novelWriter Documentation

Release 2.0.1-post1

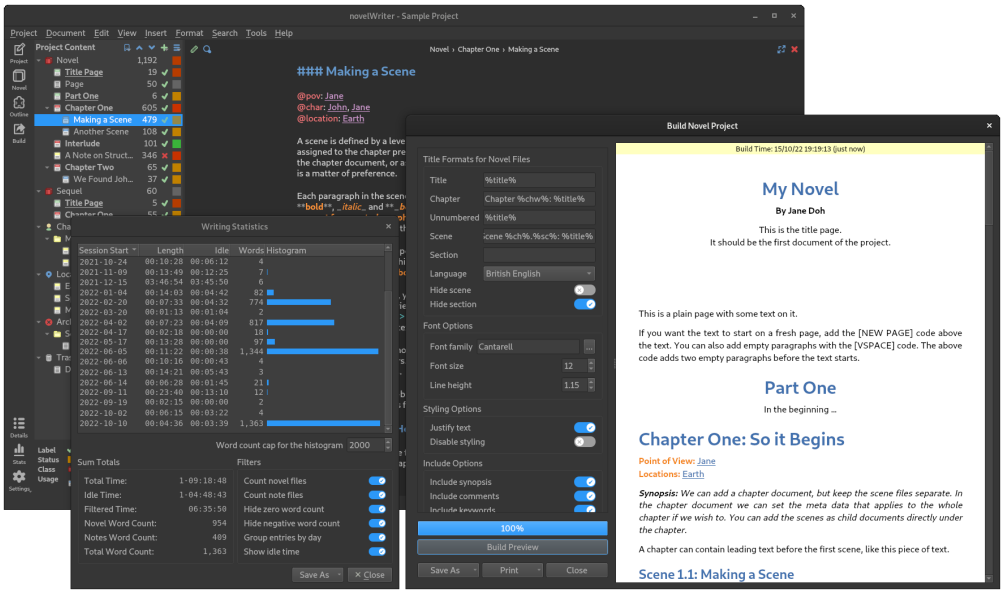
Veronica Berglyd Olsen

Tuesday, 29 November 2022 at 17:26

INTRODUCTION

For Release: 2.0.1-post1
Last Updated: Tuesday, 29 November 2022 at 17:26

novelWriter is an open source plain text editor designed for writing novels assembled from many smaller text documents. It uses a minimal formatting syntax inspired by Markdown, and adds a meta data syntax for comments, synopsis, and cross-referencing. It is designed to be a simple text editor that allows for easy organisation of text and notes, using human readable text files as storage for robustness.



The project storage is suitable for version control software, and also well suited for file synchronisation tools. All text is saved as plain text files with a meta data header. The core project structure is stored in a single project XML file. Other meta data is primarily saved as JSON files. See the *Project Storage* section for more details.

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 as well. novelWriter can also be run directly from the Python source, or installed from packages or the pip tool. See *Getting Started* for more details.

Note: Version 1.5 introduced a few changes that will require you to make some minor modifications to some of the headings in your project. It should be fairly quick and straightforward. Please see the *Format 1.3 Changes* section for more details.

Useful Links

- Website: https://novelwriter.io
- Documentation: https://novelwriter.readthedocs.io

- Internationalisation: https://crowdin.com/project/novelwriter
- 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
- Social Media: https://fosstodon.org/@novelwriter

CHAPTER

ONE

KEY FEATURES

At the core, novelWriter is a multi-document plain text editor. It uses a markup syntax inspired by Markdown to apply simple formatting to the text. It is designed for writing fiction, so the formatting features are limited to those relevant for this purpose. It is not suitable for technical writing.

Your novel project is organised as a collection of separate plain text documents instead of a single, large document. The idea here is to make it easier to reorganise your project structure without having to cut and paste text.

You can at any point split documents by their header, or merge multiple documents into single documents. This makes it easier to use variations of the popular Snowflake method for writing.

Below are some key features of novelWriter.

Focus on writing
: The aim of the user interface is to let the user focus on writing instead of spending time formatting text. Formatting is therefore limited to a small set of formatting tags for simple things like text emphasis and paragraph alignment. When you really want to focus on just writing, you can switch the editor into *Focus Mode* where only the text editor window itself is vissible.

Keep an eye on your notes
: The main window can optionally show a document viewer to the right of the editor. This view panel is intended for displaying another scene document, you character notes, plot notes, or any other document you may need to reference while writing.

Organise your documents how you like
: You can split your novel project up into as many individual documents as you want to. When you build the project, they are all glued together in the top-to-bottom order in which they appear in the project tree. You can use as few text documents as you like, but splitting the project up into chapters and scenes means you can easily reorder them using the drag and drop feature. You can start out with a few documents and then later split the document into multiple documents based on its headers.

Keep track of your plot elements
: All notes in your project can be assigned a *tag* you can *reference* from any other document or note. In fact, you can add a new tag under each heading of a note if you need to be able to reference a specific section. Note tags are organised into categories with specific reference keywords.

