdir "${JAVA_DIR}"
tar xf bellsoft-jre24.0.2+12-linux-amd64-full.tar.gz \
  -C "${JAVA_DIR}" --strip-components=1
ln -sf "${JAVA_DIR}" jre
```

Pointing the symbolic link `/opt/jre` to `/opt/jre-24.0.2+12-full` enables upgrading the JRE without having to update the `.bashrc` for each new version.

2.2.3 Unix launch script

On Unix-based systems (Linux, macOS, FreeBSD, alpine, etc.), install the launch script as follows:

1. Open a terminal.
2. Download the launcher script into `${HOME}/.local/bin`.
   ```
   wget --quiet --content-on-error \
     "https://repo.autonoma.ca/keenwrite.git/raw/HEAD/scripts/keenwrite.sh"
   ```
3. Make the script executable.
   ```
   chmod +x "keenwrite.sh"
   ```
4. Download or move the `.jar` file into `${HOME}/.local/bin`.

Provided `${HOME}/.local/bin` is part of the `PATH` environment variable, KeenWrite can be run by typing `keenwrite.sh` and pressing ↵ Enter.

2.2.4 Windows launch script

On Windows-based systems, install the launch script as follows:

1. Browse to the **launcher script**.
2. Right-click anywhere on the page.
3. Click **Save Page As** (or similar).
4. Browse to `C:\Users\%USERNAME%\keenwrite`.
5. Confirm saving the file.
6. Rename the file from `.bat.txt` to `.bat`, if necessary.

The Java version of KeenWrite is installed.

2.3 ConTEXt LMTX

For users who prefer more control over the installation process, manually download and configure the typesetting software as follows:

1. Start KeenWrite.
2. Click **File → Export As → PDF**.
3. Note the following details (e.g., Windows x86 64-bit):
   - operating system name;
   - instruction set; and
   - architecture.
4. Click the **install page** link in the dialog.
5. Download the archive file for your operating system and architecture.

Follow the steps that apply to the computer's operating system:

- **Unix** (includes macOS, FreeBSD, Linux, and similar)
- **Windows** (includes Windows 7, Windows 10, and similar)

2.3.1 Unix

On Linux, macOS, FreeBSD, and similar systems, replace `context-linux-64.zip` with the version appropriate to your operating system and CPU architecture. Proceed:

1. Open a terminal.
2. Run the following commands:

```
sudo su -
export CONTEXT_FILE="context-linux-64.zip"
export CONTEXT_DIR="/opt/context"
mkdir -p "${CONTEXT_DIR}"
cd "${CONTEXT_DIR}"
wget "https://lmtx.pragma-ade.nl/install-lmtx/${CONTEXT_FILE}"
unzip "${CONTEXT_FILE}"
rm "${CONTEXT_FILE}"
chmod +x install.sh
./install.sh
exit
echo export CONTEXT_HOME=\"${CONTEXT_DIR}\" >> ${HOME}/.bashrc
echo PATH="\"\${PATH}:\${CONTEXT_HOME}/tex/texmf-linux-64/bin\"" \
  >> ${HOME}/.bashrc
source ${HOME}/.bashrc
```

ConTEXt is installed; jump to **Section 2.3.3**.

2.3.2 Windows

On a Windows operating system, install the typesetting software as follows:

1. Extract the `.zip` file into `C:\Users\%USERNAME%\AppData\Local\context` (the "root" directory).
2. Run **install.bat** to download and install the software.
   - If prompted, click **Run** (or click **More info** first).
3. Download **localpath.bat**.
4. Save the file to the typesetting software's "root" directory.
5. Rename the file from `.bat.txt` to `.bat`, if necessary.
6. Run `localpath.bat` (to set and save the `PATH` environment variable).

ConTEXt is installed.

2.3.3 Verification

After completing the installation process, verify that the typesetting software has been installed correctly as follows:

1. Open a new terminal or command prompt.
2. Type: `context --version`
3. Press ⏎ Enter.

If version information is displayed then the software is installed correctly.

2.4 Theme packs

Theme packs define the visual appearance and styling of typeset documents. Installing and configuring theme packs is essential for producing high-quality PDF files. Each theme has its own requirements.

2.4.1 Install theme pack

Install and configure the theme pack as follows:

1. Download the **theme-pack.zip** archive.
2. Extract the archive.

   - On Unix, use `${HOME}/.local/share/keenwrite/`.
   - On Windows, use `C:\Users\%USERNAME%\AppData\Roaming\keenwrite\`.
3. Start KeenWrite, if not already running.
4. Click **Edit → Preferences**.
5. Click **Typesetting**.
6. Click **Browse** beside **Themes**.
7. Navigate to the `themes` directory.
   - On Unix, `${HOME}/.local/share/keenwrite/themes/`.
   - On Windows, `C:\Users\%USERNAME%\AppData\Roaming\keenwrite\themes\`.
8. Click **Open**.
9. Click **OK**.

The theme pack is installed.

2.4.2 Configure theme

Once a theme pack is installed, additional configuration may be required based on the theme's formatting needs. Configure individual themes based on their specific requirements. For example, the *Boschet* theme requires downloading and installing specific font families, which include:

- **Inconsolata**
- **Libre Baskerville**
- **Niconne**
- **Open Sans Emoji**
- **Source Serif 4**
- **TW Kai**
- **Underwood Quiet Tab**

See your operating system instructions for how to install a font.

When running ConTEXt within the typesetter container image, all fonts are included.

2.5 Run

Running KeenWrite will depend on the operating system and installation method. **Table 2.1** lists the executable file name used to start the application.

Table 2.1 KeenWrite launchers

Operating System	Method	Launcher file name
Unix	Binary	`keenwrite.bin`
Unix	Java archive	`keenwrite.sh`
macOS	Binary	`keenwrite.app`
macOS	Java archive	`keenwrite.sh`
Windows	Binary	`keenwrite.exe`
Windows	Java archive	`keenwrite.bat`

3 Overview

This chapter describes how to basic editing functionality.

3.1 First document

Create and open a document as follows:

1. Start KeenWrite.
2. Click **File → Save As** to save the untitled file.
3. Set the file name to: `metadata.yaml`
4. Save the (empty) file.
5. Click **File → Open** to load the variable definition file.
6. Change **Source Files** to **Variable Files** to list variable definition files.
7. Browse to and select the `.yaml` file.
8. Click **Open**.

The variable definitions appear in the variable definition tab under the heading of **Variables**.

3.2 Basic syntax

The editor supports standard Markdown features listed in **Table 3.1**. While editing, try the toolbar icons to see how formatting appears in the preview tab.

Table 3.1a Basic Markdown syntax

Style	Syntax
Bold	`**strong**`
Italic	`*emphasis*` or `_emphasis_`
~~Struck text~~	`~~struck~~`
Horizontal rule	`---` on an empty line
Unordered lists	`- item` or `* item`
Ordered lists	`1. item`
Inline `code`	`` `code` ``
Links	`[label](url)`

Table 3.1b Basic Markdown syntax

Images	`![alt text](resource)`
Blockquotes	`> quoted text`
Headings	`#`, `##`, `###`

Remember that most of these items are suggestions to the presentation layer. How they may appear in the final document may differ from what you anticipate.

3.3 Preview

The preview tab updates in real time as you type. For example, if your variable file contains:

```
novel:
  title: "Diary of {{novel.author}}"
  author: "Anne Frank"
```

Then typing:

```
The novel *{{novel.title}}* is a widely read book.
```

Will render as:

> The novel *Diary of Anne Frank* is a widely read book.

To autocomplete variable names:

1. Create a new file.
2. Type in a partial variable value, such as `Dia`.
3. Press Ctrl+Space (hold down the Control key and tap Space).

The editor inserts:

```
{{novel.title}}
```

And the preview tab shows:

> Diary of Anne Frank

3.4 Export

To export your document as a PDF:

1. Click **File → Export As → PDF**.
2. Select a theme from the drop-down list.
3. Click **OK** or press ⏎ Enter.
4. Choose a file name and location.
5. Click **Save**.

The document will be typeset using ConTEXt and saved as a PDF. You can view it in any PDF reader.

To combine multiple documents into one PDF:

1. Click **File → Export As → Joined PDF**.
2. Or press Ctrl+Shift+p.

4 Variables

Variables are placeholders for values that can be defined once and reused throughout the document. When the document is built, each variable is replaced with its corresponding value.

4.1 File format

YAML (meaning, *YAML Ain't Markup Language*) is a plain-text format used for storing structured data. It's commonly used in configuration files due to its readability and simplicity. YAML files allow encoding key-value pairs within a nested hierarchy.

Any number of variables can be defined, in any order:

```
key_1: Value 1
key_2: Value 2
```

A variable's value can reference other variables by enclosing the referenced key in double curly braces (`{{ }}`)—also known as "moustache" notation—wrapped in quotation marks:

```
key: Value
key_1: "{{key}} 1"
key_2: "{{key}} 2"
```

Variables can be nested to group related information:

```
document:
  title: Document Title
  author: Author Name
  isbn: 978-3-16-148410-0
```

Reference nested keys using a period, such as:

```
novel:
  author: Author Name
copyright:
  owner: "{{novel.author}}"
```

4.2 Usage

The preview tab updates as you type. Imagine a variable file contains:

```
novel:
  title: "Diary of {{novel.author}}"
  author: "Anne Frank"
```

Then typing:

```
The novel *{{novel.title}}* is a widely read book.
```

Displays in the preview pane as:

> The novel *Diary of Anne Frank* is a widely read book.

4.3 Autocomplete

Typing variable names is laborious. To autocomplete variable names:

1. Type in a partial variable value, such as `Dia`.
2. Press Ctrl+Space (hold down the Control key and tap Space).

The editor inserts:

