novelWriter Documentation

Release 1.0.4

Veronica Berglyd Olsen

Saturday, 06 February 2021 at 01:28

Introduction

Last Updated: Saturday, 06 February 2021 at 01:28

novelWriter is a markdown-like text editor designed for writing novels and larger projects of many smaller plain text documents. It uses its own flavour of markdown that supports a meta data syntax for comments, synopsis and cross-referencing between documents. The idea is to have a simple text editor which allows for easy organisation of text documents and notes, built on a plain text file project repository for robustness.

The plain text storage is suitable for version control software, and also well suited for file synchronisation tools. The core project structure is stored in a single project XML file. Other meta data is primarily saved as JSON files.

Any operating system that can run Python 3 and has the Qt 5 libraries should be able to run novelWriter. It runs fine on Linux, Windows and macOS, and users have tested it on other platforms too. novelWriter can be run directly from the Python source, or installed from the pip tool or via the provided setup script. A setup.exe is provided for Windows. See *Getting Started* for further details.

Useful Links

- Website: https://novelwriter.io
- Documentation: https://novelwriter.readthedocs.io
- Source Code: https://github.com/vkbo/novelWriter
- Source Releases: https://github.com/vkbo/novelWriter/releases
- Issue Tracker: https://github.com/vkbo/novelWriter/issues
- Feature Discussions: https://github.com/vkbo/novelWriter/discussions
- PyPi Project: https://pypi.org/project/novelWriter

CHAPTER 1

Introduction

novelWriter is a simple, multi-document plain text editor using a modified markdown syntax to apply simple formatting to the text. It is designed for writing novels, and allows for the component documents to be ordered freely to create the desired structure of the novel. More details about how projects are structured is covered in *Novel Structure*.

In addition, the project can contain notes on the various plot elements, characters, locations, etc, that make up the story. These notes are organised in a set of category-specific top-level folders (root folders), and each entry can be tagged and cross-referenced from within the novel documents and notes. These tags make it possible to inter-link documents, and generate an overview of the entire novel project and how the various documents and plot elements are interconnected. This is covered in *Novel Projects* and *Project Notes*.

These additional features are not standard in markdown, but are available through special meta keywords described in *Tag References*. Syntax highlighting is provided to make it easier to verify that the markdown tags are used correctly.

An overview of the supported markdown syntax is covered in *User Interface*.

1.1 Design Philosophy

The user interface of novelWriter is intended to be as minimalistic as practically possible, while at the same time provide a complete set of features needed for writing a novel.

Note: novelWriter is not intended to be a full office type word processor. It doesn't support images, links, tables, and other complex structures and objects often needed for such documents. Formatting is limited to headers, and bold, italicised and strikethrough text.

The main window does not have a toolbar like many other applications do. This reduces clutter, and since the documents are formatted with markdown tags, is more or less redundant. However, all formatting features supported are available through convenient keyboard shortcuts. They are also available in the main menu. A full list of shortcuts can be found in the *Keyboard Shortcuts* section.

In addition, novelWriter offers a *Focus Mode* where all the user interface elements other than the document editor itself are hidden away.

The colour scheme of the user interface defaults to that of the host operating system. In addition, a dark theme is provided, and can be enabled in *Preferences* from the *Tools* menu. A number of syntax highlighting themes are also available in *Preferences*. A set of icon themes in colour and greyscale are also offered. The icons are based on the Typicons icon set designed by Stephen Hutchings.

The main window is split in two, or optionally three, panels. The left-most panel contains the project tree and all the documents in your project. The second panel is the document editor, and the optional third panel is a document viewer which can view any document in your project independently of the document editor.

A second tab is also available on the main window. This is the *Outline* tab where the entire novel structure can be displayed, with all the tags and references listed. Depending on how you structure your novel documents, this outline can be quite different from your project tree. Your project tree lists individual documents, your Outline tree lists the structure of the novel itself as it appears in the text of the documents.

1.2 Project Layout

You are free to organise your project documents as you wish into subfolders, and split the text between documents in whatever way suits you. All that matters to novelWriter is the linear order the documents appear at in the project tree (top to bottom). The chapters, scenes and sections of the novel are determined by the headings within those documents.

The four heading levels (**H1** to **H4**) are treated as follows:

- **H1** is used for the book title, and for partitions.
- **H2** is used for chapter tiles.
- **H3** is used for scene titles – optionally replaced by separators.
- **H4** is for section titles within scenes, if such granularity is needed.

This header level structure is only taken into account for novel documents. For the project notes, the header levels have no structural meaning, and the user is free to do whatever they want. See *Novel Structure* and *Project Notes* for more details.

1.3 Project Export

The project can at any time be exported to a range of different formats through the *Build Novel Project* tool. Natively, novelWriter supports export to plain text file, HTML document, novelWriter flavoured markdown, standard markdown (requires Qt 5.14), and to a basic Open Document format.

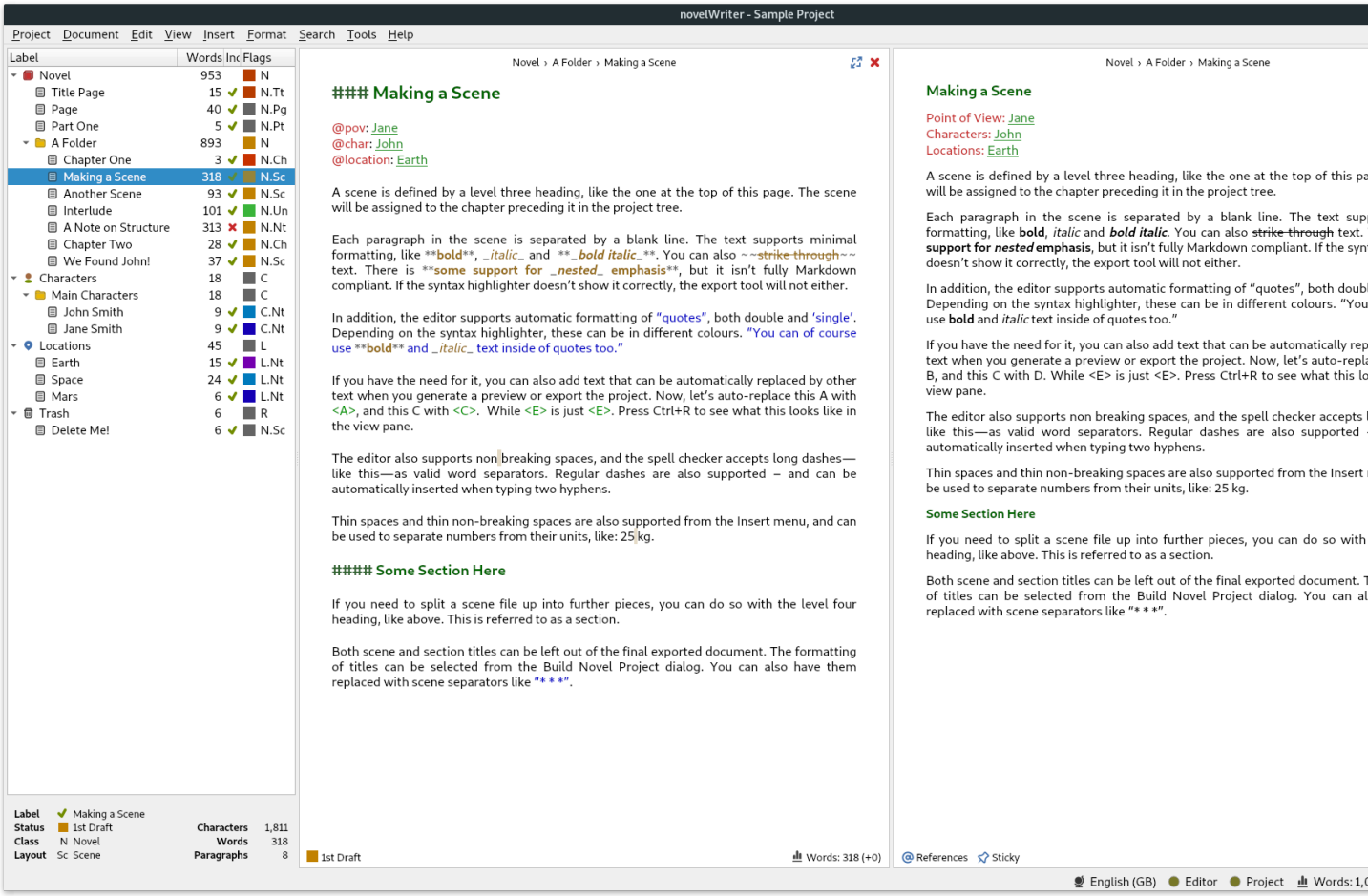
In addition, printing and printing to PDF is also possible. The best supported export format is HTML, which can be imported or converted by a number of other tools like Pandoc, or simply imported into Libre Office Writer and similar word processors.

It is also possible to export the content of the project to a JSON file. This is useful if you want to write your own processing script in for instance Python as the entire novel can be read into a Python dictionary with a couple of lines of code.