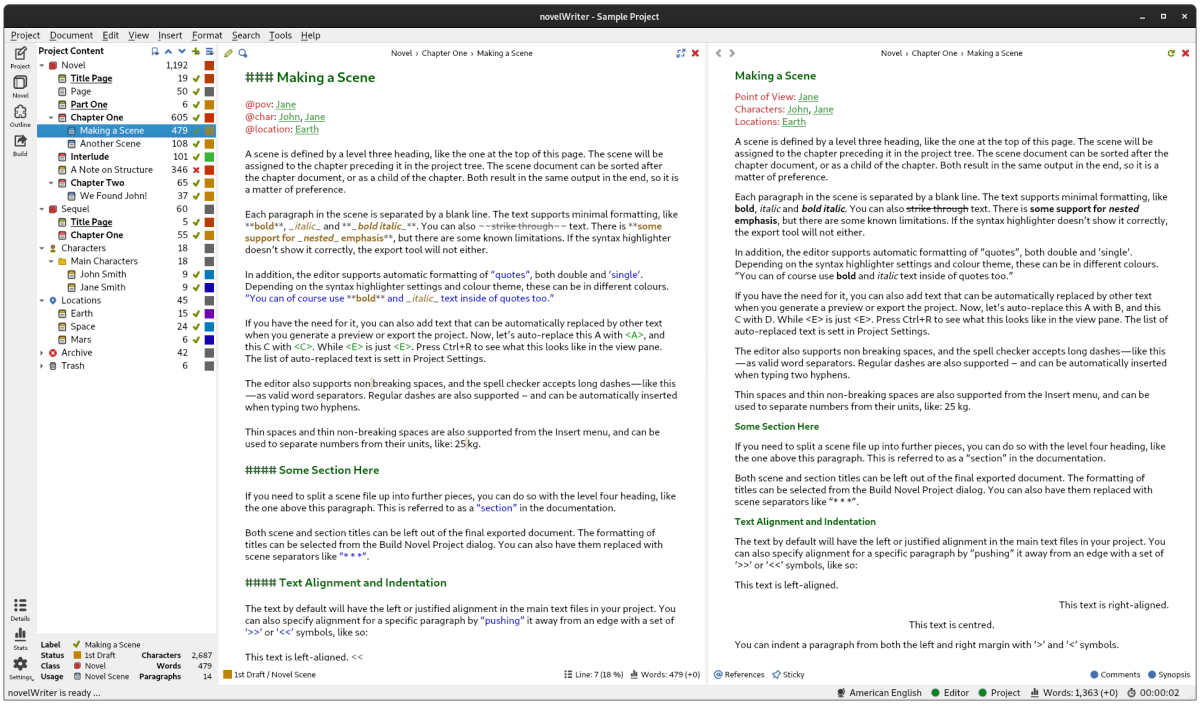
Get an overview of your plot elements
: In the *Outline View* on the main window you can see an outline of all the chapters, scenes, and sections of your project. If they have any references in them, these are listed in additional columns. You can also add a synopsis to each document, which can be listed here as well. You have the option to add or remove columns of information from this outline. A subset of the outline information is also available in the *Novel View* as a replacement for the main project tree.

Building your manuscript

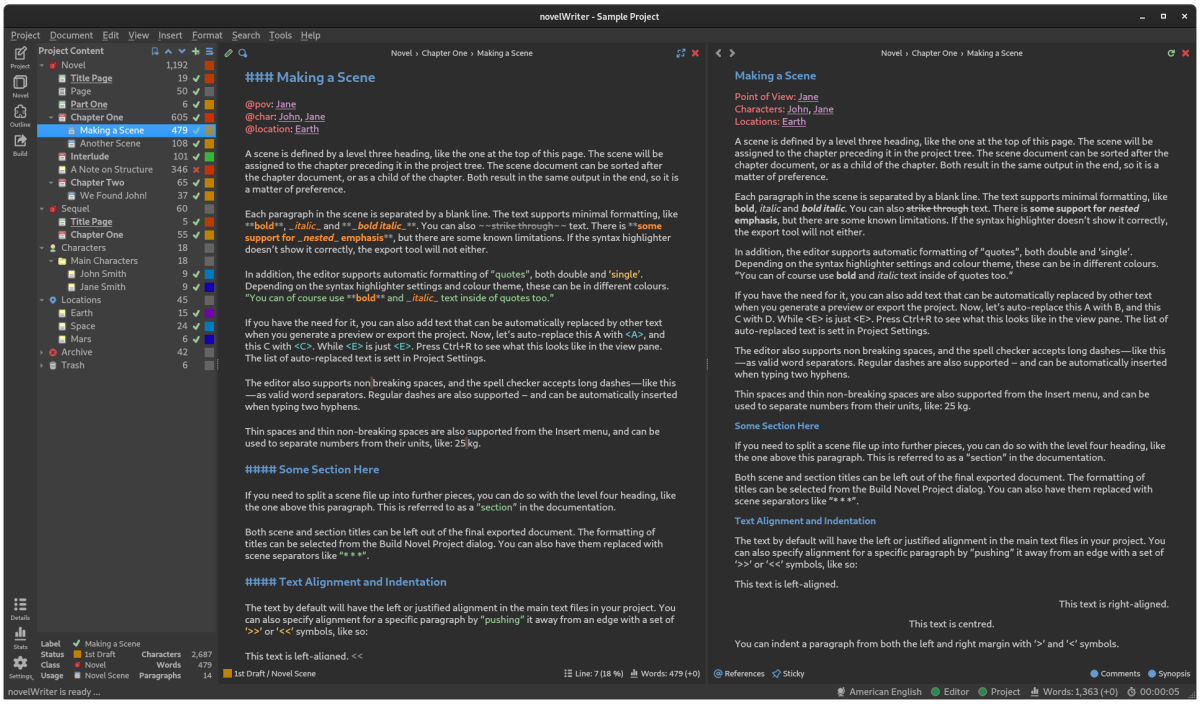
Whether you want to compile a manuscript, or export all your notes, or generate an outline of your chapters and scenes with a synopsis, you can use the *Build Novel Project* tool. The tool lets you select what information you want to include in the generated document, and how it is formatted. You can send the result to a printer, a PDF, or to an Open Document file that can be opened by most office type word processors. You can also generate the result as HTML, or Markdown, both suitable for further conversion to other formats.