```
{{novel.title}}
```

And the preview tab shows:

> Diary of Anne Frank

5 Metadata

Metadata is information about a document such as title, author keywords, copyright date, ISBN, and more. Most systems require coupling metadata directly to the source document. KeenWrite fetches reads metadata from the variable definitions file instead. This allows reusing the same metadata across documents.

Metadata can be inserted into the output document either through the document properties in the user interface or via the command line.

5.1 Document properties

Metadata can be mapped in the user preferences, shown in **Figure 5.1**. When the document is exported, the document metadata variables are added to the `<head>` tag as `<meta>` tags.

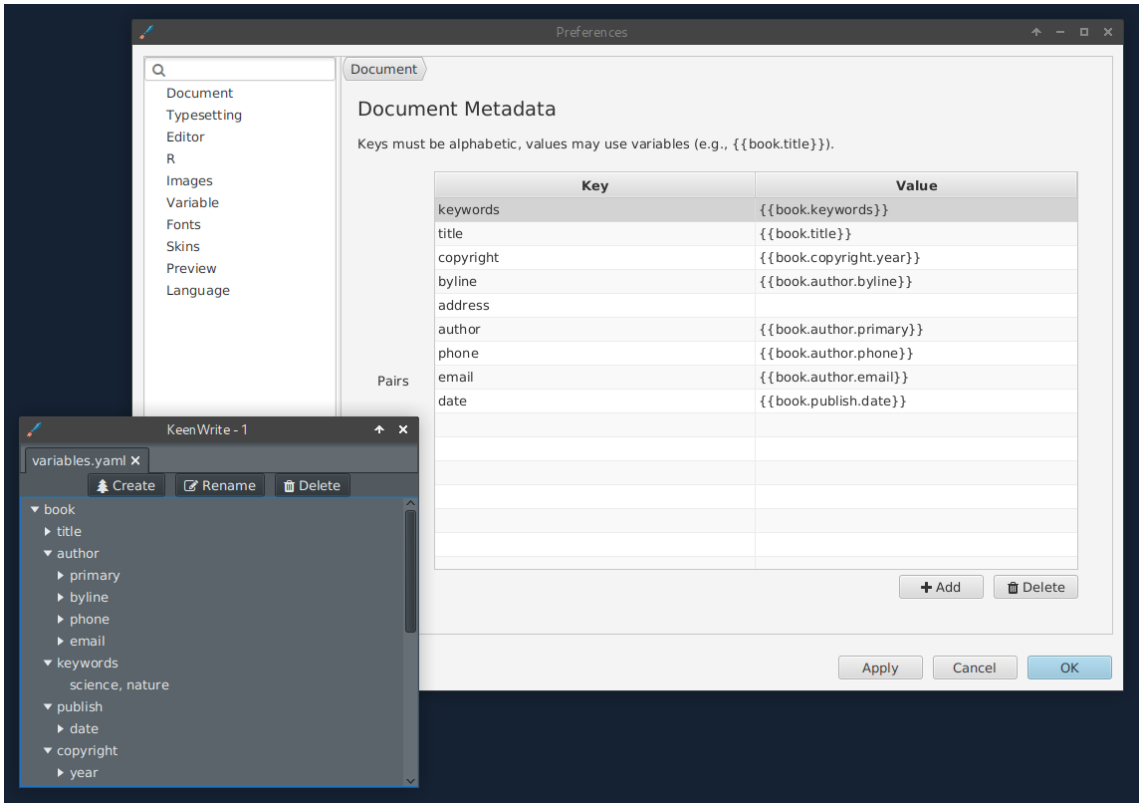

Figure 5.1 Document metadata preferences

The **Key** column lists metadata names and the **Value** column lists the metadata content for each corresponding **Key**. The content may include references to variable definitions. When the document is typeset, the values for the variables will be substituted upon export. When exporting to XHTML format, the header will include the keys and values as `<meta>` tags. For example:

```
<head>
  <title>Document Title</title>
  <meta content="science, nature" name="keywords"/>
  <meta content="Penn Surnom" name="author"/>
  <meta content="4311" name="count"/>
</head>
```

5.2 Command-line

Recall the document metadata from earlier:

```
document:
  title: Document Title
  author: Author Name
```

These data can be provided via command-line arguments as follows:

```
keenwrite.bin \
  -i "01-introduction.md" \
  -v "metadata.yaml" \
  --metadata="title=KeenWrite" \
  --metadata="author={{document.author}}"
```

The resulting XHTML document will resemble:

```
<head>
  <title>Document Title</title>
  <meta content="Author Name" name="author"/>
  <meta content="1234" name="count"/>
</head>
```

We'll take a deeper look at command-line usages in **Chapter 14**.

5.3 Special fields

Certain metadata keys have special handling and formatting rules when documents are exported to different formats. When exporting the document, note the following special metadata:

- **count** – Document word count, always included.
- **author** – Included as PDF metadata.
- **keywords** – Included as PDF metadata.
- **document.title** – Populates the `<title>` tag, not a `<meta>` tag.

Except for **count**, all of these metadata variables can be customized.

Some themes, such as *Handrit*, have additional metadata that may be included in the final document automatically. *Handrit*, in particular, has a way to pass in a pseudonym (or by-line) for publishing manuscripts under a pseudonym.

6 Basic Markdown

This section outlines the basic Markdown syntax supported by KeenWrite, including paragraphs, headers, lists, links, and more. Countless Markdown syntax resources exist, so only a brief refresher is provided.

6.1 Paragraphs

Paragraphs are created by separating blocks of text with one or more blank lines.

```
This is the first paragraph.

This is the second paragraph.
```

Without a blank line, the prose will be typeset as a single paragraph.

6.2 Headers

Headers are created using one or more hash (`#`) symbols followed by a space and the header text. The number of hash symbols indicates the level of the header. While not strictly necessary, adding a blank line after a header can make the document easier to read in its plain-text format.

```
# Header level 1

## Header level 2

### Header level 3
```

In technical documents, try to limit subheading levels to maintain clarity; usually, three levels are sufficient.

6.3 Lists

Ordered lists use numbers followed by a period and a space. Unordered lists use asterisks (`*`), plus signs (`+`), or hyphens (`-`) followed by a space.

6.3.1 Ordered list

My preference is to use `1.` for each ordered item and let the computer renumber them because it makes adding and removing items faster, and guarantees that there will never be a number skipped.

```
1. First item
1. Second item
1. Third item
```

6.3.2 Unordered list

My preference is to use `*` for the first level of unordered items.

```
* Item one
* Item two
* Item three
```

6.4 Links

Create links using square brackets for the link text followed by parentheses for the URL.

```
[Download](https://keenwrite.com)
```

6.5 Tables

Tables use pipes (`|`) to indicate separate columns and hyphens (`-`) to mark the header row. Markdown does not define a syntax for table footers.

```
| Column A | Column B | Column C |
|----------|----------|----------|
| Value 1  | Value 2  | Value 3  |
| Value 4  | Value 5  | Value 6  |
```

Table cell justification can be suggested using a colon (`:`) on the left, right, or both sides (centering) of the header separator line.

```
| Column A   | Column B  |  Column C |
|:-----------|:---------:|----------:|
| Value 1    |  Value 2  |   Value 3 |
| Value 4    |  Value 5  |   Value 6 |
```

The colon location is a hint to the presentation layer, which has the freedom to ignore it.

7 Extended Markdown

This chapter covers advanced formatting features that enhance document structure and presentation. These features help create professional documents with rich content organization, proper cross-referencing, and typographic excellence.

7.1 Fenced divs

Fenced divs are transformed into XHTML `div` elements. They allow semantic organization and styling beyond standard Markdown.

7.1.1 Basic syntax

A fenced div starts with at least three colons (`:::`), followed by one or more spaces, and then any word. Content follows immediately on the next line. Fences must be terminated with at least three colons alone on a line. For example:

```
::: name
Content
:::
```

The output resembles:

```
<div class="name">
<p>Content</p>
</div>
```

7.1.2 Extended syntax

Fenced divs have an extended syntax for more influence over the presentation. The syntax allows for a unique identifier (marked with a hash, `#`), multiple classes (each starting with a period, `.`), and key-value pairs (of the form `name=value`):

```
::: {#poem-01 .stanza author="Emily Dickinson" year=1890}
Because I could not stop for Death --
```

```
He kindly stopped for me --
The Carriage held but just Ourselves --
And Immortality.
:::
```

This produces:

```
<div id="poem-01" class="stanza" data-author="Emily Dickinson" data-year="1890">
<p>Because I could not stop for Death -
He kindly stopped for me -
The Carriage held but just Ourselves -
And Immortality.</p>
</div>
```

While this syntax is not standard Markdown (i.e., CommonMark), it is supported by many tools, including pandoc.

7.1.3 Nested syntax

Fenced blocks may be nested to create hierarchical document structures:

```
::: poem
:::::: stanza
Because I could not stop for Death --
He kindly stopped for me --
The Carriage held but just Ourselves --
And Immortality.
::::::
:::
```

This produces nested `div` elements:

```
<div class="poem"><div class="stanza">
<p>Because I could not stop for Death -
He kindly stopped for me -
The Carriage held but just Ourselves -
And Immortality.</p>
</div></div>
```

Such nesting can help with complex presentation logic.

7.2 Code blocks

In technical documentation, source code snippets help explain algorithms. An example snippet syntax is given in the following quintessential BASIC program:

```
``` basic
10 print "hi"
20 goto 10
```
```

The word `basic` suggests that the presentation layer colourise the syntax according to the rules of the BASIC programming language. Code blocks are also used to draw diagrams using text-based syntaxes. See **Section 8.3** for details.

7.3 Inline classes

Inline classes can be assigned to words or phrases. This allows presentation logic to stylize words based on their classification. The general form of inline classes follows:

```
[word]{.class}
```

For example:

```
[radiation]{.glossary .warning}
```

The given example outputs the following XHTML:

```
<span class="glossary warning">radiation</span>
```

Note the semantic `warning` class instead of a direct colour class such as `red`. This allows the colour to be changed at the presentation layer without the class name becoming stale.

7.4 Captions and cross-references

Captions provide context and descriptions for tables, figures, equations, and so forth. Cross-references allow linking to captioned items throughout the document.

7.4.1 Caption syntax

A caption *must* start with a double colon (`::`) and *must* be separated by a blank line from the item it describes.

For images:

```
![image title](https://example.com/image.png)

:: Figure caption text
```

For tables:

```
| a | b | c |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

:: Table caption text
```

For equations:

```
$$E = mc^2$$

:: Equation caption text
```

Note that this syntax is not supported by any other Markdown tool.

7.4.2 Cross-references

Cross-references use anchor type names and unique identifiers such as:

- `{#type-name:id}` – defines an anchor name
- `[@type-name:id]` – references the anchor

Example image with cross-reference:

```
There is no cuter animal than the one in [@fig:animal].