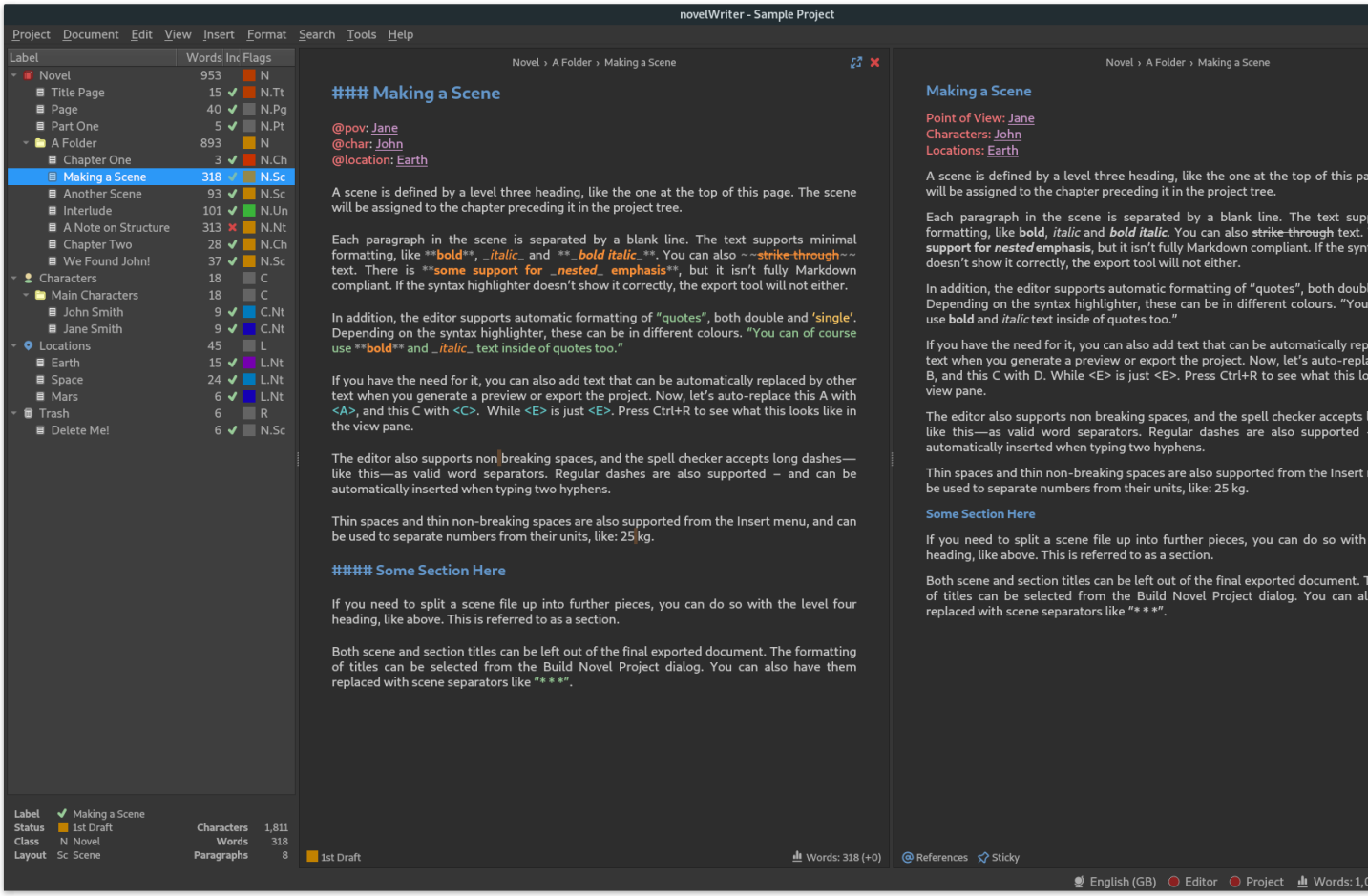
A number of filter options can be applied to the produced document, allowing you to export a draft manuscript, a reference document of notes, an outline based on chapter and scene titles with a synopsis each, and so on. See *Exporting Projects* for more details on export features and formats.

1.4 Screenshots

novelWriter with default system theme:



novelWriter with dark theme:


novelWriter - Sample Project
Project Document Edit View Insert Format Search Tools Help
Label Words Inc Flags
Novel 953 N
Title Page 15 N.Tt
Page 40 N.Pg
Part One 5 N.Pt
A Folder 893 N
Chapter One 3 N.Ch
Making a Scene 318 N.Sc
Another Scene 93 N.Sc
Interlude 101 N.Un
A Note on Structure 313 N.Nt
Chapter Two 28 N.Ch
We Found John! 37 N.Sc
Characters 18 C
Main Characters 18 C
John Smith 9 C.Nt
Jane Smith 9 C.Nt
Locations 45 L
Earth 15 L.Nt
Space 24 L.Nt
Mars 6 L.Nt
Trash 6 R
Delete Me! 6 N.Sc
Label Making a Scene
Status 1st Draft
Class N Novel
Layout Sc Scene
Characters 1,811
Words 318
Paragraphs 8
Novel › A Folder › Making a Scene
Making a Scene
@pov: Jane
@char: John
@location: Earth
A scene is defined by a level three heading, like the one at the top of this page. The scene will be assigned to the chapter preceding it in the project tree.
Each paragraph in the scene is separated by a blank line. The text supports minimal formatting, like **bold**, _italic_ and **_bold italic_**. You can also ~~strike through~~ text. There is **some support for _nested_ emphasis**, but it isn't fully Markdown compliant. If the syntax highlighter doesn't show it correctly, the export tool will not either.
In addition, the editor supports automatic formatting of "quotes", both double and 'single'. Depending on the syntax highlighter, these can be in different colours. "You can of course use **bold** and _italic_ text inside of quotes too."
If you have the need for it, you can also add text that can be automatically replaced by other text when you generate a preview or export the project. Now, let's auto-replace this A with <A>, and this C with <C>. While <E> is just <E>. Press Ctrl+R to see what this looks like in the view pane.
The editor also supports non breaking spaces, and the spell checker accepts long dashes—like this—as valid word separators. Regular dashes are also supported – and can be automatically inserted when typing two hyphens.
Thin spaces and thin non-breaking spaces are also supported from the Insert menu, and can be used to separate numbers from their units, like: 25 kg.
Some Section Here
If you need to split a scene file up into further pieces, you can do so with the level four heading, like above. This is referred to as a section.
Both scene and section titles can be left out of the final exported document. The formatting of titles can be selected from the Build Novel Project dialog. You can also have them replaced with scene separators like "* * *".
1st Draft
Words: 318 (+0)
Novel › A Folder › Making a Scene
Making a Scene
Point of View: Jane
Characters: John
Locations: Earth
A scene is defined by a level three heading, like the one at the top of this pa
will be assigned to the chapter preceding it in the project tree.
Each paragraph in the scene is separated by a blank line. The text sup
formatting, like bold, italic and bold italic. You can also strike through text.
support for nested emphasis, but it isn't fully Markdown compliant. If the syn
doesn't show it correctly, the export tool will not either.
In addition, the editor supports automatic formatting of "quotes", both doub
Depending on the syntax highlighter, these can be in different colours. "You
use bold and italic text inside of quotes too."
If you have the need for it, you can also add text that can be automatically rep
text when you generate a preview or export the project. Now, let's auto-repl
B, and this C with D. While <E> is just <E>. Press Ctrl+R to see what this lo
view pane.
The editor also supports non breaking spaces, and the spell checker accepts
like this—as valid word separators. Regular dashes are also supported
automatically inserted when typing two hyphens.
Thin spaces and thin non-breaking spaces are also supported from the Insert
be used to separate numbers from their units, like: 25 kg.
Some Section Here
If you need to split a scene file up into further pieces, you can do so with
heading, like above. This is referred to as a section.
Both scene and section titles can be left out of the final exported document.
of titles can be selected from the Build Novel Project dialog. You can al
replaced with scene separators like "* * *".
References
Sticky
English (GB)
Editor
Project
Words: 1,

CHAPTER 2

Getting Started

This section contains brief guides to how you can get novelWriter running on your computer. These are the methods currently supported by the developer. Packages may also be available in other package managers, but those are not managed by the developer. A Windows installer file is also provided on the GitHub releases page and linked from the main website.

As novelWriter matures, more options for how to install it and get it running will be added. For non-Windows users the install process is at the present time best suited for people used to working with the command line. But even if you're not, the install process is fairly straightforward.

The next pages have specific install instructions for the various operating systems novelWriter can run on. The instructions below are supplementary information, instructions for alterbative methods, and additional build options.

Note: The text below assumes the command `python` corresponds to a Python 3 executable. For operating systems with both Python 2 and 3, the command `python3` may be needed instead. On Linux, the scripts can also be made executable and run without the `python` command. Likewise, `pip` may need to be replaced with `pip3`.

2.1 Dependencies

novelWriter has been designed to rely on as few dependencies as possible. Aside from the packages needed to communicate with the Qt GUI libraries, only one package is required for handling the XML format of the main project file. Everything else is handled with standard Python libraries.

The following Python packages are needed to run novelWriter:

- `pyqt5` – needed for connecting with the Qt5 libraries.
- `lxml` – needed for full XML support.
- `pyenchant` – needed for efficient spell checking (optional).

PyQt/Qt should be at least 5.2.1, but ideally 5.10 or higher for nearly all features to work. Exporting to standard Markdown, for instance, requires PyQt/Qt 5.14. Searching using regular expressions requires 5.3, and for full Unicode

support, 5.13. There is no known minimum version requirement for package `lxml`, but the code was originally written with 4.2, which is therefore set as the minimum. It may work on lower versions. You have to test it.

Optionally, a package can be installed to interface with the Enchant spell checking libaries, but this isn't strictly required. If no external spell checking library is available, novelWriter falls back to using the internal `difflib` of Python to check spelling. This is a much slower approach, and it is less sophisticated than full spell checking libaries, but if you only work with small files, the performance loss is not noticeable. The spell check library must be at least 3.0 to work with Windows. On Linux, 2.0 also works fine.

If you install from PyPi, these dependencies should be installed automatically. If you install from source, dependencies can still be installed from PyPi with:

```
pip install -r requirements.txt
```

2.2 Installing via PyPi

The application is written in Python 3 using Qt5 via PyQt5. It is developed on Linux, but it should in principle work fine on other operating systems as long as dependencies are met.