1.1 Screenshots

novelWriter with default theme:



novelWriter with dark theme:


novelWriter - Sample Project
Project Document Edit View Insert Format Search Tools Help
Project Content
Novel 1,192
Title Page 19
Page 50
Part One 6
Chapter One 605
Making a Scene 479
Another Scene 108
Interlude 101
A Note on Structure 346
Chapter Two 65
We Found John! 37
Sequel 60
Title Page 5
Chapter One 55
Characters 18
Main Characters 18
John Smith 9
Jane Smith 9
Locations 45
Earth 15
Space 24
Mars 6
Archive 42
Trash 6
Label Making a Scene
Status 1st Draft Characters 2,687
Class Novel Words 479
Usage Novel Scene Paragraphs 14
Novel › Chapter One › Making a Scene
Making a Scene
@pov: Jane
@char: John, Jane
@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. The scene document can be sorted after the chapter document, or as a child of the chapter. Both result in the same output in the end, so it is a matter of preference.
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 there are some known limitations. 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 settings and colour theme, 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 list of auto-replaced text is sett in Project Settings.
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 the one above this paragraph. This is referred to as a "section" in the documentation.
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 "* * *".
Text Alignment and Indentation
The text by default will have the left or justified alignment in the main text files in your project. You can also specify alignment for a specific paragraph by "pushing" it away from an edge with a set of '>>' or '<<' symbols, like so:
This text is left-aligned. <<
1st Draft / Novel Scene
Line: 7 (18 %) Words: 479 (+0)
Novel › Chapter One › Making a Scene
Making a Scene
Point of View: Jane
Characters: John, Jane
Locations: 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. The scene document can be sorted after the chapter document, or as a child of the chapter. Both result in the same output in the end, so it is a matter of preference.
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 there are some known limitations. 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 settings and colour theme, 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 B, and this C with D. While <E> is just <E>. Press Ctrl+R to see what this looks like in the view pane. The list of auto-replaced text is sett in Project Settings.
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 the one above this paragraph. This is referred to as a "section" in the documentation.
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 "* * *".
Text Alignment and Indentation
The text by default will have the left or justified alignment in the main text files in your project. You can also specify alignment for a specific paragraph by "pushing" it away from an edge with a set of '>>' or '<<' symbols, like so:
This text is left-aligned.
This text is right-aligned.
This text is centred.
You can indent a paragraph from both the left and right margin with '>' and '<' symbols.
References Sticky
Comments Synopsis
novelWriter is ready ...
American English Editor Project Words: 1,363 (+0) 00:00:05

CHAPTER

TWO

OVERVIEW

novelWriter is built on Python 3, a cross platform programming language that doesn't require a compiler to build and run. That means that the code can run on your computer right out of the box, or from a zip file.

While it is developed for Linux primarily, it runs just fine on Windows as well. It also works fine on macOS, but the author is not a mac user, so support for mac is dependant on user feedback and contributions.

In order to run novelWriter, you also need a few additional packages. The user interface is built with Qt 5, a cross platform library for building graphical user interface applications. It also uses a third party XML package. If you want spell checking, you also need the spell check package Enchant developed for AbiWord

For install instructions, see *Getting Started*.

For information on how to add spell check dictionaries, see *Spell Check Dictionaries*.

2.1 Using novelWriter

In order to use novelWriter effectively, you need to know the basics of how it works. The following sections will explain the main principles. It starts with the basics, and gets more detailed as you read on.

How it Works **– Essential Information**
: This section explains the basics of how the application works and what it can and cannot do.

The User Interface **– Recommended Reading**
: This section will give you a more detailed explanation of what the various elements on the user interface do and how you can use them more effectively.

Formatting Your Text **– Essential Information**
: This section covers how you should format your text. The editor is plain text, so text formatting requires some basic markup. The structure of your novel is also inferred by how you use headings. Tags and references are implemented by simple codes.

Keyboard Shortcuts **– Optional / Lookup**
: This section lists all the keyboard shortcuts in novelWriter and what they do. Most of the shortcuts are also listed next to the menu items inside the app, or in tool tips, so this section is mostly for reference.

Typographical Notes **– Optional**
: This section gives you an overview of the special typographical symbols available in novelWriter. The auto-replace feature can handle the insertion of standard quote symbols for your language, and

other special characters. If you use any symbols aside from these. their intended use is explained here.

Project Format Changes **– Optional**
This section is more technical and has an overview of changes made to the way your project data is stored. The format has changed a bit from time to time, and sometimes the changes require that you make small modifications to your project. Everything you need to know is listed in this section.

2.2 Organising Your Projects

In addition to manage a collection of plain text files, novelWriter can interpret and map the structure of your novel and show you additional information about its flow and content. In order to take advantage of these features, you must structure your text in a specific way and add some meta data for it to extract.

Novel Projects **– Essential Information**
This section explains how you organise the content of your project, and how to set up automated backups of your work.

Novel Structure **– Essential Information**
This section covers the way your novel's structure is encoded into the text documents. It explains how the different levels of headings are used, and how you can include information about characters, plot elements, and other meta data in your text.

Project Notes **- Recommended Reading**
This section briefly describes what novelWriter does with the note files you add to your project. Generally, the application doesn't do much with them at all aside from looking through them for tags you've set so that it knows which file to open when you click on a reference.