![image title](https://placecats.com/500/300)

:: World's cutest animal {#fig:animal}
```

Anchors can include any type name and identifier, such as:

```
{#fig:cats}
{#图版:猫}
{#eq:mass-energy}
{#eqn:laplace}
```

7.4.3 Predefined type names

To simplify writing captions and references, predefined type names correspond to labels commonly used in documents. **Table 7.1** lists predefined cross-reference type names along with their associated labels. Labels are inserted into the document automatically.

Table 7.1 Cross-reference types and labels

Type name	Label
algorithm, alg	Algorithm
equation, eqn, eq	Equation
figure, fig	Figure
formula	Formula
listing, list, lst, source, src	Listing
lyric	Lyrics
music, score	Score
table, tbl, tab	Table

You may add your own labels as follows:

1. Edit `xml-references.tex` in the `xhtml` theme subdirectory.
2. Insert a new line within the `ReferenceLabels` data set:
   ```
   \setdataset[ReferenceLabels][en][
     typename=label,
   ]
   ```
3. Ensure there's a trailing comma (`,`).
4. Save the file.

7.5 Quotation marks

KeenWrite integrates KeenQuotes, a library for curling straight quotes in both British and American English, with a high degree of accuracy. To achieve this, all straight quotes are examined throughout the document and, based on context, emitted as a particular entity encoding.

Understanding quotation mark encoding helps ensure proper typography in both digital and print formats. Different encoding options provide flexibility for various output requirements. **Table 7.2** lists available encoding choices.

Table 7.2 Single straight quote encoding options

Option	Encoding	Description
regular		Do not encode
apos	`&apos;`	Curled when typeset to PDF
aposhex	`&#x27;`	Apostrophe's numeric value
quote	`&rsquo;`	Right single quotation mark
quotehex	`&#8217;`	Right single quotation mark's numeric value
modifier	`&#x2bc;`	The modifier letter apostrophe

When typesetting to PDF, only the semantically correct `&apos;` value will be curled. Authors may enter quotation marks by hand (“”), which will typeset as expected.

7.5.1 History

The origins of quotation marks trace back to scholarly annotation practices in Ancient Greece. Greek scribes used marginal symbols like the diplé (〉) to highlight notable passages. A variant, the diplé periestigmene (⪢)—a diplé accompanied by dots—was used by scholars to indicate disagreement with earlier interpretations. These marks laid the groundwork for distinguishing quoted text.

By the seventeenth century, quotation marks appeared regularly in printed texts. Western European typographers in the nineteenth century adopted the convention of turning quotation mark pairs outward, creating the familiar “double curved” style.

Early mechanical typewriters lacked many punctuation marks. Straight single and double quotes served multiple roles: quotation marks, apostrophes, feet and inches,

and mathematical primes. Computer character sets later codified straight quotes (such as ASCII character 34, `"`), creating ongoing typographical challenges.

Modern standards recommend using the right single quotation mark (`’`) for apostrophes, though this creates semantic ambiguity since the same character also closes dialogue. Consider the sentence:

> Ambiguity lurks in "’cause the horses’".

The meaning of `'cause` (*because* vs. *induce*) determines whether to use an opening quote or apostrophe; historical decisions still impact modern typography.

7.6 Summary

Table 7.3 summarizes the syntaxes described in this chapter.

Table 7.3 Extended syntax

Syntax	Meaning	Marks
`::: name ¶ :::`	Fenced div	Triple colons
``` ``` name ¶ ``` ```	Code block	Triple backticks
`[word]{.class}`	Inline class	Brackets and braces
`{#type:id}`	Cross-reference definition	Braces and hash
`[@type:id]`	Cross-reference anchor link	Brackets and at

Note that the pilcrow (`¶`) means that the opener is separated from its closer by one or more blank lines (i.e., paragraphs).

8 Images

This chapter covers how to insert and manage document images. You can embed standard images, create text-based diagrams, and—as we'll see later—integrate plots from R. The application supports various image formats and provides flexible path resolution for both local and remote images.

8.1 Inserting images

Insert an image into your Markdown document using the following syntax:

```
![image description](path)
```

Replace `image description` with a descriptive caption for the image and `path` with either a local file path or URL where the image is located.

For example, to embed a local image file named `chart.png` located in the same directory as your document, you could write:

```
![Sales Chart](chart.png)
```

To embed an image from a remote server, use its URL:

```
![Company Logo](https://keenwrite.com/images/logo/title.svg)
```

The application will display the image in the preview tab upon download.

8.2 Image server

By default, all text diagrams are sent to **Kroki** for conversion into SVG images. A network connection is required to render all text diagrams. There is a configuration option to change servers and Kroki is open-source software, so it is possible to keep all source diagrams on premises by setting up a locally hosted image server.

8.3 Text diagrams

KeenWrite distinguishes between diagrams and source code listings by prefixing diagrams with a `diagram-` prefix, such as:

```
``` diagram-plantuml
@startuml
Alice -> Bob: Hello
@enduml
```
```

This has a few benefits. First, it allows rendering diagram types using a service without having to codify all possible diagram types. Second, when a new type of diagram is added, it's available immediately without needing to upgrade the text editor. Third, it clearly distinguishes between a code block to be rendered visually and one to be listed as verbatim source code.

Typically, source code is presented in code blocks that include the language name so that syntax highlighting can be applied:

```
``` c
main() {
    printf( "hello, world" );
}
```
```

8.3.1 Mermaid

GitHub created a *de facto* standard that prevents parsers from dynamically distinguishing between a diagram to render and source code to list. The following code block could be either a source code listing or a pie chart:

```
``` mermaid
pie
    title Turkish Empire Proportions, 1789
    "Asia" : 66
    "Africa" : 20
    "Europe" : 14
```
```

While tagging the block as `mermaid-lang` could help, it creates a special case that needs to be programmed into Markdown parsers. (Having both `c-lang` and `c` is redundant.) The `diagram-` prefix side-steps the issue while keeping true to the

human-readability nature of Markdown. For this reason, using `mermaid` alone for a fenced code block will show the source code rather than draw a diagram.

8.4 Image paths

Image file discovery enables flexible and intelligent handling of image links within documents. It supports both remote and local image sources, and attempts to resolve incomplete or relative paths.

8.4.1 Resolution strategy

The algorithm follows a multi-step process to determine the file to display:

1. **Remote URLs** – If the image reference begins with a remote protocol (e.g., `http://`, `https://`), it is accepted as valid and used directly.
2. **Fully qualified paths** – If the image refers to a specific file path on the local system and the file is readable, it is used as-is.
3. **Relative paths** – For unqualified links, the system checks if the image exists relative to the base directory of the current document. If not found, and there's no extension, all common image file name extensions (e.g., `.svg`, `.png`, `.jpg`) are tried.

If the image cannot be resolved through any of the above methods, an error is logged to indicate the missing file.

8.4.2 Resolution example

A document that includes `![Logo](company-logo)` is resolved as follows:

1. If `company-logo` is a remote URL, try to fetch it.
2. Look for a local file named `company-logo`.
3. Look for a file named `company-logo` in a specified images directory.
4. Look for `company-logo.png`, `company-logo.jpg`, etc.
5. Use the first valid readable matching file found.
6. Log an error if the file cannot be found.

9 Exporting documents

KeenWrite can convert Markdown documents to XHTML, PDF, and plain text formats. The application has conveniences for concatenating multiple files together prior to typesetting.

9.1 Collating

When exporting multiple documents, the application uses an alphanumeric sorting algorithm to determine the order files are concatenated prior to processing. This means files are collated in a natural order that honours both alphabetic and numeric sequences within file names.

Consider the following file names:

```
chapter_1.md
chapter_2a.md
chapter_3.md
chapter_10.md
```

Without alphanumeric sorting, `chapter_10.md` would incorrectly appear before `chapter_2a.md` in a standard alphabetic sort because the computer would otherwise compare the *text* `1` vs. `2` rather than the *numbers* `10` vs. `2`.

9.2 Batch Processing

The application can process a single document or multiple documents. Batch processing automatically discovers and processes additional files based on the actively edited file's location and extension.

9.2.1 Single document export

Export a single document as follows:

1. Open the document to export.
2. Click **File → Export As → PDF** (or type Ctrl+p).

3. Select a theme from the drop-down list.
4. Set the output file name.
5. Click **Save**.

The document is exported.

9.2.2 Multi-document export

Export multiple documents into a single, concatenated file:

1. Open any document in the directory containing files to export.
2. Click **File → Export As → Joined PDF** (or type `Ctrl+Shift+p`).
3. Select a theme from the drop-down list.
4. Set the output file name.
5. Click **Save**.

The application searches recursively through the directory hierarchy starting from the actively edited file's location. All files with the same extension are discovered, sorted using the alphanumeric algorithm, and concatenated prior to exporting.

If the first file processed contains numeric digits in its name, only files containing numeric digits will be included in the export. If the first file contains no digits, all matching files will be processed in alphabetical order.

9.2.3 Chapter ranges

Chapter ranges can be restrict exporting to specific chapters. The range specification supports several formats:

- Individual chapters: `1,3,5` exports only chapters 1, 3, and 5.
- Range: `1-5` exports chapters 1 through 5, inclusive.
- Starting range: `-5` exports chapters 1 through 5.
- Open-ended range: `3-` exports chapter 3 and all subsequent chapters.
- Multiple ranges: `1,3-7,10-` exports chapter 1, chapters 3 through 7, and chapter 10 onwards.

9.3 XHTML exports

The application produces standards-compliant XHTML documents that can be viewed in web browsers or further processed. XHTML export features include:

- Document metadata is added to the `<head>` tag as `<meta>` tags
- Diagrams and visualizations are embedded as `<image>` elements
- Cross-references and internal links are preserved
- Mathematical expressions can be output in multiple formats

Mathematical expressions in XHTML exports can be rendered as:

- TEX notation for processing by KaTEX or MathJax; or
- SVG images for direct display without requiring additional libraries.

9.4 PDF exports

PDF exports use the ConTEXt typesetting system to produce high-quality output. Mathematical expressions are rendered directly by TEX-based software, ensuring optimal typography for technical content. Before exporting to PDF format, be sure to have installed the theme pack and ConTEXt. See **Chapter 2** for details.

9.5 Theme selection

Themes control the visual appearance of exported PDF documents, including fonts, spacing, colours, and layout. The application includes several predefined themes, each optimized for different document types and purposes.

Some themes include:

- ***Boschet*** – Based on Baskerville font with elegant styling.
- ***Handrit*** – Double-spaced manuscript format using a courier font.
- ***Tarmes*** – Based on Times Roman font with minimal styling.
- ***TEXomus*** – A format suitable for technical documentaion.

Theme selection is available during PDF export through the theme drop-down menu in the export dialog. Each theme may have specific font requirements that must be installed separately.

Create themes by copying and modifying existing ones (e.g., *TEXomus* or *Tarmes*).

10 Themes

Themes in KeenWrite define the visual appearance and layout of typeset PDF documents. They control elements fonts, spacing, colours, page dimensions, and structural formatting. Users can achieve high-quality outputs without modifying document content. Themes are based on ConTEXt setups and can be selected during PDF export to match different document purposes, such as manuscripts, user manuals, or wills.

10.1 Customization

Users may define unique document appearances by creating new theme setups. This involves structuring a theme directory with required files to integrate with the application's typesetting process.

All themes must have:

- a separate directory;
- a main entry point;
- a reference to XHTML setups; and
- a properties file.

For example, a new theme named "dīvus" would have the following file structure, relative to the themes directory:

```
dīvus/
├── main.tex
├── xhtml.tex
└── theme.properties
```

10.1.1 Theme directory

The theme directory organizes all files for a single theme. For consistency, the theme directory name:

- must not include spaces;
- must a sibling directory of all other themes;

- should be lowercase; and
- should be a single word (Latin works well).

10.1.2 Entry point

The theme's main entry point defines how the document is presented during typesetting. The main entry point, `main.tex`, is passed to the typesetting software by the editor immediately prior to typesetting. The purpose of the entry point is to import all the other files that define how the document is to be presented.

Minimally, the file must include the XHTML setups file, which may be accomplished with the following line:

```
\input xhtml
```

10.1.3 XHTML setups

XHTML setups map document elements to typesetting commands. To typeset the document, the editor first converts the content into a data format known as XHTML. Using XHTML allows mapping common document elements to commands that the typesetting software can execute to format the output document.