You can download the latest version of novelWriter from the source repository on GitHub. novelWriter is also hosted on PyPi, and can be installed on all operating systems that support Qt5 and Python 3. It is regularly tested on Linux, Windows and macOS. The latest version of novelWriter is 1.0.4.

To install from PyPi you must first have the `python` and `pip` commands available on your system. If you don't, see specific instructions for your operating system later in this document. To install novelWriter from PyPi, use the following command:

```
pip install novelwriter
```

To upgrade an existing installation, use:

```
pip install --upgrade novelwriter
```

When installing via pip, novelWriter can be launched from command line with:

```
novelWriter
```

Make sure the install location for pip is in your PATH variable. This is not always the case by default.

2.3 Building the Documentation

If you installed novelWriter from a package, the documentation should be included. If you're running novelWriter from the source code, a local copy of this documentation can be generated. It requires the following Python packages on Debian and Ubuntu.

- `python3-sphinx`
- `python3-sphinxcontrib.qthelp`

Or from PyPi:

```
pip install sphinx sphinxcontrib-qthelp
```

The documentation can then be built from the `docs` folder in the source code by running:

```
make html
```

If successful, the documentation should be available in the `docs/build/html` folder and you can open the `index.html` file in your browser.

The documentation can also be built for the Qt Assistant. To build the help packages from the documentation source, run the following from the root source folder:

```
python setup.py qthelp
```

The setup script will copy the generated files into the `nw/assets/help` folder, and novelWriter will detect the presence of the files and redirect the menu help entry to open help locally instead of sending the user to the website. Pressing the `F1` key will in any case try to open help locally first, then send you to the website as a fallback.

Note: In order for the local version of help to work, the Qt Assistant must be installed on the local computer. If it isn't available, or novelWriter cannot find it, the help feature will fall back to redirecting you to the documentation website.

CHAPTER 3

Setup on Linux

This is a brief guide to how you can get novelWriter running on a Linux computer. There are currently no packaged version of novelWriter for Linux, so it is recommended that you just extract the source to a practical location on your system and run the `setup.py` script.

3.1 Running from Source

To run novelWriter from source, download the latest source package from the release page on GitHub or the main website, or if you have git running on your computer, you can also clone the repository.

3.1.1 Step 1: Installing Dependencies

The dependencies of novelWriter are generally available from Linux distro repositories. For Debian and Ubuntu, they can be installed with:

```
sudo apt install python3-pyqt5 python3-lxml python3-enchant
```

If you prefer to install dependencies via PyPi, or the repository dependencies are out of date, you can install them with:

```
pip3 install --user -r requirements.txt
```

3.1.2 Step 2: Install Package (Optional)

You can install novelWriter to the correct location for Python packages with:

```
./setup.py install
```

This is equivalent to what the `pip` installer does. It puts novelWriter in the location on your system where Python packages are usually kept. This is not really the best suited location for a GUI application like novelWriter, so you may instead copy the entire source to a suiteable location yourself.

By default, this command installs novelWriter for the current user only. To install for all users, run the script with the `sudo` command.

This should install novelWriter to either `~/.local/bin/novelWriter` if installed for local user only, or to `/usr/local/bin/novelWriter` if installed for all users.

3.1.3 Step 3: Create Launcher Icons

To set up the novelWriter desktop launcher, the icons, and the project file association, run:

```
./setup.py xdg-install
```

By default, these commands install novelWriter and its icons for the current user only. To install for all users, run the script with the `sudo` command.

Tip: All options of the setup script can be listed with: `./setup.py help`.

CHAPTER 4

Setup on macOS

This is a brief guide to how you can get novelWriter running on macOS. There are currently no packaged version of novelWriter for macOS, so it is recommended that you just extract the source to a practical location on your system and run it.

4.1 Running from Source

To run novelWriter from source, download the latest source package from the release page on GitHub or the main website, or if you have git running on your computer, you can also clone the repository.

4.1.1 Step 1: Installing Dependencies

These instructions assume you're using brew, and have Python and pip set up. If not, see the brew docs for help. Main requirements are installed via the requirements file. You also need to install the `pyobjc` package, so you should run:

```
pip3 install --user -r requirements.txt
pip3 install --user pyobjc
```

For spell checking you may also need to install the enchant package. It comes with a lot of default dictionaries.

```
brew install enchant
```

With the dependencies in place, you can launch the `novelWriter.py` script directly to run novelWriter.

4.1.2 Step 2: Install Package (Optional)

You can install novelWriter to the correct location for Python packages with:

```
./setup.py install
```

This is equivalent to what the `pip` installer does. It puts novelWriter in the location on your system where Python packages are usually kept. This is not really the best suited location for a GUI application like novelWriter, so you may instead copy the entire source to a suiteable location yourself.

After this, you should be able to launch novelWriter by running `novelWriter` in a command line window.

Note: Right now there isn't a better integration with macOS available. Contributions from someone more familiar with macOS would be very much appreciated.

CHAPTER 5

Setup on Windows

This is a brief guide to how you can get novelWriter running on a Windows computer.

Unlike other operating systems, Windows does not come prepared with an environment to run Python applications, so you must first install Python. After that, running novelWriter is straightfoeward.

If you have any problems, you can always open a question on the project's discussions page. This requires a GitHub account.

5.1 Using the Installer

The installer for Windows is no longer provided. You can still create it yourself if you want to. It can be generated with the provided `setup.py` script. use the script's `help` command to get further instructions.

Please use the "Running from Source" option below instead.

5.2 Running from Source

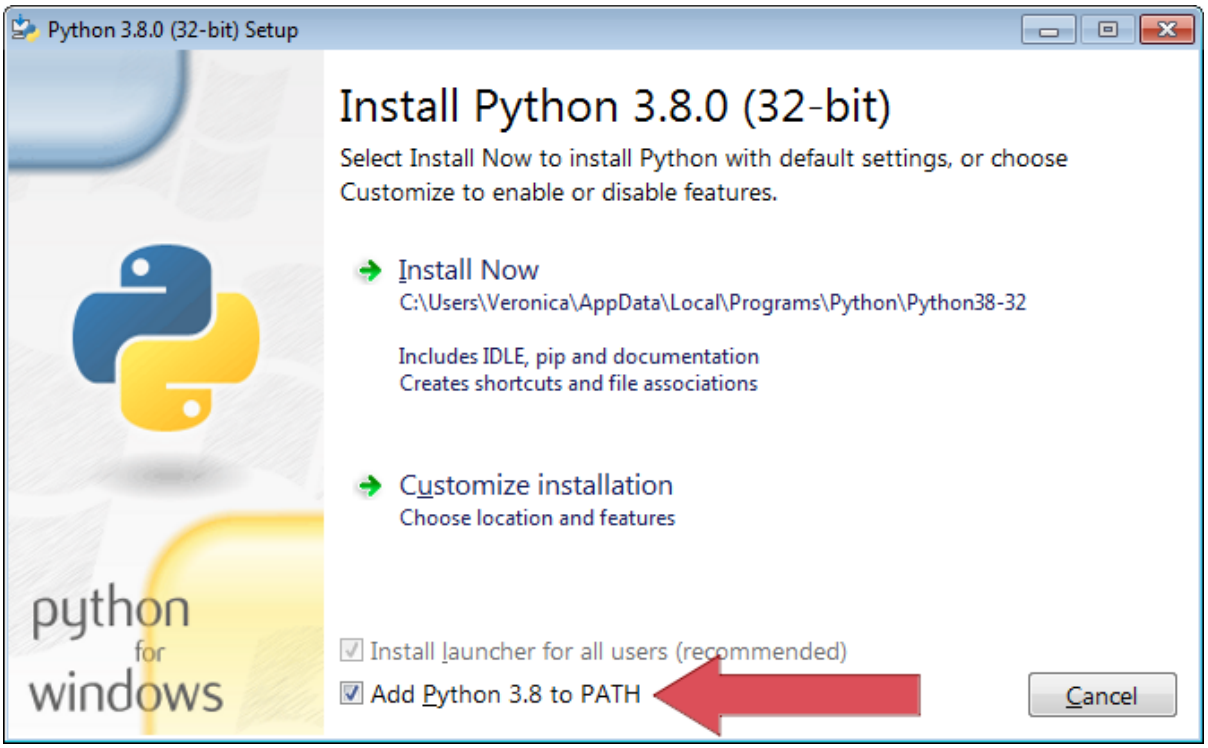
To run novelWriter from source, download the latest source zip file from the release page on GitHub or the main website. The file named `novelWriter-x.y.z-minimal.zip`, where `x.y.z` is the version number, is ideal for this. You can also download the full source.

In order to make novelWriter run on your system, you must first have Python installed (Step 1). Thereafter, a script will do the rest of the job (Step 2). or you can do it yourself manually if you wish to.

5.2.1 Step 1: Installing Python

If you already have Python installed, you can skip this step. If you don't have it installed, you can download it from the python.org website. novelWriter should work with Python 3.6 or higher, but it is recommended that you install the latest version of Python.

Make sure you select the "Add Python to PATH" option during installation, otherwise the `python` command will not work in the command line window.



5.2.2 Step 2: Dependencies and Icons

Alternative A: By Script

Open the folder where you extracted the novelWriter source, and double-click the file named `setup_windows` or `setup_windows.bat`. This should open a command line window and run the setup script to install dependencies, and add desktop and start menu icons. It will also check that it can find Python from Step 1.

Note: If you upgrade Python to a newer version and the path to `pythonw.exe` changes, you may need to run this script again.

Alternative B: Manual Installation

The above alternative can also be run manually.

Open the windows command prompt. It can be launched by pressing the `Win` key and typing "cmd". The "Command Prompt" app should then be in the list of applications.