Building the Manuscript **- Recommended Reading**
This section explains in more detail how the build tool works. In particular how you can control the way chapter titles are formatted, and how scene and section breaks are handled.

CHAPTER

THREE

GETTING STARTED

If you are using Windows or a Debian-based Linux distribtuion, you can install novelWriter from package installers. If you are on macOS, you have the option to run novelWriter from a standalone folder. See *Minimal Package Install*. This option is also available for Windows and Linux. The third option is to install novelWriter from the Python Package Index. See *Install from PyPi*.

Spell checking in novelWriter is provided by a third party library called Enchant. Generally, it should pull dictionaries from your operating system automatically. However, on Windows they must be installed manually. See *Spell Check Dictionaries* for more details.

Help Wanted

If you would like to help making more installers, the project is currently looking for people who can help make releases for Red Hat-based Linux distros (RPM) and for macOS. See the issues posted for RPM and macOS on GitHub.

3.1 Install on Windows

You can install novelWriter with both Python and library dependencies embedded using the Windows Installer (setup.exe) file from the main website, or from the Releases page on GitHub. Installing it should be straightforward.

If you have any issues, try uninstalling the previous version and making a fresh install. If you already had a version installed via a different method, you should uninstall that first.

3.2 Install on Linux

A Debian package can be downloaded from the main website, or from the Releases page on GitHub. This package should work on both Debian, Ubuntu and Linux Mint, at least.

If you prefer, you can also add the novelWriter repository on Launchpad to your package manager.

3.2.1 Ubuntu

You can add the Ubuntu PPA and install novelWriter with the following commands.

```
sudo add-apt-repository ppa:vkbo/novelwriter
sudo apt update
sudo apt install novelwriter
```

If you want pre-releases, add the `ppa:vkbo/novelwriter-pre` repository instead.

3.2.2 Debian and Mint

Since this is a pure Python package, the Launchpad PPA can in principle also be used on Debian or Mint. However, the above command will fail to add the signing key.

Instead, run the following commands to add the repository and key:

```
sudo gpg --no-default-keyring --keyring /usr/share/keyrings/novelwriter-ppa-
↪keyring.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪F19F1FCE50043114
echo "deb [signed-by=/usr/share/keyrings/novelwriter-ppa-keyring.gpg] http://
↪ppa.launchpad.net/vkbo/novelwriter/ubuntu focal main" | sudo tee /etc/apt/
↪sources.list.d/novelwriter.list
```

Then run the update and install commands as for Ubuntu:

```
sudo apt update
sudo apt install novelwriter
```

Note: Please use the Ubuntu 20.04 (focal) packages for Debian. The newer Ubuntu packages use a different compression algorithm that Debian doesn't currently support.

3.2.3 AppImage Releases

For other Linux distros than the ones mentioned above, the primary option is AppImage. These are completely standalone images for the app that include the necessary environment to run novelWriter. They can of course be run on any Linux distro, if you prefer this to native packages.

3.3 Minimal Package Install

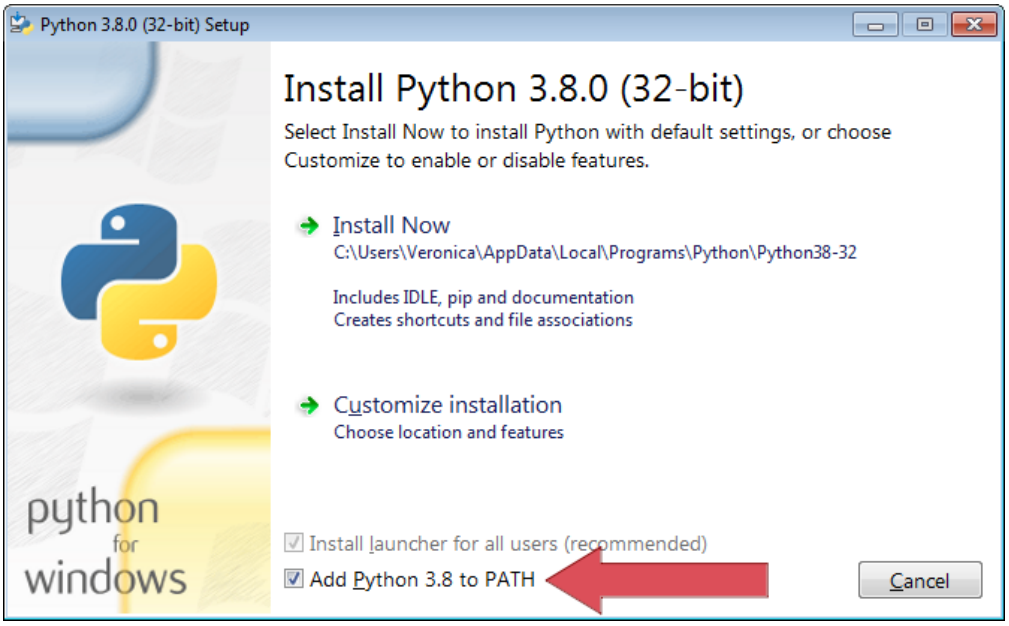
On the main website and on the Releases page on GitHub you will also find "Minimal Package" install files for Windows, Linux and macOS. These are zip files of just the files you need to run novelWriter on that specific platform.

These zip files don't include any dependencies, so you must install them separately.

3.3.1 Minimal Package on Windows

First, make sure you have Python installed on your system. If you don't, you can download it from python.org. Python 3.7 or higher is required, but it is recommended that you install the latest version.

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



When Python is installed, extract the novelWriter zip file and move the extracted folder to a suitable location. You should probably not keep it on your desktop or in your downloads folder where it may be accidentally deleted. Instead, move and rename it to for instance `C:\novelWriter`.