A file named `xhtml.tex` must exist alongside the main file (that is, in the same directory). The file must contain the following:

```
\usesubpath[../xhtml]
\input ../xhtml/setups
```

These commands instruct the typesetter to import instructions from the `xhtml` directory, which must exist in the parent directory. The XHTML setups are responsible for translating XML elements from the XHTML document into ConTEXt macros.

10.1.4 Properties file

The `theme.properties` files identify a theme's name to the application. This file allows the editor to display a drop-down having user-friendly names. The format of the file follows:

```
name=Theme Name
```

The `name` is the key that the editor uses and the `Theme Name` is the value that the theme's author may assign. For the dīvus example, the file would resemble:

```
name=Dīvus
```

For consistency, the theme name:

- should use title case (start words with an uppercase letter);
- should be short (will be truncated to 20 characters);
- must be uniquely named; and
- must not contain punctuation (letters and numbers only).

If a theme does not contain this file, the theme will be unavailable for users to select when exporting.

10.2 Text replacement

Sometimes we want to bend typography to our will. KeenWrite themes provide a mechanism to leverage the power of TEX to change the typesetting for particular words without adding hints to the source document. This truly exhibits the power of moving presentation logic elsewhere.

Lua is a programming language that is tightly integrated with ConTEXt. Here's an example of using Lua to create a list of replacements:

```
\startluacode
userdata = userdata or {}

userdata.TextReplacements = {
  [1] = { "a.m.", "\\cap{am}" },
  [2] = { "TeX", "\\TeX{}" },
}
\stopluacode
```

This will replace `a.m.` with its small caps version and all occurrences of `TEX` with its iconic lowered-E formatting. Replacements are case-sensitive. Most themes, including *Boschet*, list replacement rules in a file called `replace.tex` located in the theme's subdirectory.

10.3 Resources

Styling themes with ConTEXt macros is beyond this manual's scope. Refer to the theme source code for examples. Additionally, the following links are excellent resources for support:

- **ConTEXt Mailing List**
- **TEX Stack Exchange**

See **Chapter B** for a library of books on ConTEXt.

11 Math

Mathematical expressions are typeset using TEX notation, a powerful system developed by Dr. Donald E. Knuth for producing documents with complex mathematical and scientific content. In TEX, you write plain text commands that describe the structure and formatting of your document, and then compile them to generate the final output.

KeenWrite supports only plain TEX, allowing usage of either ConTEXt or LaTEX.

11.1 TEX notation

All TEX commands start with a backslash (`\`), followed by the name of the command, and often use braces (`{}`) for any arguments.

Some simple mathematical elements using standard TEX syntax include:

- Fractions: `\fraction{numerator}{denominator}` (or `\frac{}{}`)
- Square roots: `\sqrt{expression}`
- Superscripts: `x^2`
- Subscripts: `x_1`
- Greek letters: `\alpha`, `\beta`, `\gamma`, etc.
- Integrals: `\int`
- Summations: `\sum`
- Limits: `\lim_{x \to 0}`

More complex expressions combine these elements:

```
\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}

:: Basel problem
```

Produces:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} \qquad \text{(Basel problem)}$$

When exported to XHTML, mathematical expressions can be output as TEX notation or SVG images. Using TEX notation allows rendering the mathematics using a JavaScript library such as KaTEX or MathJax. When exported to PDF, the typesetting system renders expressions directly.

11.2 Sigil syntax

A sigil is a special symbol used to mark or identify different types of elements, much like how medieval magicians and alchemists used them to represent spirits, forces, or intentions. Rooted in occult traditions, sigils served as visual signatures—condensed symbols encoding names, ideas, or goals in a hidden or abstract form.

Mathematical expressions must be enclosed within sigils. Spaces are not allowed after the opening sigil or before the closing one.

11.3 Inline expressions

Enclose inline TEX code inside single dollar sign sigils (`$`):

```
The quadratic formula is $x=\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.
```

Outputs an equation within a sentence:

> The quadratic formula is $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

11.4 Standalone expressions

Enclose standalone TEX code inside double dollar sign sigils (`$$`):

```
$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$
```

Outputs a lone paragraph:

> $$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi} \tag{11.1}$$

The assigned equation reference number will differ, of course.

12 R

This chapter describes how to use R within the application to add dynamic content to documents. It covers integration basics, environment setup, creating visualizations, working with external data, and using YAML variables in R code.

12.1 Integration

R integration allows embedding R code directly into documents for dynamic content generation, such as statistical analysis and data plots. The application supports R Markdown (`.Rmd`) files, where R code can be executed inline or in chunks to produce results, including numerical outputs and visualizations. KeenWrite's R integration is largely compatible with the syntax for pandoc / knitr.

12.2 Environment

Setting up the R environment involves configuring scripts and directories to load custom functions and manage file locations. This ensures R code runs smoothly within the application.

12.2.1 Bootstrapping

A bootstrap script loads custom R libraries or functions at startup. Complete the following steps to call an R function from your own library:

1. Click **File → New** to create a new file.
2. Click **File → Save As**.
3. Browse to your home directory.
4. Set **Name** to: `library.R`.
5. Click **Save**.
6. Set the contents to:

```
sum <- function( a, b ) {
  a + b
}
```

7. Click the **Save** icon.
8. Click **R → Script**.
9. Set the **R Startup Script** contents to:
   ```
   source( 'library.R' );
   ```
10. Click **OK**.
11. Create a new file.
12. Set the contents to:
    ```
    `r#sum( 5, 5 )`
    ```
13. Save the file as `math.R`.

The preview tab shows the result of calling the `sum` function:

> 10.0

This shows how the bootstrap script can load `library.R`, which defines a `sum` function that is called by name in the Markdown document.

12.2.2 Working directory

The working directory determines where R searches for source files. Accomplish this as follows:

1. Click **R → Directory**.
2. Set **Directory** to a different directory.
3. Click **OK**.
4. Create the directory if it does not exist.
5. Move `library.R` into the directory.
6. Append a new function to `library.R` as follows:
   ```
   mul <- function( a, b ) {
     a * b
   }
   ```
7. Click **R → Script**.
8. Set the **R Startup Script** contents to:
   ```
   setwd( v$application$r$working$directory );
   source( "library.R" );
   ```
9. Change `math.Rmd` to:
   ```
   `r#mul( 5, 5 )`
   ```

10. Close the file `math.Rmd`.
11. Confirm saving the file when prompted.
12. Re-open `math.Rmd`.

The preview tab shows:

> 25.0

Calling `setwd` using `v$application$r$working$directory` changes the working directory where the R engine searches for source files.

12.3 Data visualizations

Data plots can be generated using R. First, create a new file:

1. Start the application.
2. Click **File → New** to create a new file.
3. Click **File → Save As**.
4. Set **Name** to: `plot.Rmd`
5. Click **Save**.

Setting the file name extension tells the application what processor to use when transforming the contents. To insert visual content, such as a plot, use the knitr-compatible syntax of ```` ```{r} ```` and ```` ``` ```` to enclose multi-line R snippets:

````
```{r}
x <- seq( 0, 2 * pi, length.out=100 )
y <- sin( x )
plot( x, y, type="l" )
```
````

12.4 Variable definitions

Variables provide a way to define reusable values that can be referenced dynamically within R code. To see how variable definitions work in R, try the following:

1. Create a new file.
2. Change the contents to (use spaces to indent, not tabs):

```
project:
  title: Project Title
  author: Author Name
```

3. Save the file as `definitions.yaml`.
4. Click **File → Open**.
5. Set **Source Files** to **Variable Files**.
6. Select `definitions.yaml`.
7. Click **Open**.
8. Open `math.Rmd` if it is not already open.
9. Type: `je`
10. Press Ctrl+Space.

The editor inserts the following text (matches `je` against Pro**je**ct):

```
`r#x( v$project$title )`
```

The preview tab shows:

> 25.0

This is because the application inserts variable reference names based on the type of file being edited. By default, the R engine does not have a function named `x` defined. The `x` function performs fail-safe string conversions.

Continue as follows:

1. Click **R → Script**.
2. Append the following:

```
x <- function( s ) {
  tryCatch( {
    r = eval( parse( text = s ) )

    ifelse( is.atomic( r ), r, s );
  },
  warning = function( w ) { s },
  error = function( e ) { s } )
}
```

3. Click **OK**.
4. Close and re-open `math.Rmd`.

The preview tab shows:

> 25.0
>
> Project Title

The `x` function attempts to evaluate the expression defined by the variable. This means that the variables can also include expressions that R is capable of evaluating.

Externally defined variables are assigned to the `v` object in R code. This allows dereferencing them using a `$`-separated string. The moustache and dot-separated notation also works (e.g., `{{project.title}}`), but better to be consistent.

12.5 Comma-separated values

Many R functions are included, like one that converts comma-separated values to tables. Consider travel fares saved as `fares.csv` alongside the main document:

```
Distance, "Fare ($)"
26.7,         73.75
16.4,         56.00
32.7,         83.25
77.0,        190.50
```

Call `csv2md` using a path relative to the R working directory, such as:

```
`r# csv2md( '../docs/fares.csv' );`

:: Taxi fares vs. distance
```

Convert the document from R Markdown to Markdown from the command line:

```
keenwrite.bin \
  --r-script="${HOME}/writing/R/bootstrap.R" \
  --r-dir="${HOME}/writing/R" \
  -i 00.Rmd \
  -o 00.md
```

The output file, `00.md`, contains the result from running the function:

```
|Distance| Fare ($)|
|:---|---:|
|26.7| 73.75|
|16.4| 56.00|
|32.7| 83.25|
|77|190.50|
|Totals|481.65|

:: Taxi fares vs. distance
```

By isolating the data to an external file, it is possible to update the file and rebuild the document to insert the latest values.

13 Graphical user interface

This chapter covers customizing the graphical user interface (GUI).

13.1 Skins

A skin defines the visual style of the application. Skins, which use a syntax based on cascading style sheets, configure the user interface colours, fonts, spacing, highlights, drop-shadows, gradients, and so forth. See the **W3C CSS tutorial**.

13.1.1 Order

The order that stylesheets are applied dictates how styles get overriden. The application loads and applies stylesheets in the following order:

1. **scene.css** – Defines toolbar styling.
2. **markdown.css** – Defines text editor styling.
3. **skins/skin_name.css** – Main application chrome, extensible.
4. **custom.css** – User-defined preferences.

13.1.2 Customization

This section walks through creating and applying a custom skin step-by-step, to personalize the application's appearance. Create a custom skin as follows:

1. Start the application.
2. Click **File → New** to create a new file.
3. Click **File → Save As** to rename the file.
4. Save the file as `custom.css`.
5. Change the content to the following:

```
.root {
  -fx-base: rgb( 30, 30, 30 );
  -fx-background: -fx-base;
}
```

Apply the skin as follows:

1. Click **Edit → Preferences** to open the preferences dialog.
2. Click **Skins** to view the available options.
3. Click **Browse** to select a custom file.
4. Browse to and select `custom.css`, created previously.
5. Click **Open**.
6. Click **Apply**.