With the command prompt open, navigate to the folder where you extracted the novelWriter source, and run the following commands:

```
python setup.py pip
python setup.py win-install
```

The first command will install the dependencies on your system from the Python Package Index, and the second command will create a desktop icon and a start menu icon. That should be all that you need.

CHAPTER 6

User Interface

The user interface is kept as simple as possible to avoid distractions when writing. This page lists all the main GUI elements, and explains what they do.

6.1 The Project Tree

The main window contains a project tree in the left-most panel. It shows the entire structure of the project. It has four columns:

Label The first column shows the item icon and its label. The labels can be edited from the *Project* menu, or by pressing `F2` or `CtrlE`. The label is not the same as the title you set inside the document, but it will appear in the header above the document text itself.

Words The second column shows the word count of the document, or the sum of words of the child items if it is a folder. If the counts seem incorrect, they can be updated by rebuilding the project index from the *Tools* menu, or by pressing `F9`.

Inc The third column indicates whether the document is included in the final project build or not. You may want to filter out documents that you no longer want to keep in the final manuscript, but want to keep in the project for reference.

Flags The fourth column shows various meta data flags for the item. The first is an icon indicating the importance or status of the document. These are colour coded status levels that you control and define yourself. They can be changed in *Project Settings* from the *Project* menu. The first character after the icon indicates the class of the item, that is `N` for **Novel**, `C` for **Character**, etc (see *Tag References*). The characters after the dot indicate the document layout type (see *Project Roots*).

Right-clicking an item in the project tree will open a context menu under the cursor, displaying a selection of actions that can be performed on the selected item.

Below the project tree you will find a small details panel showing the full information of the currently selected item. This panel also includes the latest paragraph and character counts in addition to the word count.

6.1.1 Project Tree Drag and Drop

The project tree allows drag and drop to a certain extent. This feature is primarily intended for rearranging the order of your documents within each root folder, and has only limited support for moving documents elsewhere in the project tree. In general, bulk actions are not allowed. This is deliberate to avoid accidentally messing up your project. The project tree has no undo function.

Documents and their folders can be rearranged freely within their root folders. Novel documents cannot be moved out of the *Novel* folder, except to *Trash* and the *Outtakes* folders. Notes can be moved freely between all root folders.

Folders cannot be moved at all outside their root tree. Neither can a folder containing documents be deleted. You must first delete the containing documents.

Root folders in the project tree cannot be dragged and dropped at all. If you want to reorder them, you can move them up or down with respect to eachother from the *Tools* menu, the right-click context menu, or by pressing `CtrlShift` and the `Up` or `Down` key.

6.2 Editing and Viewing Documents

To edit a document, double-click it in the project tree, or press the `Return` key while having it selected. This will open the document in the document editor. The editor uses a simplified markdown format. The format is described in the *Markdown Format* section below. The editor has a maximise button (toggles the *Distraction Free Mode*) and a close button in the top-right corner.

Any document in the project tree can also be viewed in parallel in a right hand side document viewer. To view a document, press `CtrlR`, or select *View Document* in the menu. If you have a middle mouse button, middle-clicking on the document will also open it in the viewer. The document viewed does not have to be the same document currently being edited. However, If you *are* viewing the same document, pressing `CtrlR` again will update the document with your latest changes. You can also press the reload button in the top-right corner of the view panel next to the close button to achieve the same thing.

Both the document editor and viewer will show the label of the document in the header at the top of the edit or view panel. Optionally, the full project path to the document can be shown. This can be set in the *Preferences* dialog from the *Tools* menu. Clicking on the document title bar will select and reveal its location in the project tree, making it easier to locate in a large project.

Any tag reference in the editor can be opened in the viewer by moving the cursor to the label and pressing `CtrlReturn`. You can also control-click them with your mouse. In the viewer, the references become clickable links. Clicking them will replace the content of the viewer with the content of the document the reference points to. The document viewer keeps a history of viewed documents, which you can navigate with the arrow buttons in the top-left corner of the viewer. If your mouse has back and forward navigation buttons, these can be used as well.

At the bottom of the view panel there is a *References* panel. (If it is hidden, click the icon to reveal it.) This panel will show links to all documents referring back to it, if any has been defined. The *Sticky* button will freeze the content of the panel to the current document, even if you navigate to another document. This is convenient if you want to quickly look through all documents in the list in the *References* panel without losing the list in the process.

Note: The *References* panel relies on an up-to-date index of the project. The index is maintained automatically. However, if anything is missing, or seems wrong, the index can always be rebuilt by selecting *Rebuild Index* from the *Tools* menu, or by pressing `F9`.

6.3 Auto-Replace as You Type

A few auto-replace features are supported by the editor. You can control every aspect of the auto-replace feature from *Preferences*. You can also disable it entirely.

Tip: If you don't like auto-replacement, all symbols inserted by this feature are also available in the *Insert* menu, and via convenient *Insert Shortcuts*.

The editor is able to replace two and three hyphens with short and long dashes, triple points with ellipsis, and replace straight single and double quotes with user-defined quote symbols. It will also try to determine whether to use the opening or closing symbol, although this feature isn't always accurate. Especially distinguishing between closing single quote and apostrophe can be tricky for languages that use the same symbol for these.

Tip: If the auto-replace feature changes a symbol when you did not want it to change, pressing `CtrlZ` immediately after the auto-replacement will undo it without undoing the character you typed.

6.4 Markdown Format

The document editor uses a simplified markdown format. That is, it supports basic formatting like emphasis (italic), strong importance (bold) and strikethrough text, as well as four levels of headings.

Some non-standard markdown features have been added. For instance, novelWriter allows for comments, a synopsis tag, and a set of keyword and value sets used for tags and references.

6.4.1 Headings

Four levels of headings are allowed. For documents of layout `Note`, they are free to be used as you see fit, but for all other layouts used for the novel text itself, they indicate the structural level of the novel. See *Importance of Headings* for more details.

`# Title` Heading level one. If the document is a novel file, the header level indicates the start of a new partition. This heading level can also be used for the title page's novel title.

`## Title` Heading level two. If the document is a novel file, the header level indicates the start of a new chapter. Chapter numbers can be inserted automatically when exporting the manuscript.

`### Title` Heading level three. If the document is a novel file, the header level indicates the start of a new scene. Scene numbers or scene separators can be inserted automatically when exporting the manuscript, so you can use the title field as a working title for your scenes.

`#### Title` Heading level four. If the document is a novel file, the header level indicates the start of a new section. Scene titles can be replaced by separators or removed when exporting the manuscript, so you can use the title field as a working title for your sections.

Note: The space after the `#` characters is mandatory. The syntax highlighter will change colour and font size when the heading is correctly formatted.

6.4.2 Text Emphasis

A minimal set of text emphasis styles are supported.

text The text is rendered as emphasised text (italicised).

****text**** The text is rendered as strongly important text (bold).

~~text~~ Strikethrough text.

In markdown guides it is often recommended to differentiate between strong importance and emphasis by using `**` for strong and _ for emphasis, although markdown generally also supports __ for strong and `*` for emphasis. However, since the differentiation makes the highlighting and conversion significantly simpler and faster, in novelWriter this is a rule, not just a recommendation.

In addition, the following rules apply:

1. The emphasis and strikethrough formatting tags do not allow spaces between the words and the tag itself. That is, `**text**` is valid, `**text **` is not.
2. More generally, the delimiters must be on the outer edge of words. That is, `some **text in bold** here` is valid, `some** text in bold** here` is not.
3. If using both `**` and _ to wrap the same text, the underscore must be the inner wrapper. This is due to the underscore also being a valid word character, so if they are on the outside, they violate rule 2.
4. Text emphasis does not span past line breaks. If you need to add emphasis to multiple lines or paragraphs, you must apply it to each of them in turn.

6.4.3 Comments and Synopsis

In addition to these standard markdown features, novelWriter also allows for comments in documents. The text of a comment is ignored by the word counter. The text can also be filtered out when exporting or viewing the document.

If the first word of a comment is `Synopsis:` (with the colon), the comment is treated specially and will show up in the *Project Outline View* in a dedicated column. The word `synopsis` is not case sensitive. If it is correctly formatted, the syntax highlighter will indicate this by altering the colour of the word.

% text... This is a comment. The text is not exported by default (this can be overridden), seen in the Viewer, or counted towards word counts.

% Synopsis: text... This is a synopsis comment. It is generally treated in the same way as a regular comment, except that it is also captured by the indexing algorithm and displayed in the *Project Outline View*. It can also be filtered separately when exporting the project to for instance generate an outline document of the whole project.

Note: Only one comment can be flagged as a synopsis comment for each heading. If multiple comments are flagged as synopsis comments, the last one will be used and the rest ignored.

6.4.4 Tags and References

The document editor supports a minimal set of keywords used for setting tags, and making references between documents. The tags and references can be set once per section defined by a heading. Using them multiple times under the same heading will just override the previous setting.

@keyword: value A keyword argument followed by a value, or a comma separated list of values.

The available tag and reference keywords are listed in the *Tag References* section. They can also be inserted at the cursor position in the editor via the *Insert* menu.

6.4.5 Additional Markdown and Non-Standard Features

The editor and viewer also support markdown standard hard line breaks, and preserves non-breaking spaces if running with Qt 5.9 or higher. For older versions, the non-breaking spaces are lost when the document is saved. This is unfortunately hard-coded in the Qt text editor.

- A hard line break can be achieved by leaving two or more spaces at the end of the line. This is standard markdown syntax. Alternatively, the user can press `CtrlK`, `Return` to insert this type of space.
- A non-breaking space can be inserted with `CtrlK`, `Space`.
- Thin spaces are also supported, and can be inserted with `CtrlK`, `ShiftSpace`.
- Non-breaking thin space can be inserted with `CtrlK`, `CtrlSpace`.