After you've got the folder where you want it, open it and double-click the file named `windows_install.bat`. This will open a command line window and run the setup script to install dependencies, and add desktop and start menu icons.

Running `windows_uninstall.bat` will reverse the process if you wish to uninstall. After that, you can just delete the novelWriter folder.

3.3.2 Minimal Package on Linux

On Linux you need to install the following packages on Debian-based distros, including Ubuntu and Linux Mint:

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

On Fedora, you need the following packages:

```
sudo dnf install python3-qt5 python3-lxml python3-enchant
```

A standard desktop launcher can be installed via the main setup script. It will create the needed desktop file and add it to the Applications menu. The necessary icons will also be installed, and a file association with `.nwx` project files added.

To set this up, run the following from inside the extracted novelWriter folder:

```
python3 setup.py xdg-install
```

This installs icons for the current user. Run with `sudo` to install system-wide.

To uninstall the icons, run:

```
python3 setup.py xdg-uninstall
```

3.3.3 Minimal Package on macOS

These instructions assume you're using brew, and have Python and pip set up. If not, see the brew docs for help. The main requirements for novelWriter 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.

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

3.4 Install from PyPi

novelWriter is also available on the Python Package Index, or PyPi. This install method works on all supported operating systems.

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 in this documentation on how to get the Python environment set up.

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.

Note: On systems with both Python 2 and 3, you may have to replace the `pip` command with `pip3`.

CHAPTER

FOUR

RUNNING FROM SOURCE

This section describes various ways of running novelWriter directly from the source code, and how to build the various components like the translation files and documentation.

Note: The text below assumes the command `python` corresponds to a Python 3 executable. Python 2 is now deprecated, but many systems still have both Python 2 and 3. For such systems, the command `python3` may be needed instead. Likewise, `pip` may need to be replaced with `pip3`.

4.1 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 spell checking (optional).

PyQt/Qt should be at least 5.10, but ideally 5.13 or higher for all features to work. For instance, searching using regular expressions with full Unicode support requires 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.

If you want spell checking, you must install the `PyEnchant` package. 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
```

4.2 Install from Source

You can download the latest version of novelWriter from the source repository on GitHub and run the setup manually. It is equivalent to what the `pip install` command does, and it installs novelWriter to the default location for Python packages.

This step requires that you have `setuptools` installed on your system. If you don't have it installed, it can usually be installed from your distro's repository. For Debian and Ubuntu this is achieved with:

```
sudo apt install python3-setuptools
```

The package is also available from PyPi:

```
pip install --user setuptools
```

With `setuptools` in place, novelWriter can be installed to the user space with:

```
python setup.py install --user
```

Tip: The main setup script has a number of options that may be useful to you. You can list them by running `python setup.py help`.

4.3 Building the Translation Files

If you installed novelWriter from a package, the translation files should be pre-built and included. If you're running novelWriter from the source code, you will need to generate the files yourself. The files you need will be written to the `novelwriter/assets/i18n` folder, and will have the `.qm` file extension.

You can build the `.qm` files with:

```
python setup.py qtlrelease
```

This requires that the Qt Linguist tool is installed on your system. On Ubuntu and Debian, the needed package is called `qttools5-dev-tools`.

Note: If you want to improve novelWriter with translation files for another language, or update an existing translation, instructions for how to contribute can be found in the `README.md` file in the `i18n` folder of the source code.

4.4 Building the Example Project

In order to be able to create new projects from example files, you need a `sample.zip` file in the `assets` folder of the source. This file can be built from setup script by running:

```
python setup.py sample
```

4.5 Building the Documentation

A local copy of this documentation can be generated as HTML. This requires the following Python packages from PyPi:

```
pip install furo sphinx
```

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

```
make -C docs 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.

You can also build a PDF manual from the documentation using the setup script:

```
python setup.py manual
```

This will build the documentation as a PDF using LaTeX. The file will then be copied into the assets folder and made available in the *Help* menu in novelWriter. The Sphinx build system has a few extra dependencies when building the PDF. Please check the Sphinx Docs for more details.

CHAPTER

FIVE

CUSTOMISATIONS

There are a few ways you can customise novelWriter youself. Currently, you can add new GUI themes, your own syntax themes, and install additional dictionaries.

5.1 Spell Check Dictionaries

novelWriter uses Enchant as the spell checking tool. Depending on you operating system, it may or may not load installed spell check dictionaries.

Linux
: On Linux, you generally only have to install hunspell or aspell dictionaries on your system like you do for other applications. See your distro's documentation for how to do this. These dictionaries should then show up as available spell check languages in novelWriter.

Windows
: For Windows, English is included with the installation. For other languages you have to download and add dictionaries yourself. You can find the various dictionaries on the Free Desktop website. You should find a folder for your language, if it is available at all, and download the files ending with `.aff` and `.dic`. These files must then be copied to the following location:

  `C:\Users\<USER>\AppData\Local\enchant\hunspell`

  This assumes your user profile is stored at `C:\Users\<USER>`. The last one or two folders may not exist, so you may need to create them.

5.2 Syntax and GUI Themes

Adding your own GUI and syntax themes is relatively easy. The themes are defined by simple plain text config files with meta data and colour settings.

In order to make your own versions, first copy one of the existing files to your local computer and modify it as you like.

- The existing syntax themes are stored in novelwriter/assets/syntax.
- The existing GUI themes are stored in novelwriter/assets/themes.

Remember to also change the name of your theme by modifying the `name` setting at the top of the file, otherwise you may not be able to distinguish them in *Preferences*.