The user interface immediately changes to a dark mode. Continue:

1. Click **OK** to close the dialog.
2. Change the **rgb** numbers in **custom.css** from `30` to `60`.
3. Click **File → Save** to save the CSS file.

The user interface immediately changes colour.

13.1.3 Classes

For more control, this section explains how to use pre-existing skin templates with defined classes, making it easier to tweak the GUI. Accomplish this as follows:

1. Visit the **skin** repository directory
2. Click one of the files (e.g., `haunted_grey.css`).
3. Click **Raw**.
4. Copy the entire text.
5. Return to `custom.css`.
6. Delete the contents.
7. Paste the copied text.
8. Save the file.

To see how the CSS styles are applied to the text editor, open **markdown.css**, which is also in the repository.

13.1.4 Modena

The basic look used by the application is *Modena Light*. Typically we only need to override a few classes to completely change the application's look and feel. For a full listing of available styles see the OpenJDK's **Modena CSS file**.

13.1.5 JavaFX CSS

JavaFX CSS has styling capabilities beyond regular CSS. The **Java CSS Reference Guide** shows differences between JavaFX CSS and W3C CSS. The guide covers functions for manipulating colours and gradients using existing colour definitions.

13.1.6 RichTextFX

The application uses RichTextFX to render the text editor. Styling various text editor classes can require using the prefix `-rtfx` instead of the regular JavaFX `-fx`.

13.2 Fonts

The preview tab uses two categories of fonts: regular and monospace.

13.2.1 Regular font

To change the regular preview font, complete the following steps:

1. Click **Edit → Preferences**.
2. Click **Fonts**.
3. Click **Change** under **Preview Font** for the **Preview tab font name**.
4. Find the font name by typing or scrolling.
5. Click the desired font family.
6. Click **OK**.
7. Click **Apply**.

The regular preview font is changed.

13.2.2 Monospace font

To change the monospace preview font, complete the following steps:

1. Click **Edit → Preferences**.
2. Click **Fonts**.
3. Click **Change** under **Preview Font** for the **Monospace font name**.

4. Find the font name by typing or scrolling.
5. Click the desired font family.
6. Click **OK**.
7. Click **Apply**.

The monospace font is changed.

13.3 Internationalization

Internationalization support is essential for editing and previewing text in multiple languages. Both fonts and language must be set for non-Latin-based text. **Table 13.1** lists example Chinese-Japanese-Korean (CJK) font settings for the editor and preview tabs.

Table 13.1 Editor and preview tab font families

Tab	Font Family	Language
Editor	Noto Sans CJK KR	Korean
Editor	Noto Sans CJK JP	Japanese
Editor	Noto Sans CJK HN	Chinese
Editor	Noto Sans CJK SC	Simplified Chinese
Preview	Noto Serif CJK KR	Korean
Preview	Noto Serif CJK JP	Japanese
Preview	Noto Serif CJK HN	Chinese
Preview	Noto Serif CJK SC	Simplified Chinese

13.3.1 Editor font

Follow these steps to change the editor font:

1. Click **Edit → Preferences**.
2. Click **Fonts**.
3. Click **Change** under **Editor Font**.
4. Find the font name by typing or scrolling.
5. Click the desired font family.
6. Click **OK**.
7. Click **Apply**.

The text editor font is changed.

Note the following:

- Install the desired fonts (restart the application).
- Enter font name directly if it cannot be selected.

13.3.2 Language Settings

Language settings control the locale used by the application, which is important when using CJK fonts or other locale-specific features.

Change the locale as follows:

1. Click **Edit → Preferences**.
2. Click **Language**.
3. Select a value for **Locale**.
4. Click **Apply**.

The language is set.

14 Command-line interface

This chapter covers how to use the application from the command line to convert Markdown files to various output formats. Without specifying any command-line arguments, the application will launch a graphical user interface.

14.1 Common arguments

The most frequently used command-line arguments include:

- `-h` – displays all command-line arguments, then exits.
- `-i` – sets the input file name, must be a full path.
- `-o` – sets the output file name, can be a relative path.
- `-s` – sets a variable name and value at build time (dynamic data).

Setting a variable name on the command line useful, for example, to pass in a product guide version number.

14.2 Example usage

The following invocations will convert `01.md` into the respective file formats, determined by the file name extension. In the first case, it will become an XHTML page. In the second case, it will become a plain-text document with all variables interpolated and replaced.

```
keenwrite.bin \
  -i "${HOME}/document/01.md" \
  -o document.xhtml

keenwrite.bin \
  -i "${HOME}/document/01.md" \
  -o document.txt \
  -v "${HOME}/document/variables.yaml"
```

The following invocation will convert `01.Rmd` to a PDF file and replace the metadata using values from the variable definitions file.

```
keenwrite.bin \
  -i "${HOME}/document/01.Rmd" \
  -o document.pdf \
  --image-dir="${HOME}/document/images" \
  -v "${HOME}/document/variables.yaml" \
  --metadata="title={{book.title}}" \
  --metadata="author={{book.author}}" \
  --r-dir="${HOME}/document/R" \
  --r-script="${HOME}/document/R/bootstrap.R" \
  --theme-dir="${HOME}/.local/share/keenwrite/themes/boschet"
```

Directory or file names containing spaces must be quoted. For example, on Windows:

```
keenwrite.bin -i "C:\Users\My Documents\01.Rmd" -o document.pdf
```

15 Troubleshooting

This chapter addresses common issues that may arise while using KeenWrite. When problems occur, the status bar displays a brief summary of a problem. Detailed information is available through the log viewer.

15.1 Log

The application maintains a list of error messages to help diagnose issues.

Click **View → Log** (or press F12) to open the log view.

The log window shows various types of information:

- Error messages that indicate specific problems
- Warning messages about potential issues
- Information about file operations and processing
- System events and application state changes

When reporting issues or seeking help, the log information can provide valuable context about what the application was doing when problems occurred.

15.2 Container

If you encounter the following error (or similar) on Windows:

> Error: unable to load machine config file: "json: cannot unmarshal string into Go struct field MachineConfig.ImagePath of type define.VMFile"

Complete the following steps to remove obsolete containers:

1. Stop KeenWrite.
2. Uninstall podman.
3. Delete `C:\Users\%USERNAME%\.config\containers\`.
4. Delete `C:\Users\%USERNAME%\AppData\Roaming\containers\`.
5. Open a command prompt.

6. Run:
   ```
   wsl -l -v
   wsl --unregister podman-machine-default
   podman -v
   podman machine reset -f
   ```
7. Reboot.
8. Start KeenWrite.
9. Retry exporting as a PDF file.

Note that downloading the typesetter image can take several minutes and does not show a progress indicator. Be patient.

15.3 SVG compatibility

Vector graphics files created in certain applications may encounter compatibility issues when embedded in documents. These problems typically manifest as rendering failures, missing content, or incorrectly displayed graphics.

When referencing a vector graphic using Markdown, the status bar may show the following error:

> The current document is unable to create an element of the requested type (namespace: http://www.w3.org/2000/svg, name: flowRoot).

This error occurs due to a version mismatch of the `flowRoot` element that Inkscape creates. Resolve the issue by changing the SVG version number as follows:

1. Edit the vector graphics file using any text editor.
2. Find `version="1.1"`.
3. Change the version to: `version="1.2"`.
4. Save the file.

The SVG will now appear inside the application; however, the text may appear as black blocks.

15.3.1 Black blocks

Depending on how text is added to a vector graphic in Inkscape, the text may be inserted within an element called a `flowRoot`. Although `flowRoot` is recognized,

the contents may not be fully interpreted, resulting in black blocks being drawn instead of text, as shown in **Figure 15.1**.

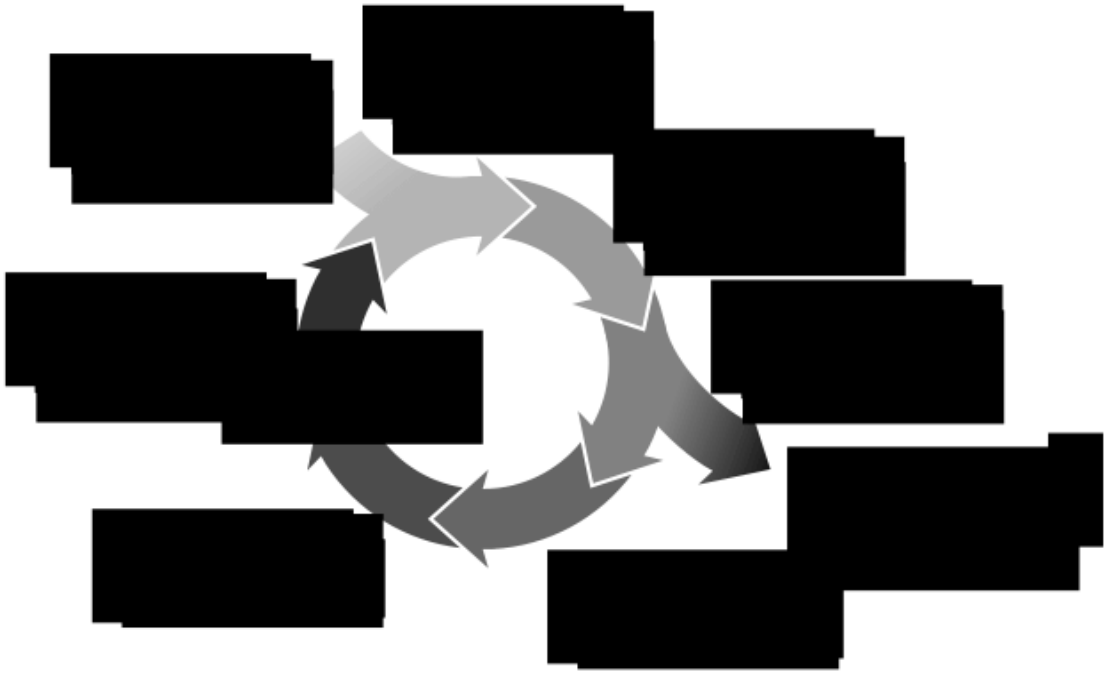

Figure 15.1 Blocked text

Resolve the issue by “unflowing” all text elements as follows:

1. Start Inkscape.
2. Load the SVG file.
3. Select all the text elements.
4. Click **Text → Unflow**.

The text may change size and position; recreate the text without dragging using the text tool. After all the text areas have been recreated, continue as follows:

1. Click **Edit → XML Editor**.
2. Expand the **XML Editor** to see more elements.
3. Delete all elements named `svg:flowRoot`.
4. Save the file.

When the illustration is reloaded, the black blocks will have disappeared, but the text elements may ignore any assigned colour.

15.3.2 Blackened text

When an SVG `style` attribute contains a reference to `-inkscape-font-specification`, all values that follow said reference are ignored. This results in black text, shown in **Figure 15.2**.