These are all insert features, and the *Insert* menu has more. They are also listed in *Insert Shortcuts*.

Both hard line breaks and non-breaking spaces are highlighted by the syntax highlighter as an alternate coloured background, depending on the selected theme.

6.5 Project Outline View

The project's Outline view is available as the second tab on the right hand side of the main window labelled *Outline*. The outline provides an overview of the novel structure, displaying a tree hierarchy of the elements of the novel, that is, the level 1 to 4 headings representing partitions, chapters, scenes and sections.

The document containing the heading can also be displayed as a separate column, as well as the line number where it occurs. Double-clicking an entry will open the corresponding document in the editor.

Note: Since the internal structure of the novel does not depend directly on the folder and document structure of the project tree, these will not necessarily look the same, depending on how you choose to organise your documents. See the *Novel Structure* page for more details.

Various meta data and information extracted from tags can be displayed in columns in the outline. A default set of such columns is visible, but you can turn on or off more columns by right clicking the header and selecting the columns you want to show. The order of the columns can also be rearranged by dragging them to a different position.

Note: The *Title* column cannot be disabled or moved.

The information viewed in the outline is based on the project's main index. While novelWriter does its best to keep the index up to date when content changes, you can always rebuild it manually by pressing `F9` if something isn't right.

The outline view itself can be regenerated by pressing `F10`. You can also enable automatic updating in the *Tools* menu, which will trigger an update whenever the index is updated and the *Outline* tab is active. You may want to disable this feature if your project is very large,

The *Synopsis* column of the outline view takes its information from a specially formatted comment. See *Comments and Synopsis*.

6.6 Keyboard Shortcuts

Most features are available as keyboard shortcuts. These are as follows:

Table 1: Keyboard Shortcuts

Shortcut	Description
`Alt1`	Switch focus to the project tree.
`Alt2`	Switch focus to document editor.
`Alt3`	Switch focus to document viewer.
`AltLeft`	Move backward in the view history of the document viewer.
`AltRight`	Move forward in the view history of the document viewer.
`Ctrl.`	Open menu to correct word under cursor.
`Ctrl,`	Open the *Preferences* dialog.
`Ctrl/`	Change block format to comment.
`Ctrl-`	Strikethrough selected text, or word under cursor.
`Ctrl0`	Remove block formatting for block under cursor.
`Ctrl1`	Change block format to header level 1.
`Ctrl2`	Change block format to header level 2.
`Ctrl3`	Change block format to header level 3.
`Ctrl4`	Change block format to header level 4.
`CtrlA`	Select all text in the document.
`CtrlB`	Format selected text, or word under cursor, with strong emphasis (bold).
`CtrlC`	Copy selected text to clipboard.
`CtrlD`	Wrap selected text, or word under cursor, in double quotes.
`CtrlE`	If in the project tree, edit a document or folder settings. (Same as `F2`.)
`CtrlF`	Open the search bar and search for the selected word, if any is selected.
`CtrlG`	Find next occurrence of search word in current document. (Same as `F3`.)
`CtrlH`	Open the search and replace bar and search for the selected word, if any is selected. (On Mac, this is `Cmd=`.)
`CtrlI`	Format selected text, or word under cursor, with emphasis (italic).
`CtrlK`	Activate the insert commands. The commands are listed in *Insert Shortcuts*.
`CtrlN`	Create new document.
`CtrlO`	Open selected document.
`CtrlQ`	Exit novelWriter.
`CtrlR`	If in the project tree, open a document for viewing. If the editor has focus, open current document for viewing.
`CtrlS`	Save the current document in the document editor.
`CtrlV`	Paste text from clipboard to cursor position.
`CtrlW`	Close the current document in the document editor.
`CtrlX`	Cut selected text to clipboard.
`CtrlY`	Redo latest undo.
`CtrlZ`	Undo latest changes.
`CtrlF7`	Toggle spell checking.
`CtrlF10`	Toggle automatic updating of project outline.
`CtrlDel`	If in the project tree, move a document to trash, or delete a folder.
`CtrlEnter`	Open the tag or reference under the cursor in the Viewer.
`CtrlShift,`	Open the *Project Settings* dialog.
`CtrlShift/`	Remove block formatting for block under cursor.
`CtrlShift1`	Replace occurrence of search word in current document, and search for next occurrence.

Continued on next page

Table 1 – continued from previous page

Shortcut	Description
CtrlShiftA	Select all text in current paragraph.
CtrlShiftD	Wrap selected text, or word under cursor, in single quotes.
CtrlShiftG	Find previous occurrence of search word in current document. (Same as ShiftF3.)
CtrlShiftI	Import text to the current document from a text file.
CtrlShiftN	Create new folder.
CtrlShiftO	Open a project.
CtrlShiftR	Close the document viewer.
CtrlShiftS	Save the current project.
CtrlShiftW	Close the current project.
CtrlShiftZ	Alternative sequence for redo last undo.
CtrlShiftUp	Move item one step up in the project tree.
CtrlShiftDown	Move item one step down in the project tree.
F1	Open the documentation. This will either open the Qt Assistant, if available, or send you to the documentation website.
F2	If in the project tree, edit a document or folder settings. (Same as CtrlE)
F3	Find next occurrence of search word in current document. (Same as CtrlG)
F5	Open the *Build Novel Project* dialog.
F6	Open the *Writing Statistics* dialog.
F7	Re-run spell checker.
F8	Activate *Focus Mode*, hiding the project tree and document viewer.
F9	Re-build the project index.
F10	Re-build the project outline.
F11	Activate full screen mode.
ShiftF1	Open the online documentation in the system default browser.
ShiftF3	Find previous occurrence of search word in current document. (Same as CtrlShiftG.)
Return	If in the project tree, open a document for editing.

Note: On macOS, replace Ctrl with Cmd.

6.6.1 Insert Shortcuts

A set of insert features are also available through shortcuts, but they require a double combination of key sequences. The insert feature is activated with CtrlK, followed by a key or key combination for the inserted content.

Table 2: Keyboard Shortcuts

Shortcut	Description
`CtrlK, -`	Insert a short dash (en dash).
`CtrlK, _`	Insert a long dash (em dash).
`CtrlK, .`	Insert an ellipsis.
`CtrlK, 1`	Insert a left single quote.
`CtrlK, 2`	Insert a right single quote.
`CtrlK, 3`	Insert a left double quote.
`CtrlK, 4`	Insert a right double quote.
`CtrlK, '`	Insert a modifier apostrophe.
`CtrlK, Return`	Insert a hard line break.
`CtrlK, Space`	Insert a non-breaking space.
`CtrlK, ShiftSpace`	Insert a thin space.
`CtrlK, CtrlSpace`	Insert a thin non-breaking space.
`CtrlK, G`	Insert a `@tag` keyword.
`CtrlK, V`	Insert a `@pov` keyword.
`CtrlK, C`	Insert a `@char` keyword.
`CtrlK, P`	Insert a `@plot` keyword.
`CtrlK, T`	Insert a `@time` keyword.
`CtrlK, L`	Insert a `@location` keyword.
`CtrlK, O`	Insert an `@object` keyword.
`CtrlK, E`	Insert an `@entity` keyword.
`CtrlK, X`	Insert a `@custom` keyword.

CHAPTER 7

Novel Projects

A novelWriter project requires a dedicated folder for storing its files on the local file system. See *Technical Information* for further details on how files are organised.

A new project can be created from the *Project* menu by selecting *New Project*. This will open the *New Project Wizard* that will assist you in creating a barebone project suited to your needs.

A list of recently opened projects is maintained, and displayed in the *Open Project* dialog. A project can be removed from this list by selecting it and pressing the `Del` key.

Project-specific settings are available in *Project Settings* in the *Project* menu. See further details below in the *Project Settings* section.

7.1 Project Roots

Projects are structured into a set of top level folders called *root folders*. They are visible in the project tree at the left side of the main window.

The core novel documents go into a root folder of type *Novel*. Other supporting documents go into the other root folders. These other root folder types are intended for your notes on the various elements of your story. Using them is of course entirely optional.

A new project may not have all of the root folders present, but you can add the ones you want from *Create Root Folder* in the *Project* menu.

The root folders are intended for the following use, but aside from the *Novel* folder, no restrictions are enforced by the application. You can use them however you want.

Novel This is the root folder of all text that goes into the final novel. This class of documents have other rules and features than other documents in the project. See *Novel Structure* for more details.

Plot This is the root folder where main plots can be outlined. It is optional, but adding at least dummy notes can be useful in order to tag plot elements for the Outline view. Tags in this folder can be references using the `@plot` keyword.

Characters Character notes go in this root folder. These are especially important if one wants to use the Outline view to see which character appears where, and which part of the story is told from a specific character's point-of-view. Tags in this folder can be references using the `@pov` keyword for point-of-view characters, or the `@char` keyword for other characters.

Locations The locations folder is for various scene locations that you want to track. Tags in this folder can be references using the `@location` keyword.

Timeline If the story has multiple plot timelines or jumps in time within the same plot, this class of notes can be used to track this. Tags in this folder can be references using the `@time` keyword.

Objects Important objects in the story, for instance important objects that change hands often, can be tracked here. Tags in this folder can be references using the `@object` keyword.

Entities Does your plot have many powerful organisations or companies? Or other entities that are part of the plot? They can be organised here. Tags in this folder can be references using the `@entity` keyword.

Custom The custom root folder can be used for tracking anything else not covered by the above options. Tags in this folder can be references using the `@custom` keyword.

The root folders correspond to the categories of tags that can be used to reference them. For more information about the tags listed, see *Tag References*.