For novelWriter to be able to locate the custom theme files, you must copy them to the *Application Data* location in your home or user area. There should be a folder there named `syntax` for syntax themes and just `themes` for GUI themes. These folders are created the first time you start novelWriter.

Once the files are copied there, they should show up in *Preferences* with the label you set as `name` inside the file.

Note: In novelWriter 2.0 the `icontheme` value was added to GUI themes. Make sure you set this value in existing custom themes. Otherwise it defaults to `typicons_light`, which may not match your theme colour scheme.

5.2.1 Gustom GUI Theme

A GUI theme conf file consists of the follwing settings:

```
[Main]
name        = My Custom Theme
description = A description of my custom theme
author      = Jane Doe
credit      = John Doe
url         = https://example.com
license     = CC BY-SA 4.0
licenseurl  = https://creativecommons.org/licenses/by-sa/4.0/
icontheme   = typicons_light

[Palette]
window          = 100, 100, 100
windowtext      = 100, 100, 100
base            = 100, 100, 100
alternatebase   = 100, 100, 100
text            = 100, 100, 100
tooltipbase     = 100, 100, 100
tooltiptext     = 100, 100, 100
button          = 100, 100, 100
buttontext      = 100, 100, 100
brighttext      = 100, 100, 100
highlight       = 100, 100, 100
highlightedtext = 100, 100, 100
link            = 100, 100, 100
linkvisited     = 100, 100, 100

[GUI]
statusnone      = 100, 100, 100
statussaved     = 100, 100, 100
statusunsaved   = 100, 100, 100
```

In the Main section you must at least define the `name` and `icontheme` settings. The `icontheme` settings should correspond to one of the internal icon themes. Either `typicons_light` or `typicons_dark`.

The Palette values correspond the the Qt enum values for QPalette::ColorRole, see the Qt documentation

for more details. The colour values are RGB numbers on the format `r, g, b` where each is an integer from to 255. Omitted values are not loaded and will use default values.

5.2.2 Custom Syntax Theme

A syntax theme conf file consists of the follwing settings:

```
[Main]
name        = My Syntax Theme
author      = Jane Doe
credit      = John Doe
url         = https://example.com
license     = CC BY-SA 4.0
licenseurl = https://creativecommons.org/licenses/by-sa/4.0/

[Syntax]
background     = 255, 255, 255
text           =   0,   0,   0
link           =   0,   0,   0
headertext     =   0,   0,   0
headertag      =   0,   0,   0
emphasis       =   0,   0,   0
straightquotes =   0,   0,   0
doublequotes   =   0,   0,   0
singlequotes   =   0,   0,   0
hidden         =   0,   0,   0
keyword        =   0,   0,   0
value          =   0,   0,   0
spellcheckline =   0,   0,   0
errorline      =   0,   0,   0
replacetag     =   0,   0,   0
modifier       =   0,   0,   0
```

In the Main section, you must define at least the `name` setting. The Syntax colour values are RGB numbers on the format `r, g, b` where each is an integer from to 255. Omitted values are set to black, except `background` which defaults to white,

CHAPTER

SIX

HOW IT WORKS

The main features of novelWriter are listed in the *Key Features* section. Here, we go into some more details on how they are implemented. This is intended as an overview. Later on in this documentation, these features will be covered in even more detail.

6.1 GUI Layout and Design

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.

The main window does not have an editor toolbar like many other applications do. This reduces clutter, and since the documents are formatted with style tags, is more or less redundant. However, most formatting features supported are available through convenient keyboard shortcuts. They are also available in the main menu so you don't have to look up formatting codes every time you need them. For reference, a list of all shortcuts can be found in the *Keyboard Shortcuts* section.

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, emphasis, text alignment, and a few other simple features.

On the left edge of the main window, you will find a sidebar. This bar has buttons for the standard views you can switch between, a quick link to the *Build Novel Project* tool, and a set of project-related tools as well as quick access to settings at the bottom.

6.1.1 Project Tree View

When in *Project Tree View* mode, the main work area of 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. An optional third panel on the right is a document viewer which can view any document in your project independently of what is open in the document editor. This panel is not intended as a preview window, although you can use it for this if you wish as it will apply the formatting tags you have specified. The main purpose of the viewer is for viewing your notes next to your editor while you're writing.

The editor also has a *Focus Mode* you can toggle either from the menu, from the icon in the editor header, or by pressing `F8`. When *Focus Mode* is enabled, all the user interface elements other than the document editor itself are hidden away.

6.1.2 Novel Tree View

When in *Novel Tree View* mode, the project tree is replaces by an overview of your novel structure. Instead of showing individual documents, the tree now shows all headings of your novel text. This includes multiple headings within the same document.

Each heading is indented according to the heading level. You can open and edit your novel documents from this view as well. All headings contained in the currently open document should be highlighted in the view to indicate which ones belong together.

If you have multiple Novel root folders, the header of the novel view becomes a droopdown box. You can then switch between them by clicking the "Outline of ..." text. You can also click the novel icon button next to it.

Generally, the novel view should update when you make changes to the novel structure, including edits of the current document in the editor. The information is only updated when the automatic save of the document is initiated though. You can adjust the aut-save interval in *Preferences*. You can also regenerate the whole novel view by pressing the refresh button at the top.

It is possible to show an optional third column in the novel view, The settings are available from the menu button ath the top.

If you click the arrow icon to the right of each item, a tooltip will pop up showing you all the meta data collected for that heading entry.