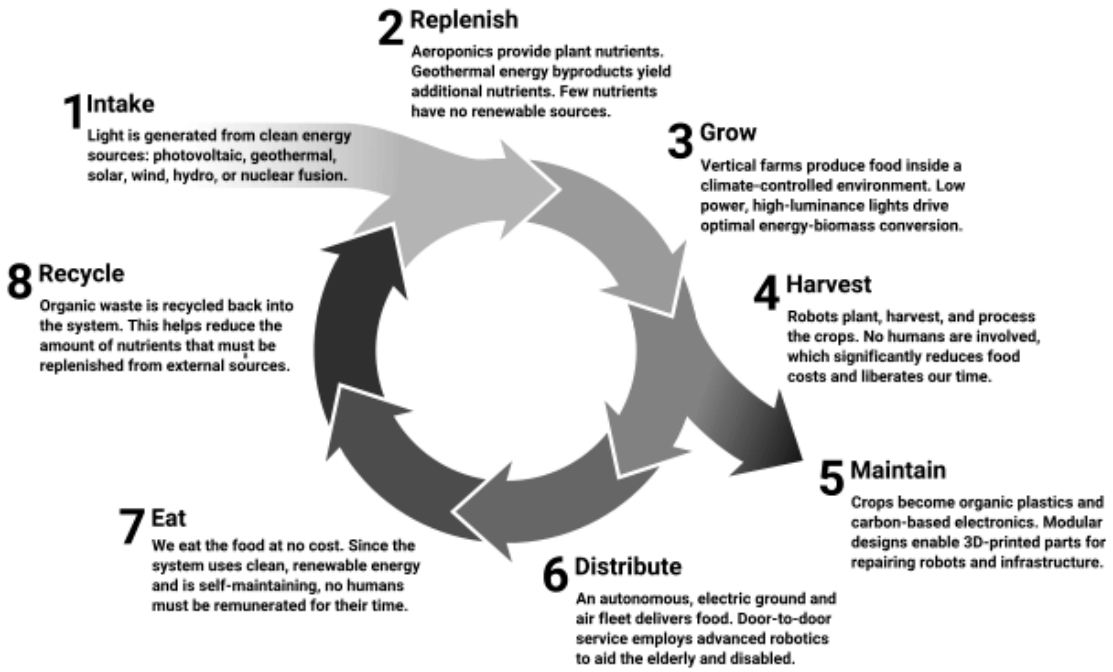

Figure 15.2 Blackened text

Resolve the issue of colourless text as follows:

1. Open the SVG file in a plain-text editor.
2. Remove all references `-inkscape-font-specification:'<FONT>';`, including the trailing (or leading) semicolon.
3. Save the file.

Figure 15.3 shows colours having reappeared after reloading.

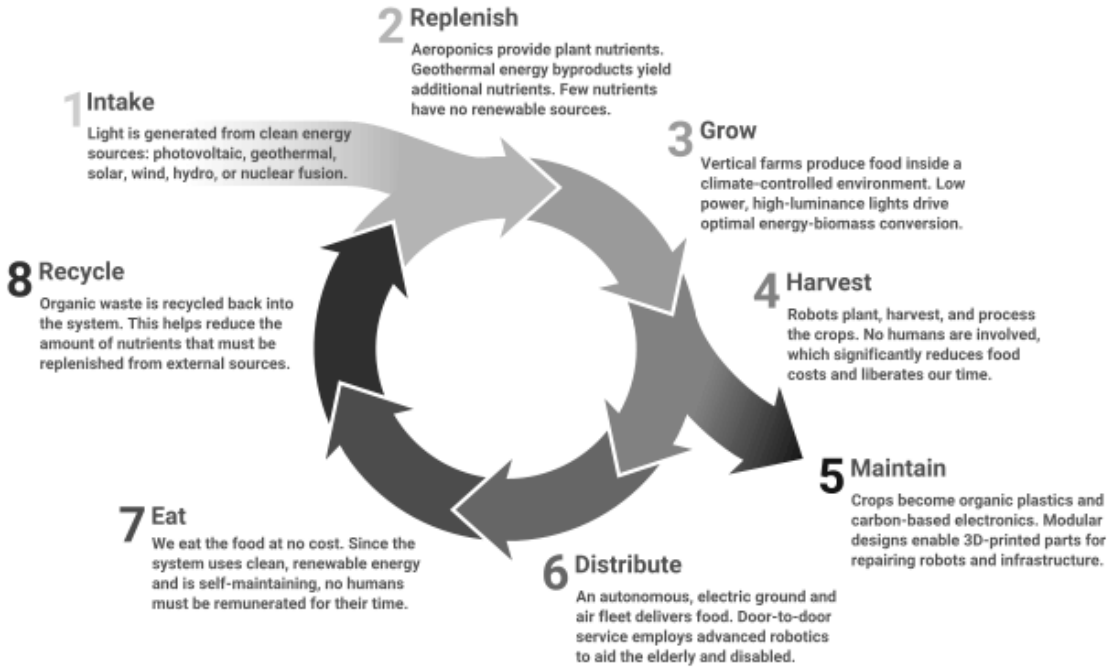

Figure 15.3 Resolved text

16 Open-source software

KeenWrite is free, open-source software.

16.1 License

16.2 Software architecture

Figure 16.1 illustrates the software's high-level application architecture. The processing pipeline transforms textual content through multiple stages to produce the final document. Each processor in the chain receives a document from its predecessor. This allows each processor to handle a single responsibility, such as curling quotation marks, replacing variable names, or inserting a caret position.

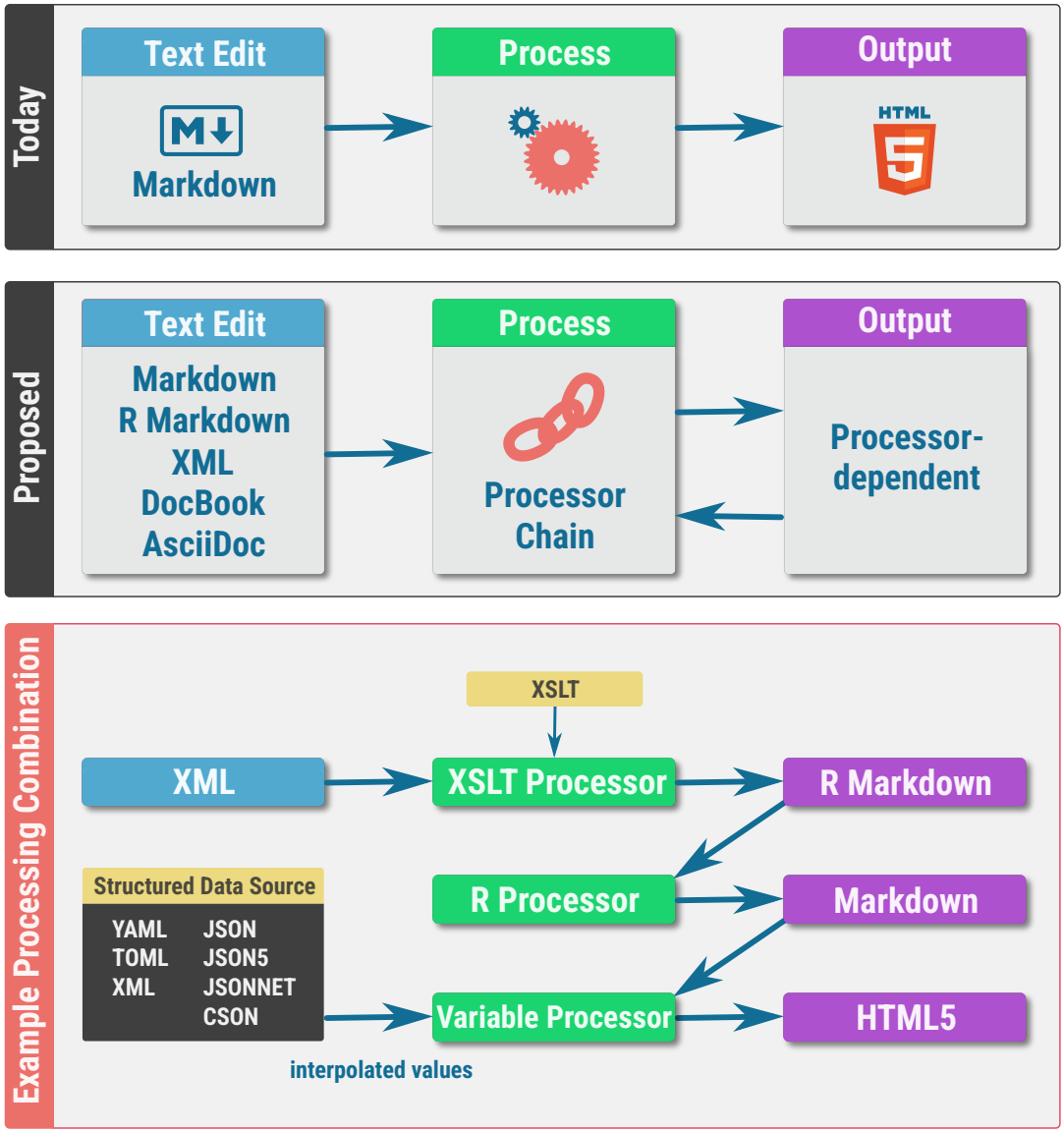

Figure 16.1 Software architecture

- The **"Today"** section shows a single flow where a user writes in Markdown, which is then processed and converted into a final HTML output. This is typical of most conversion programs written today: they can only handle a single type of input and a single type of output.
- The **"Proposed"** section shows that the architecture can be updated to handle a wider range of input formats, all feeding into a more flexible *Processor*

Chain. The two-way arrow indicates that the output from one processing pass can be be fed back into the next process.

- The **“Example Processing Combination”** section provides a detailed look at how the system could handle multiple, complex inputs and outputs simultaneously. It shows how different data formats (XML, YAML, JSON, etc.) are processed through dedicated processors (XSLT Processor, R Processor, etc.) to produce a variety of outputs (R Markdown, Markdown, HTML5), demonstrating a more robust and interconnected system.

A Screenshots

This section provides application screenshots.