Tip: You can rename root folders to whatever you want. The first character in the *Flags* column in the project tree will still indicate what type they are, and so will the icon if you are using one of the optional icon sets.

7.1.1 Deleted Documents

Deleted documents will be moved into a special *Trash* root folder. Documents in the trash folder can then be deleted permanently, either individually, or by emptying the trash from the menu. Documents in the trash folder are removed from the project index and cannot be referenced.

Folders and root folders can only be deleted when they are empty. Recursive deletion is not supported. A document or a folder can be deleted from the *Project* menu, or by pressing `CtrlDel`.

7.1.2 Archived Documents (Outtakes)

If you don't want to delete a document, or put it in the *Trash* folder where it may be deleted, but still want it out of your main project tree, you can create an *Outtakes* root folder from the *Project* menu. You are not allowed to move folders to this root folder, only documents. If you need folders in it to organise your documents, you can of course create new ones there.

You can drag any document to this folder and preserve its settings. The document will always be excluded from the *Build Novel Project* builds. It is also removed from the project index, so the tags and references defined in it will not show up anywhere else.

7.1.3 Recovered Documents

If novelWriter crashes or otherwise exits without saving the project state, or if you're using a file synchronisation tool that runs out of sync, there may be files in the project folder that aren't tracked in the core project file. These files, when discovered, are recovered and added back into the project if possible.

The discovered files are scanned for meta information that gives clues as to where the document may previously have been located in the project. The project loading routines will try to put them back as close as possible to this location,

if it still exists. Generally, it will be appended to the end of the folder where it previously was located. If that folder doesn't exist, it will try to add it to the correct root folder. If it cannot figure out which root folder is correct, the document will be added to the *Novel* root folder. Only if the *Novel* folder is missing will it give up.

If the title of the document can be recovered, the word "Recovered:" will be added as a prefix. If the title cannot be determined, the document will be named "Recovered File N" where N is a sequential number.

7.1.4 Project Lockfile

To prevent lost documents caused by file conflicts when novelWriter projects are synced with file synchronisation tools, a project lockfile is written to the project folder. If you try to open a project which has such a file present, you will be presented with a warning, and some information about where else novelWriter thinks the project is also open. You will be give the option to ignore this warning, and continue opening the project at your own risk.

Note: If, for some reason, novelWriter crashes, the lock file may remain even if there are no other instances keeping the project open. In such a case it is safe to ignore the lock file warning when re-opening the project.

Warning: If you choose to ignore the warning and continue opening the project, and multiple instances of the project are in fact open, you are likely to cause inconsistencies and create diverging project files, potentially resulting in loss of data and orphaned files. You are not likely to lose any actual text unless both instances have the same document open in the editor, and novelWriter will try to resolve inconsistencies the next time you open the project.

7.1.5 Using Folders in the Project Tree

Folders, aside from root folders, have no structural significance to the project. When novelWriter is processing the documents in the novel, like for instance during export, these folders are ignored. Only the order of the documents themselves matter.

The folders are there purely as a way for the user to organise the documents in meaningful sections and to be able to collapse and hide them in the project tree when you're not working on those documents.

Tip: You can use folders to sort your scene documents into chapters. You will still need to add a chapter document as the first item of your chapter folder, and the scene documents as the following items.

7.2 Project Documents

New documents can be created from the *Document* menu, or by pressing `CtrlN` while in the project tree. This will create a new, empty document, and open the *Item Settings* dialog where the document label and various other settings can be changed. This dialog can also be opened again later from either the *Project* menu, selecting *Edit Project Item*, or by pressing `CtrlE` or `F2` with the item selected.

The layout of the document is also defined here. For Novel documents, the full list of layout options are available. For non-Novel documents, only *Note* is available. See *Novel Document Layout* for more details.

You can also select whether the document is by default included when building the project. This setting can be overridden in the *Build Novel Project* tool if you wish to include them anyway. This is covered in the *File Selection* section. You can also toggle the included state of a document from the right-click context menu.

7.2.1 Word Counts

A character, word and paragraph count is maintained for each document, as well as for each section of a document following a header. The word count, and change of words in the current session, is displayed in the footer of any document open in the editor, and all stats are shown in the details panel below the project tree for any document selected in the project tree.

The word counts are not updated in real time, but run in the background every five seconds for as long as the document is being actively edited.

A total project word count is displayed in the status bar. The total count depends on the sum of the values in the project tree, which again depend on an up to date index. If the counts seem wrong, a full project word recount can be initiated by rebuilding the project's index. Either form the *Tools* menu, or by pressing `F9`.

7.3 Project Settings

The *Project Settings* can be accessed from the *Project* menu, or by pressing `CtrlShift,`. This will open a dialog box, with a set of tabs.

7.3.1 Settings Tab

The *Settings* tab holds the project title and author settings.

The *Working Title* can be set to a different title than the *Book Title*. The difference between them is simply that the *Working Title* is used for the GUI (main window title) and for generating the backup files. The intention is that the *Working Title* should remain unchanged throughout the project, otherwise the name of exported files and backup files may change too.

The *Book Title* and *Book Authors* settings are currently not used for anything, so setting then is just for the benefit of the author. Future features may be using them, and they are exported on some export formats in the *Build Novel Project* tool.

If your project is in a different language than your main spell checking is set to, you can override the default spell checking language here. You can also override the automatic backup setting.

7.3.2 Details Tab

This tab presents an overview of technical meta data for the project. It states where on your file system the project is saved, how may times it has been saved, how many folders and documents it contains, and how many words exist in the entire project.

7.3.3 Status and Importance Tabs

Each document or folder of type *Novel* can be given a status level, signified by a coloured icon, and each document or folder of the remaining types can be given an importance level. These are colour coded icons and labels that can be applied to each document or folder.

These are purely there for the user's convenience, and you are not required to use them for any other features to work. No other part of novelWriter accesses this information. The intention is to use these to indicate at what stage of completion each novel document is, or how important the content of a note is to the plot. You don't have to use them this way, that's just what they were intended for, but you can make them whatever you want.

Note: The status or importance level currently in use by one or more documents cannot be deleted, but they can be edited.

7.3.4 Auto-Replace Tab

A set of automatically replaced keywords can be added in this tab. The keywords in the left column will be replaced by the text in the right column when documents are opened in the viewer. They will also be applied to exports.

The auto-replace feature will replace text in angle brackets that are in this list. The syntax highlighter will add an alternate colour to text marching the syntax, but it doesn't check if the text is in this list.

Note: A keyword cannot contain spaces. The angle brackets are added by default, and when used in the text are a part of the keyword to be replaced. This is to ensure that parts of the text aren't unintentionally replaced by the content of the list.

7.4 Backup

An automatic backup system is built into novelWriter. In order to use it, a backup path to where the backup files are to be stored must be provided in *Preferences*.

Backups can be run automatically when a project is closed, which also implies it is run when the application itself is closed. Backups are date stamped zip files of the entire project folder, and are stored in a subfolder of the backup path. The subfolder will have the same name as the project *Working Title* set in *Project Settings*.

The backup feature, when configured, can also be run manually from the *Tools* menu. It is also possible to disable automated backup for a given project in *Project Settings*.

Note: For the backup to be able to run, the *Working Title* must be set in *Project Settings*. This value is used to generate the folder name for the zip files. Without it, the backup will not run at all, but it will produce a warning message.

7.5 Writing Statistics

When you work on a project, a log file records when you opened it, when you closed it, and the total word counts of your novel documents and notes at the end of the session. You can view this file in the `meta` folder in the directory where you saved your project. The file is named `sessionStats.log`.

A tool to view the content of this file is available in the *Tools* menu under *Writing Statistics*. You can also launch it by pressing `F6`.

The tool will show a list of all your sessions, and a set of filters to apply to it. You can also export the filtered data to a JSON file or to a CSV file that can be opened by a spreadsheet application like for instance Libre Office Calc.

CHAPTER 8

Novel Structure

This section covers the structure of a novel project.

It concerns documents under the *Novel* type root folder only. There are some restrictions and features that only apply to these types of documents.

8.1 Importance of Headings

Subfolders under root folders have no impact on the structure of the novel itself. The structure is instead dictated by the heading level of the headers within the documents.

Four levels of headings are supported, signified by the number of hashes (#) preceding the title. See also the *Markdown Format* section for more details about the markdown syntax.

Note: The header levels are not only important when generating the exported novel file, they are also used by the indexer when building the outline tree in the *Outline* tab. Each heading also starts a new region where new references and tags can be set.

The different header levels are interpreted as specific section types of the novel in the following way:

`# Header1` Header level one signifies that the text refers to either the novel title or the name of a top level partition. The latter is useful when you want to split the manuscript up into books, parts, or acts.

`## Header2` Header level two signifies a chapter level partition. Each time you want to start a new chapter, you must add such a heading. If you choose to split your manuscript up into one document per scene, you need a single chapter document with just the heading. You can of course also add a synopsis and reference keywords to the chapter document. If you want to open the chapter with a quote or other introductory text that isn't part of a scene, this is also where you'd put that text.

`### Header3` Header level three signifies a scene level partition. You must provide a title text, but the title text can be replaced with a scene separator or just skipped entirely when you export your manuscript.

Header4 Header level four signifies a sub-scene level partition, usually called a "section" in the documentation and the user interface. These can be useful if you want to change tag references mid-scene, like if you change the point-of-view character. You are free to use sections as you wish, and can filter the titles out of the final manuscript just like with scene titles.

Tip: There are multiple options of how to process novel titles when exporting the manuscript. For instance, chapter numbers can be applied automatically, and so can scene numbers if you want them in a draft manuscript. See the *Exporting Projects* page for more details.