6.1.3 Novel Outline View

When in *Novel Outline View* mode, the tree, editor and viewer will be replaced by a large table that shows the entire novel structure with all the tags and references listed. Pretty much all collected meta data is available here in different columns.

You can select which novel root folder to display from the dropdown box, and you can select which columns to show or hide from the menu button. You can also rearrange the columns by drag and drop. The app will remember you column order and size between sessions, and for each individual project.

6.1.4 Colour Themes

The default colour theme of the user interface is the default theme from the Qt library. By default, novelWriter is loaded with the Fusion style setting. (You can override this with the `--style=` setting when starting novelWriter.)

There is a standard dark theme provided as well, which is similar to the default Qt theme. Some other light and dark colour themes are also provided. You can select which one you prefer from in *Preferences* .

A number of syntax highlighting themes are also available in *Preferences*. These are separate settings because there are a lot more options for syntax highlighting.

Note: If you switch to dark mode on the GUI, you should also switch syntax highlighting theme to match, otherwise icons may be hard to see in the editor and viewer.

6.2 Project Layout

This is a brief introduction to how you structure your writing projects. All of this will be covered in more detail later.

The main point of novelWriter is that you are free to organise your project documents as you wish into subfolders or subdocuments, and split the text between these 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.

The project tree will select an icon for the document based on the first heading in it.

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.

Note: You can add documents as child items of other documents if you wish. This is often more useful than adding folders, since you anyway may want to have the chapter heading in a separate document from your individual scene documents.

6.3 Building the Manuscript

The project can at any time be assembled into a range of different formats through the *Build Novel Project* tool. Natively, novelWriter supports Open Document, HTML5, and various flavours of Markdown.

The HTML5 format is suitable for conversion by a number of other tools like Pandoc, or for importing into word processors if the Open Document format isn't suitable. In addition, printing and printing to PDF is also possible.

You can also 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. The JSON file can be populated either with HTML formatted text, or with the raw text as typed into the novel documents. See *Additional Formats* for more details.

A number of filter options can be applied to the *Build Novel Project* tool, allowing you to make a draft manuscript, a reference document of notes, an outline based on chapter and scene titles with a synopsis each, and so on. See *Building the Manuscript* for more details on build features and formats.

6.4 Project Storage

The files of a novelWriter project are stored in a dedicated project folder. The project structure is kept in a file at the root of this folder called `nwProject.nwx`. All the document files and associated meta data is stored in the other folders below the project folder. For more technical details about what all the files mean and how they're organised, see the *How Data is Stored* section.

This way of storing data was chosen for several reasons. Firstly, all the text you add to your project is saved directly to your project folder in separate files. Only the project structure and the text you are currently editing is stored in memory at any given time. Secondly, having multiple small files means it is very easy to sync them between computers with standard file synchronisation tools. Thirdly, if you use version control software to track the changes to your project, the file formats used for the files are well suited. Also the JSON documents have line breaks and indents, which makes it easier to track them with version control software.

Note: Since novelWriter has to keep track of a bunch of files and folders when a project is open, it may not run well on some virtual file systems. A file or folder must be accessible with exactly the path it was saved or created with. An example where this is not the case is the way Google Drive is mapped on Linux Gnome desktops using gvfs/gio.

Caution: You should not add additional files to the project folder yourself. Nor should you manually edit files within it as a general rule. If you really must manually edit the text files, e.g. with some automated task you want to perform, you need to rebuild the index when you open the project again.

Editing text files in the `content` folder is less risky as they are just plain text. Editing the main project XML file, however, may make the project file unreadable and you may crash novelWriter and lose project structure information and project settings.

CHAPTER

SEVEN

THE USER INTERFACE

This sections covers in more detail what all the information on the user interface is for, and how you can organise your project, and how you use the editor, viewer and outline panel.

7.1 The Project Tree

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

Column 1
: The first column shows the icon and label of each folder, document, or note in your project. The label is not the same as the title you set inside the document. However, the document's label will appear in the header above the document text itself so you know where in the project an open document belongs. The icon is selected based on the type of item, and for novel documents, the level of the first header in the document text.

Column 2
: The second column shows the word count of the document, or the sum of words of the child items for folders and documents with subdocuments. If the counts seem incorrect, they can be updated by rebuilding the project index from the *Tools* menu, or by pressing `F9`.

Column 3
: The third column indicates whether the document is considered active or inactive in the project. You can use this flag to indicate that a document is still in the project, but should not be considered an active part of it. When you run the *Build Novel Project* tool, you can filter based on this flag. You can change this value from the context menu.

Column 4
: The fourth column shows the user-defined status or importance labels you've assigned to each project item. See *Document Importance and Status* for more details. You can change these labels from the context menu.

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.

At the top of the tree, you will find a set of buttons.

- The first button is a quick links button that will show you a dropdown menu of all the root folders in your project. Selecting one will move to that position in the tree. You can also activate this menu by pressing `CtrlL`.
- The next two buttons can be used to move items up and down in the project tree. This is the only way to move root folders.

- The next button opens a dropdown menu for adding new items to the tree. This includes root folders. You can also activate this dropdown menu by pressing `CtrlN`.
- The last button is a menu of further actions on the entire project tree.

Tip: Under the *Transform* submenu in the context menu of an item, you will find several options on how to change a document or folder. This includes changing between document and note, splitting them into multiple documents, or merging child items into a single document.

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.

7.1.1 The Novel Tree

An alternative way to view the project structure is the novel tree. You can switch to this view by selecting the *Novel Tree View* button in the sidebar. This view is a simplified version of the view in the *Outline*. It is convenient when you want to browse the structure of the story itself rather than the document files.