A.1 PDF themes

The background of **Figure A.1** depicts a novel being edited. Note the following:

- PDF icon in the upper-left
- Novel metadata as integrated variables towards the top-left
- Theme selection dialog in the upper-middle
- Multiple styles (*Boschet*, *Handrit*, and *Tarmes*)
- Variations in page numbers
- Preferences dialog in the middle

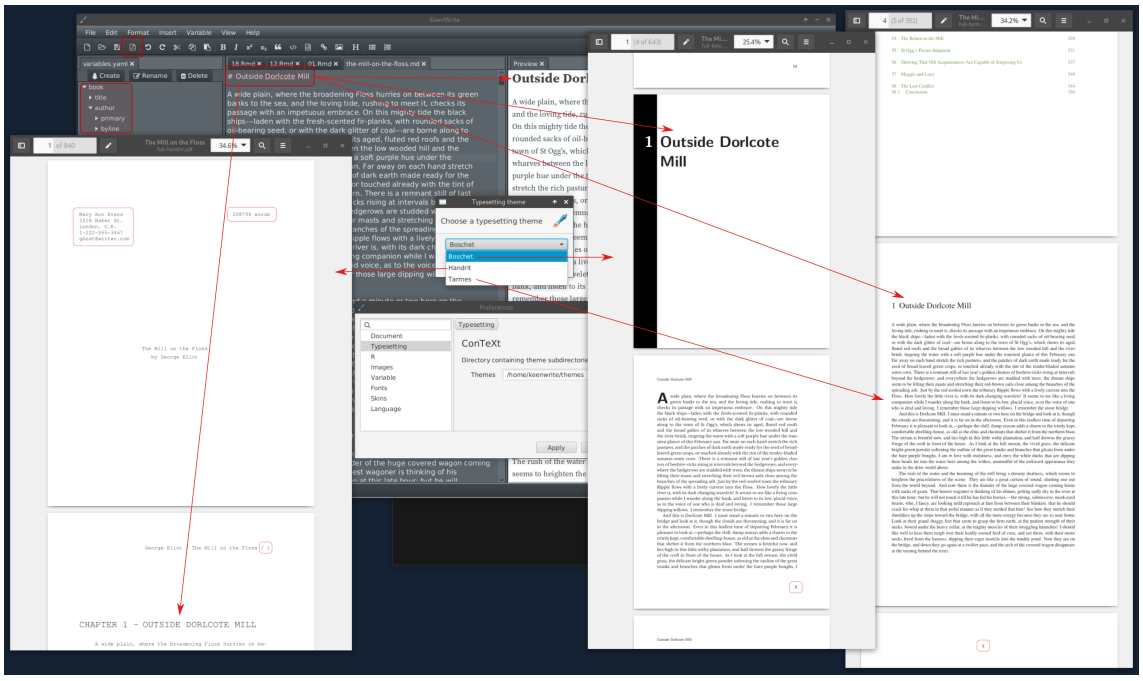

Figure A.1 PDF Themes

A.2 Dockable tabs

Figure A.2 shows a document outline opened and docked in the bottom-left corner:

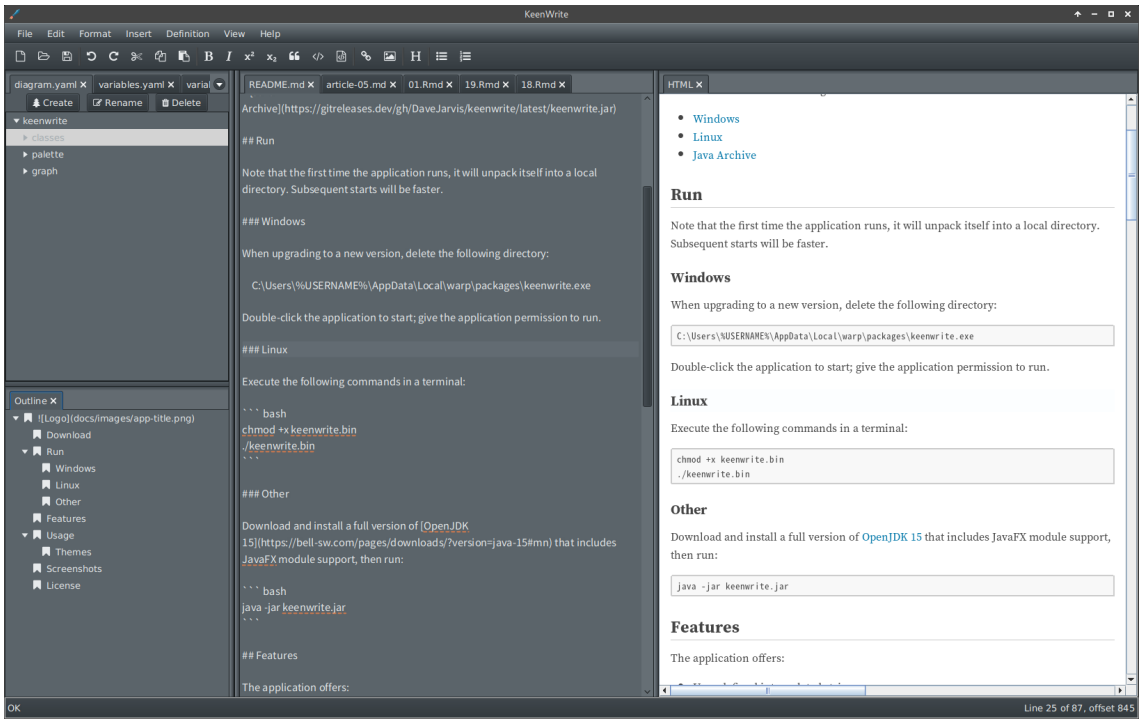

Figure A.2 Document outline

A.3 Variables

A diagram that includes variables is shown in **Figure A.3**.

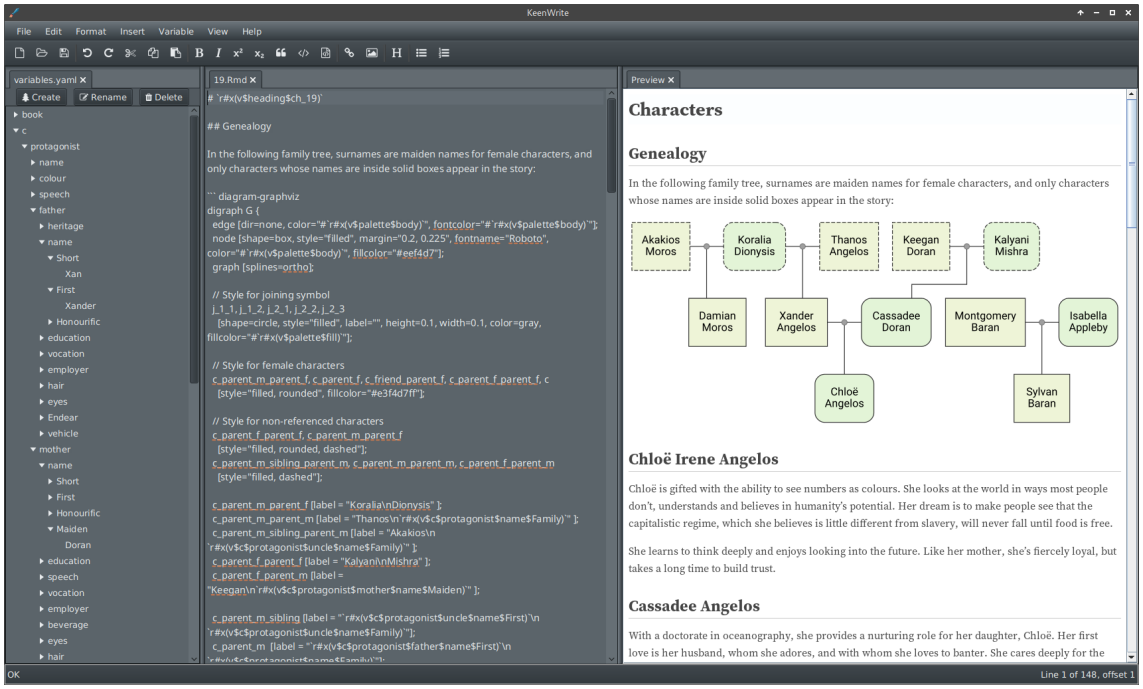

Figure A.3 Family tree diagram

A.4 Equations

Figure A.4 shows TEX equations in a detached preview tab:

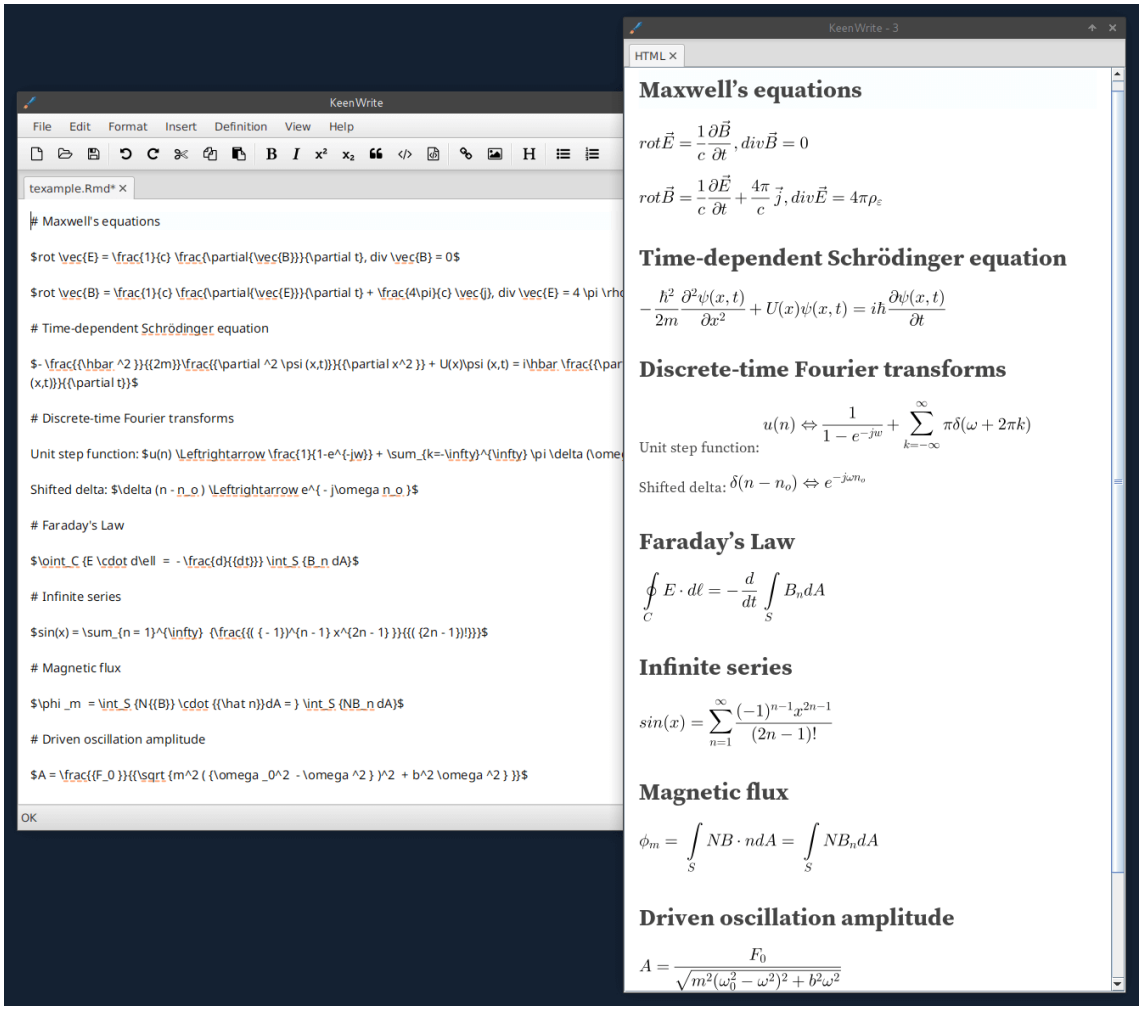

Figure A.4
TeX equations

A.5 Internationalization

Figure A.5 shows a poem with locale settings:

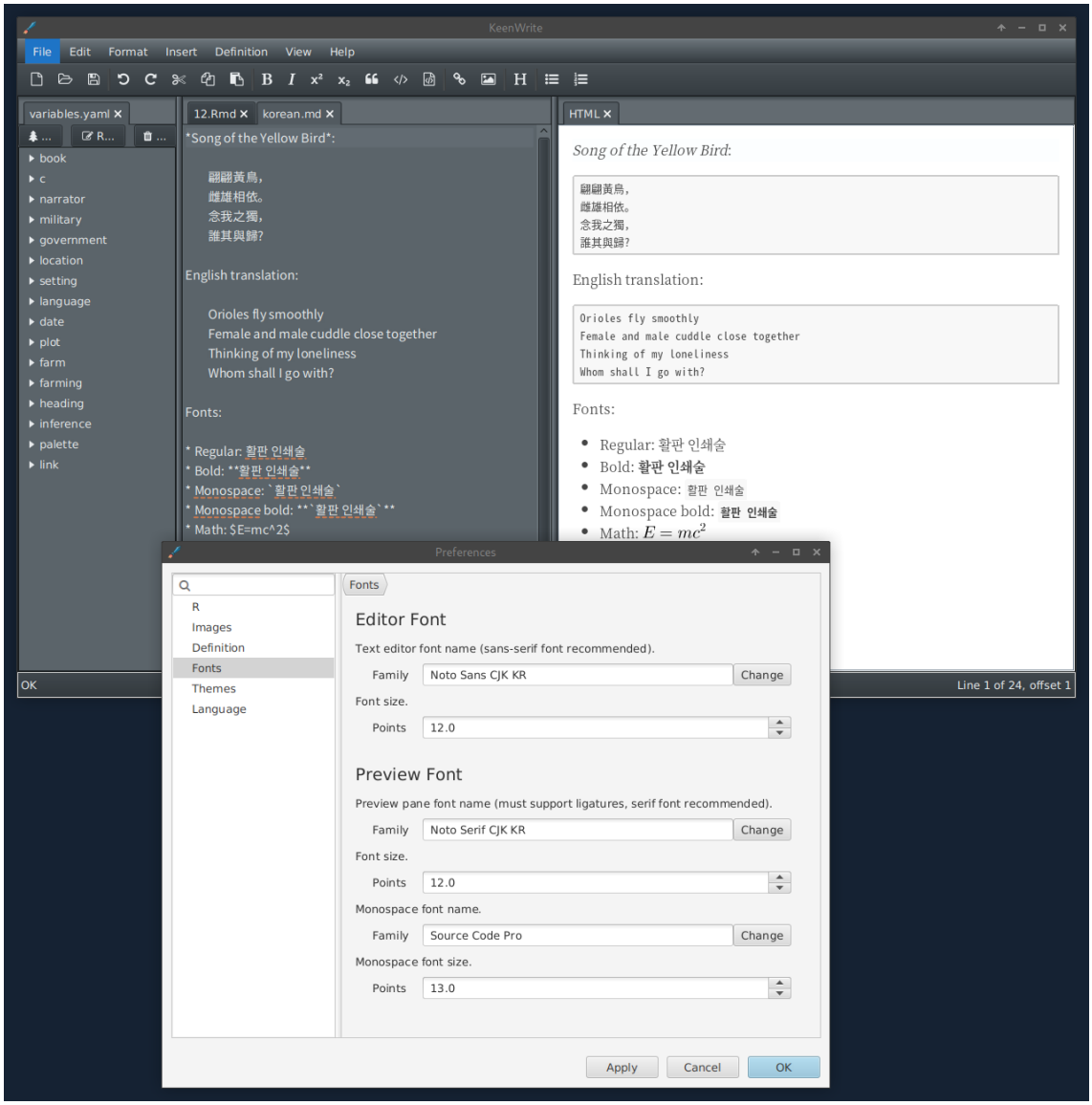

Figure A.5 Locale settings

B Typesetting resources

Documents that introduce the typesetting system:

- **What is ConTEXt?**
- **A not so short introduction to ConTEXt**
- **Dealing with XML in ConTEXt MKIV**
- **Typographic Programming**

The **documentation library** includes the following gems:

- **ConTEXt Manual**
- **ConTEXt command reference**
- **METAFUN Manual**
- **It's in the Details**
- **Fonts out of ConTEXt**

Expert-level documentation includes the **LuaTEX Reference Manual**.

C Acknowledgments

KeenWrite sits on the shoulders of a great many developers, including:

- Dr. Donald E. Knuth, **TEX**
- Hans Hagen, **ConTEXt**
- James Gosling, **Java**
- Karl Tauber, **Markdown Writer FX**

C.1 Third-party libraries

KeenWrite ships with the libraries listed in the following subsections.

C.1.1 flexmark-java

- **Version:** 0.64.8
- **Maven**
- **License(s):** **BSD-2-Clause**
- **Developers:** Vladimir Schneider

C.1.2 ControlsFX

- **Version:** 11.2.2
- **Homepage** | **Source** | **Maven**
- **License(s):** **BSD-3-Clause**
- **Developers:** Jonathan Giles

C.1.3 RichTextFX

- **Version:** 0.11.6
- **Homepage** | **Source** | **Maven**

- **License(s):** **BSD-2-Clause**, **GPL-2.0-only**
- **Developers:** Jordan Martinez, Jurgen Doll, Tomas Mikula

C.1.4 WellBehavedFX

- **Version:** 0.3.3
- **Homepage** | **Source** | **Maven**
- **License(s):** **BSD-2-Clause**
- **Developers:** Tomas Mikula

C.1.5 JavaFX Swing

- **Version:** 24.0.2
- **Homepage** | **Maven**
- **License(s):** **GPL-2.0-with-classpath-exception**
- **Developers:** Abhinay Agarwal, Eugene Ryzhikov

C.1.6 JavaFX Media

- **Version:** 24.0.2
- **Homepage** | **Maven**
- **License(s):** **GPL-2.0-with-classpath-exception**

C.1.7 PreferencesFX

- **Version:** 11.17.0
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Dirk Lemmermann, Dieter Holz, François Martin, Marco Sanfratello

C.1.8 FontAwesomeFX

- **Version:** 8.9
- **Source**
- **License(s):** **Apache-2.0**
- **Developers:** Jens Deters

C.1.9 TiwulFX Dock

- **Version:** 0.5
- **Homepage** | **Source** | **Maven**
- **License(s):** **MIT**
- **Developers:** Amrullah Syadzili

C.1.10 SnakeYAML

- **Version:** 2.5
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Alexander Maslov, Andrey Somov

C.1.11 Jackson Core

- **Version:** 2.19.2
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Tatu Saloranta

C.1.12 Jackson Databind

- **Version:** 2.19.2
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Tatu Saloranta

C.1.13 Jackson Dataformat-YAML

- **Version:** 2.19.2
- **Homepage** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Tatu Saloranta

C.1.14 jsoup

- **Version:** 1.21.2
- **Homepage** | **Source** | **Maven**
- **License(s):** **MIT**
- **Developers:** Jonathan Hedley

C.1.15 Flying Saucer Core

- **Version:** 10.0.0
- **Maven**
- **License(s):** **LGPL-2.1-or-later**
- **Developers:** Patrick Wright, Peter Brant

C.1.16 Apache Commons Compress

- **Version:** 1.28.0
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Christian Grobmeier, Damjan Jovanovic, Emmanuel Bourg, Gary Gregory, Julius Davies, Peter Alfred Lee, Rob Tompkins, Sebastian Bazley, Stefan Bodewig, Torsten Curdt

C.1.17 Apache Commons VFS

- **Version:** 2.10.0
- **Homepage** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Adam Murdoch, Bernd Eckenfels, Gary Gregory, James Carman, James Strachan, Joerg Schaible, Mario Ivankovits, Rahul Akolkar, Ralph Goers

C.1.18 Plexus Common Utilities

- **Version:** 4.0.2
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Andreas Gudian, Andrew Williams, Ben Walding, Brett Porter, Carlos Sanchez, Dan Diephouse, Emmanuel Venisse, Gabriel Belingueres, Guillaume Nodet, Hervé Boutemy, James Taylor, Jason van Zyl, Joakim Erdfelt, John Casey, Karl Heinz Marbaise, Kasper Nielsen, Kenney Westerhof, Konrad Windszus, Kristian Rosenvold, Mark Wilkinson, Michael Osipov, Michal Maczka, Oleg Gusakov, Olivier Lamy, Pete Kazmier, Rahul Thakur, Slawomir Jaranowski, Sylwester Lachiewicz, Trygve Laugstøl, Vincent Siveton

C.1.19 Renjin

- **Version:** 3.5-beta76
- **Maven**
- **License(s):** **GPL-2.0-or-later**
- **Developers:** Alex Bertram

C.1.20 RJson

- **Version:** 0.2.15-renjin-21
- **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Alex Couture-Beil

C.1.21 EchoSVG

- **Version:** 2.2
- **Homepage** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Carlos Amengual

C.1.22 Aho-Corasick

- **Version:** 0.6.3
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Daniel Beck, Dave Jarvis, Robert Bor

C.1.23 JUniversalChardet

- **Version:** 2.5.0
- **Homepage** | **Source** | **Maven**
- **License(s):** **MPL-1.1**, **GPL-3.0-only**, **LGPL-3.0-only**
- **Developers:** Alberto Fernández, Shy Shalom, Kohei Taketa

C.1.24 Jakarta Validation API

- **Version:** 3.1.1
- **Homepage** | **Source** | **Maven**
- **License(s):** **Apache-2.0**
- **Developers:** Emmanuel Bernard, Guillaume Smet, Gunnar Morling, Hardy Ferentschik

C.1.25 EventBus

- **Version:** 3.3.1
- **Homepage** | **Source** | **Maven**

- **License(s):** Apache-2.0
- **Developers:** Markus Junginger

C.1.26 picocli

- **Version:** 4.7.7
- Homepage | Source | Maven
- **License(s):** Apache-2.0
- **Developers:** Remko Popma

C.1.27 JSpecify annotations

- **Version:** 1.0.0
- Homepage | Source | Maven
- **License(s):** Apache-2.0
- **Developers:** Kevin Bourrillion

C.1.28 Java Image Scaling

- **Version:** 0.8.7
- Source
- **License(s):** BSD-3-Clause
- **Developers:** Morten Nobel-Jørgensen

C.1.29 File Preferences

- Homepage
- **License(s):** CC0-1.0
- **Developers:** David Croft

C.1.30 SLF4J API Module

- **Version:** 2.1.0-alpha1
- **Homepage** | **Maven**
- **License(s):** **MIT**
- **Developers:** Ceki Gülcü

C.1.31 SLF4J NOP Provider

- **Version:** 2.0.17
- **Homepage** | **Maven**
- **License(s):** **MIT**
- **Developers:** Ceki Gülcü