8.1.1 Unnumbered Chapter Headings

If you use layout types for your documents, the automatic numbering feature for your chapters is controlled by whether you use the *Chapter* or *Unnumbered* layout type for your document. However, if you have a different document layout where this isn't practical, you can also switch off chapter numbering for a chapter by making the first character of the chapter title an asterisk (`*`). Like so:

```
## *Unnumbered Chapter Title
```

The leading asterisk is only considered by the *Build Novel Project* tool, and will be removed before inserted at the location of the `%title%` label. See the *Exporting Projects* page for more details.

Note: If you need the first character of the title to be an actual asterisk, you must escape it: `\*`.

8.2 Tag References

Each text partition indicated by a heading of any level, can contain references to tags set in the supporting notes of the project. The references are gathered by the indexer and used to generate an outline view on the *Outline* tab of how the different parts of the novel are connected.

References and tags are also clickable in the document editor and viewer, making it easy to navigate between reference notes while writing. Clicked links are always opened in the view panel.

References are set as a keyword and a list of corresponding tags. The valid keywords are listed below. The format of a reference line is `@keyword: value1, [value2] ... [valueN]`. All keywords allow multiple values.

@pov The point-of-view character for the current section. The target must be a note tag in the *Character* type root folder.

@char Other characters in the current section. The target must be a note tag in a *Character* type root folder. This should not include the point-of-view character.

@plot The plot or subplot advanced in the current section. The target must be a note tag in a *Plot* type root folder.

@time The timelines touched by the current section. The target must be a note tag in a *Timeline* type root folder.

@location The location the current section takes place in. The target must be a note tag in a *Locations* type root folder.

@object Objects present in the current section. The target must be a note tag in an *Object* type root folder.

@entity Entities present in the current section. The target must be a note tag in an *Entities* type root folder.

@custom Custom references in the current section. The target must be a note tag in a *Custom* type root folder.

The syntax highlighter will alert the user that the tags and references are used correctly, and that the tags referenced exist.

The highlighter may be mistaken if the index of defined tags is out of date. If so, press `F9` to regenerate it, or select *Rebuild Index* from the *Tools* menu. In general, the index for a document is regenerated when it is saved, so this shouldn't normally be necessary.

8.3 Novel Document Layout

All documents in the project can have a layout format set. These layouts are important when the project is exported as they indicate how to treat the content in terms of text formatting, headings, and page breaks. The layout for each document is indicated as the last set of characters in the *Flags* column of the project tree.

Not all layout types are actually treated differently, they also help to indicate what each document is intended for in your project. The *Book* layout is a generic novel document layout that is formatted identically to *Chapter* and *Scene* layout documents, but may help to indicate what each document does in your project.

You can for instance lay out your project using *Book* documents for each act, and then later split those into chapter or scene documents by using the *Split Document* tool. Scenes can also be contained within *Chapter* type documents, but you lose the drag and drop feature that comes with having them in separate documents if you organise them this way.

Some layouts *do* have implications on how the project is exported. Documents with layout *Title Page* and *Partition* have all headings and text centred, while the *Unnumbered* layout disables the automatic chapter numbering feature for everything contained within it. The latter is convenient for Prologue and Epilogue type chapters.

The above layout formats are only usable in the Novel root folder. Documents that are not a part of the novel itself should have the *Note* layout. These documents are not getting any special formatting, and it is possible to collectively filter them out during export. Notes can be used anywhere in the project, also in the *Novel* root folder.

Below is an overview of all available layout formats.

Title Page The title page layout. The title should be formatted as a heading level one. All text is centred on export.

Plain Page A plain page layout useful for instance for front matter pages. Heading levels are ignored for this layout format, and so are formatting options like *Justify Text*. The page is exported with a page break before it.

Book This is the generic novel format that in principle can be used for all novel documents. Since the internal structure of the novel is controlled by the heading levels, this layout will produce the same result as a collection of *Partition*, *Chapter* and *Scene* layout documents. However, it does not provide the functionality of the *Unnumbered* layout format by default, but this can still be achieved by prefixing the chapter title with an asterisk (`*`).

Partition A partition can be used to split the novel into parts. Partition titles are indicated with a level one heading. You can also add text and meta data to the page. The *Partition* layout will in addition force a page break before the heading, and centre all content on the page.

Chapter Signifies the start of a new chapter. If the text itself is contained in scene documents, these documents should only contain the title, comments, synopsis, and tag references for characters, plot, etc. The heading for chapters should be level two. If you need an opening text, like a quote or other leading text before the first scene, this is also where you'd want to add this text.

Unnumbered Same as *Chapter*, but when exporting the project, and automatic chapter numbering is enabled, documents with this layout will not increment the chapter number. It also has a separate title formatting setting. This makes the layout suitable for Prologue and Epilogue type chapters.

Scene Used for scenes. This document should have a header of level three. Further sections can have headers of level four, but there are no layout specifically for sections.

Note A generic document that is optionally ignored when the novel project is exported. Use this layout for descriptions of content in the supporting root folders. Notes can also be added to the *Novel* root folder if you need to insert notes there. Note headers receive no special formatting when building the project. They are always exported as-is.

Note: The layout granularity is entirely optional. In principle, you can write the entire novel in a single document with layout *Book*. You can also have a single document per chapter if that suits you better. The *Outline* will show your structure of chapters and scenes regardless of how your documents are organised.

Tip: You can always start writing with a coarse layout with one or a few documents, and then later use the split tool to automatically split the documents into separate chapter and scene documents.

CHAPTER 9

Project Notes

novelWriter doesn't have a database and complicated forms for filling in details about plot elements, characters, and all sorts of additional information that isn't a part of the novel text itself. Instead, all such information is saved in notes. The relation between all these additional elements is extracted from the documents and notes by the project indexer, based on the tags and references you set within them.

Using notes is not required, but making at least minimal notes for each plot element, and adding a tag to them, makes it possible to use the *Outline* feature to see how each element intersects with each section of the novel itself, and adds clickable cross-references between documents in the editor and viewer.

9.1 Tags in Notes

Each new heading in a note can have a tag associated with it. The format of a tag is `@tag: tagname`, where tagname is a unique identifier. Tags can then be referenced in the novel documents, or cross-referenced in other notes, and will show up in the outline view and in the back-reference panel when a document is being viewed.

The syntax highlighter will alert the user that the keyword is correctly used and that the tag is allowed, that is, the tag is unique. Duplicate tags should be detected as long as the index is up to date. An invalid tag should have a green wiggly line under it, and will not receive the syntax colour that valid tags do.

The tag is the only part of these notes that the application uses. The rest of the document content is there for the writer to use in whatever way they wish. Of course, the content of the documents can be exported if you want to compile a single document of all your notes, or include them in an outline.

A note can also reference other notes in the same way novel documents do. When the note is opened in the view panel, the references become clickable links, making it easier to follow connections in the plot. Notes don't show up in the outline view though, so referencing between notes is only meaningful if you want to be able to click-navigate between them.

Tip: If you cross-reference between notes and export your project as an HTML document using the *Build Novel Project* tool, the cross-references become clickable links in the exported HTML document.

CHAPTER 10

Exporting Projects

The novelWriter project can be exported in various formats using the build tool available from *Build Novel Project* in the *Tools* menu, or by pressing `F5`.

10.1 Header Formatting

The titles for the five types of titles (the chapter headings come in a numbered and unnumbered version) of story structure can be formatted collectively in the build tool. This is done through a series of keyword–replace steps. They are all on the format `%keyword%`.

`%title%` This keyword will always be replaced with the title text you put after the # characters in your document.

`%ch%` This will be replaced by a chapter number. The number is incremented by one each time the build tool sees a new heading of level two in a document with layout *Chapter*. If the document has layout *Unnumbered*, the counter is *not* incremented. The latter is useful for for instance Prologue and Epilogue chapters. Adding an asterisk (`*`) in front of the title text of a level two heading will also disable the chapter counter for that heading.

`%chw%` Behaves like `%ch%`, but the number is represented as a number word.

`%chi%` Begaves like `%ch%`, but the number is represented as a lower case Roman number.

`%chI%` Behaves like `%ch%`, but the number is represented as an upper case Roman number.

`%sc%` This is the number counter equivalent for scenes. These are incremented each time a heading of level three is encountered, but reset to 1 each time a chapter is encountered. They can thus be used for counting scenes within a chapter.

`%sca%` Behaves like `%sc%`, but the number is *not* reset to 1 for each chapter. Instead it runs from 1 from the beginning of the novel to produce an absolute scene count.

`\\` This inserts a line break within the title.

Note: Header formatting only applies to novel documents. Headings in notes will be left as-is on export. However, heading levels 1 through 4 are converted to the correct heading level in the respective output formats.

Example

- The format `%title%` just reproduces the title you set in the document.
- The format `Chapter %ch%: %title%` produces something like "Chapter 1: My Chapter Title".
- The format `Scene %ch%.%sc%` produces something like "Scene 1.2" for scene 2 in chapter 1.

10.2 Scene Separators

If you don't want any titles for your scenes (or for your sections if you have them), you can leave the formatting boxes empty. If so, an empty paragraph will be inserted between the scenes or sections instead.

Alternatively, if you want a separator between them, like the common `* * *`, you can enter the desired separator text in the formatting box. In fact, if the format is a piece of static text, it will always be treated as a separator.

10.3 File Selection

Which documents and notes are selected for export can be controlled from the options on the left side of the dialog window. The switch for *Include novel files* will select any document that isn't classified as a note. The switch for *Include note files* will select any document that *is* a note. This allows for exporting just the novel, just your notes, or both, as you see fit.

In addition, you can select to export the synopsis comments, regular comments, keywords, and even exclude the body text itself.

Tip: If you for instance want to export a document with an outline of the novel, you can enable keywords and synopsis export and disable body text, thus getting a document with each heading followed by the tags and references and the synopsis.

If you need to exclude specific documents from your exports, like draft documents or documents you want to take out of your manuscript, but don't want to delete, you can un-check the *Include when building project* option for each document in the project tree. An included document has a checkmark after the status icon in the *Flags* column. The *Build Novel Project* tool has a switch to ignore this flag if you need to collectively override these settings.

10.4 Export Formats

Currently, six formats are supported for exporting.

OpenDocument Format This produces an open document `.odt` file. The document produced has very little formatting, and may require further editing afterwards. For a better formatted office document, you may get a better result by exporting to HTML and then importing that HTML document into your office word processor. They are generally good at importing HTML documents.

PDF Format The PDF export is just a shortcut for print-to-file. For a better PDF result, you may instead want to export to HTML and use a word processor to convert the HTML document to PDF.

novelWriter HTML The HTML export format writes a single `.htm` file with minimal style formatting. The exported HTML document is suitable for further processing by document conversion tools like Pandoc, for importing in word processors, or for printing from browser. It is generally the best formatted export option and supports all features of novelWriter since it is entirely generated by the application and doesn't depend on Qt library features.

novelWriter Markdown This is simply a concatenation of the project documents selected by the filters. The documents are stacked together in the order they appear in the project tree, with comments, tags, etc. included if they are selected. This is a useful format for exporting the project for later import back into novelWriter.

Standard Markdown If you have Qt 5.14 or higher, the option to export to plain markdown is available. This feature uses Qt's own markdown export feature.

Plain Text The plain text export format writes a simple `.txt` file without any formatting at all.

10.5 Additional Export Options

In addition to the above document formats, the novelWriter HTML and Markdown formats can also be wrapped in a JSON file. These files will have a meta data entry and a body entry. For HTML, also the accompanying css styles are exported.

The text body is saved in a two-level list. The outer list contains one entry per exported document, in the order they appear in the project tree. Each document is then split up into a list as well, with one entry per paragraph it contains.

These files are mainly intended for scripted post-processing for those who want that option. A JSON file can be imported directly into a Python dict object or a PHP array, to mentions a few options.

CHAPTER 11

Typographical Notes

novelWriter has some support for typographical symbols that are not usually easily available in many text editors. This includes for instance the proper unicode quotation marks, dashes, ellipsis, thin spaces, etc. All these symbols are available from the *Insert* menu, and via keyboard shortcuts. See *Insert Shortcuts*.

This chapter provides some additional information on how novelWriter handles these symbols.

11.1 Special Notes on Symbols

Some additional notes on these symbols.

11.1.1 Dashes and Ellipsis

With the auto-replace feature enabled (see *Auto-Replace as You Type*), multiple hyphens are converted automatically to short and long dashes, and three dots to ellipsis. The last auto-replace can always be reverted with the undo command `CtrlZ`, reverting the text to what you typed before the automatic replacement occurred.

11.1.2 Single and Double Quotes

All the different quotation marks listed on the Quotation Mark Wikipedia page are available, and can be selected as auto-replaced symbols for straight single and double quote key strokes. The settings can be found in *Preferences*.

Ordinarily, text wrapped in quotes are highlighted by the editor. This is meant as a convenience for highlighting dialogue between characters. This feature can be disabled in *Preferences* if this feature isn't wanted.

The editor distinguishes between text wrapped in straight quotes and with the user-selected double quote symbols. This is to help the writer recognise which parts of the text are not using the chosen quote symbols. Two convenience functions in the *Format* menu can be used to re-format a selected section of text with the correct quote symbols.

11.1.3 Modifier Letter Apostrophe

The auto-replace feature will consider any right-facing single straight quote as a quote symbol, even if it's intended as an apostrophe. This also includes the syntax highlighter, which may assume the first following apostrophe is the closing symbol of a single quoted region of text.

To get around this, an alternative apostrophe is available. It is a special Unicode character that is not categorised as punctuation, but as a modifier. It is usually renderred the same way as the right single quotation marks, depending on the font. There is a Wikipedia article for the Modifier letter apostrophe with more details.

Note: On export with the *Build Novel Project* tool, these apostrophes will be replaced automatically with the corresponding right hand single quote symbol as is generally recommended. Therefore it doesn't really matter if you only use them to correct highlighting.

11.1.4 Special Space Symbols

A few variations of the regular space character is supported. The correct typographical way to separate a number from its unit is with a thin space. It is usually 2/3 the width of a regular space. For numbers and units, this should in addition be a non-breaking space, that is, the text wrapping should not add a line break on this particular space.

A regular space can also be made into a non-breaking space if needed.

All non-breaking spaces are highlighted with a differently coloured background to make it easier to spot them in the text. The colour will depend on the selected colour theme.

The thin and non-breaking spaces are converted to their corresponding HTML codes on export to HTML format. For plain text, they are exported as regular spaces.

CHAPTER 12

Technical Information

This section contains details of how novelWriter stores and handles the project data.

12.1 How Project Data is Stored

All novelWriter files are written with utf-8 encoding. Since Python automatically converts Unix line endings to Windows line endings on Windows systems, novelWriter does not make any adaptations to the formatting on Windows systems. This is handled entirely by the Python standard library. Python also handles this fairly well when working on the same files on both Windows and Unix-based operating systems.

12.1.1 Main Project File

The project itself requires a dedicated folder for storing its files where novelWriter will create its own "file system" where the folder and file hierarchy is described in a project XML file. This is the main project file in the project's root folder with the name `nwProject.nwx`. This file also contains all the meta data required for the project, and a number of related project settings.

If this file is lost or corrupted, the structure of the project is lost, although not the text itself. It is important to keep this file backed up, either through the built-in backup tool, or your own backup solution.

Tip: The novelWriter project folder is structured so that it can easily be added to a version control system like git. If so, you may want to add a *.gitignore* file to exclude files with the extensions *.json* as JSON files are used to cache the index and various run-time settings and are generally large files that change often. You'd also want to exclude the `cache` folder.

The project XML file is indent-formatted, suitable for diff tools and version control since most of the file will stay static, although a timesetamp is set in the meta section on line 2 each time the file is saved, and various meta data entries are incremented on each save.

12.1.2 Project Documents

The project documents are saved in a folder in the main project folder named `content`. Each document has a file handle taken from the first 13 characters of a SHA256 hash of the system time when the document was first created. The documents are saved with a filename assembled from this hash and the file extension `.nwd`.

If you wish to find the physical location of a document in the project, you can either look it up in the project XML file, select *Show File Details* from the *Document* menu when having the document open, or look in the `ToC.txt` file in the root of the project folder. The `ToC.txt` file has a list of all documents in the project and where they are saved.

The reason for this cryptic file naming is to avoid issues with file naming conventions and restrictions on different operating systems, and also to have a file name that does not depend on what the user names the document within the project, or changes it to. The file meta data in the project tree is mainly saved in the project XML file, although some basic meta data is added to the header of each document file.

Each document file contains a plain text version of the text from the editor. The file can in principle be edited in any text editor, and is suitable for diffing and version control if so desired. Just make sure the file remains in utf-8 encoding, otherwise unicode chatracters may become mangled when the file is opened in novelWriter again.

The first lines of the file may contain some meta data starting with the characters `%%~`. These lines are mainly there to restore some information if it is lost from the project file, and the information may be helpful if you do open the file in an external editor as it contains the document label and the document class and layout. The lines can be deleted without any consequences to the rest of the content of the file, and will be added back the next time the document is saved in novelWriter.

12.1.3 The File Saving Process

When saving the project file, or any of the documents, the data is first saved to a temporary file. If successful, the old data file is removed, and the temporary file becomes the new file. This ensures that the previously saved data is only replaced when the new data has been successfully saved to the storage medium.

For the project XML file, a `.bak` file is in addition kept, which will always contain the previous version of the file, although when auto-save is enabled, they may have the same content. If the opening of a project file fails, novelWriter will automatically try to open the `.bak` file instead.

12.2 Project Meta Data

The project folder contains a subfolder named `meta`, containing a number of files. The meta folder contains semi-important files. That is, they can be lost with only minor impact to the project.

If you use version control software on your project, you can exclude this folder, although you may want to track the session log file. The JSON files within this folder can safely be ignored.

12.2.1 The Project Index

Between writing sessions, the project index is saved in a JSON file in `meta/tagsIndex.json`. This file is not critical. If it is lost, it can be rebuilt from within novelWriter from the *Tools* menu.

The index is maintained and updated whenever a document or note is saved in the editor. It contains all references and tags in documents and notes, as well as the location of all headers in the project, and the word counts within each header section.

While the integrity of the index is checked when the file is loaded, the check is not very deep and it is possible to corrupt the index if the file is manually edited and manipulated. If so, novelWriter may crash during launch. If this happens, you must delete the index file and rebuild the index.

12.2.2 Cached GUI Options

A file named `meta/guiOptions.json` contains the latest state of various GUI buttons, switches, dialog window sizes, column sizes, etc, from the GUI. These are the GUI settings that are specific to the project. Global GUI settings are stored in the main config file.

The file is not critical, but if it is lost, all such GUI options will revert back to their default settings.

12.2.3 Session Stats

The writing progress is saved in the `meta/sessionStats.log` file. This file records the length and word counts of each writing session on the given project. The file is used by the *Writing Statistics* tool. If this file is lost, the history it contains is also lost, but it has otherwise no impact on the project.

12.3 Project Cache

The project `cache` folder contains non-critical files. If these files are lost, there is no impact on the functionality of novelWriter or the history of the project. It contains temporary files, like the preview document in the *Build Novel Project* tool.

It should be excluded from version control tools if such are used.

CHAPTER 13

Indices and Tables

- genindex
- modindex
- search