Note: You cannot reorganise the entries in the novel tree, or add any new documents, as that would imply restructuring the content of the document files. Any editing must be done in the project tree. However, you can add new headings to existing documents, or change references, which will be updated in this view.

7.1.2 Document Importance and Status

Each document or folder in your project can have either a "Status" or "Importance" flag set. These are flags that you control and define yourself. The app doesn't do anything with them at all. To modify the labels, go to their respective tabs in *Project Settings*.

The "Status" flag is intended to tag a Novel document as for instance a draft or as completed, and the "Importance" flag is intended to tag character notes, or other notes, as for instance a main, major or minor character.

Whether a document uses a "Status" or "Importance" flag depends on which root folder it lives in. If it's in the *Novel* folder, it uses the "Status" flag, otherwise it uses an "Importance" flag. Some folders, like *Trash* and *Archive* allow both.

7.1.3 Project Tree Drag & Drop

The project tree allows drag & drop to a certain extent to allow you to reorder your documents and folders. Moving a document in the project tree will affect the text's position when you assemble your manuscript in the build tool.

Drag & drop has only limited support for moving documents. In general, bulk actions are not allowed. This is deliberate to avoid accidentally messing up your project. If you make a mistake, the last move action can be undone by pressing `CtrlShiftZ`.

Documents and their folders can be rearranged freely within their root folders. If you move a Novel documents out of a Novel folder, it will be converted to a project note. Notes can be moved freely between all root folders, but keep in mind that if you move a note into a *Novel* root folder, its "Importance" setting will be switched with a "Status" setting. See *Document Importance and Status*. The old value will not be overwritten though, and should be restored if you move it back at some point.

Root folders in the project tree cannot be dragged & dropped at all. If you want to reorder them, you can move them up or down with respect to eachother from the arrow buttons at the top of the project tree, or by pressing `CtrlShift` and the `Up` or `Down` key.

7.2 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 Markdown-like syntax for some features, and a novelWriter-specific syntax for others. The syntax format is described in the *Formatting Your Text* section.

The editor has a maximise button (toggles the *Focus Mode*) and a close button in the top–right corner. On the top–left side you will find an edit button that opens the *Item Label* dialog for the currently open document, and a search button to open the search dialog.

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 or context 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 as 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 *Preferences*.

Tip: Clicking on the document title bar will select the document in the project tree and reveal its location, 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 backward and forward navigation buttons, these can be used as well. They work just like the backward and forward features in a browser.

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 the one you're currently viewing, 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`.

7.2.1 Search & Replace

The document editor has a search and replace tool that can be activated with `CtrlF` for search mode or `CtrlH` for search and replace mode.

Pressing `Return` while in the search box will search for the next occurrence of the word, and `ShiftReturn` for the previous. Pressing `Return` in the replace box, will replace the highlighted text and move to the next result.

There are a number of settings for the search tool available as toggle switches above the search box. They allow you to search for, in order: matched case only, whole word results only, search using regular expressions, loop search when reaching the end of the document, and move to the next document when reaching the end. There is also a switch that will try to match the case of the word when the replacement is made. That is, it will try to keep the word upper, lower, or capitalised to match the word being replaced.

The regular expression search is somewhat dependant on which version of Qt your system has. If you have Qt 5.13 or higher, there is better support for unicode symbols in the search.

7.3 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 this feature entirely if you wish.

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*. You may also be using a Compose Key setup, which means you may not need the auto-replace feature.

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, like English does.

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.

7.4 Project Outline View

The project's Outline view is available as another view option from the views bar. 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.

You can select which novel folder to display from the dropdown menu. You can optionally also choose to show a combination of all novel folders.

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 from the menu button in the toolbar. The order of the columns can also be rearranged by dragging them to a different position. You column settings are saved between sessions on a per-project basis.

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 contents change, you can always rebuild it manually by pressing `F9` if something isn't right.

The outline view itself can be regenerated by pressing the refresh button. By default, the content is refreshed each time you switch to this view.

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

CHAPTER

EIGHT

FORMATTING YOUR TEXT

The novelWriter text editor is a plain text editor that uses formatting codes for setting meta data values and allowing for some text formatting. The syntax is based on Markdown, but novelWriter is *not* a Markdown editor. It supports basic formatting like emphasis (italic), strong importance (bold) and strikethrough text, as well as four levels of headings.

In addition to formatting codes, novelWriter allows for comments, a synopsis tag, and a set of keyword and value sets used for tags and references. There are also some codes that apply two whole paragraphs. See *Text Paragraphs* below for more details.

8.1 Syntax Highlighting

The editor has a syntax highlighter feature that is meant to help you know when you've used the formtatting tags or other features correctly. It will change the colour and font size of your headings, change the text colour of emphasised text, and it can also show you where you have dialogue in your text.

When you use the commands to set tags and references, these also change colour. Correct commands have a dedicated colour, and the references themselves will get a colour if they are valid. Invalid references will get a squiggly error line underneath. The same applies to duplicate tags.

There are a number of syntax highlighter colour themes available, both for light and dark GUIs. You can select them for *Preferences*.

8.2 Headings

Four levels of headings are allowed. For project notes, they are free to be used as you see fit. That is, novelWriter doesn't assign the different headings any particular meaning. However, for novel documents they indicate the structural level of the novel and must be used correctly to produce the intended result. See *Importance of Headings* for more details.

`# Title Text`
: Heading level one. For novel documents, the header level indicates the start of a new partition.

`## Title Text`
: Heading level two. For novel documents, the header level indicates the start of a new chapter. Chapter numbers can be inserted automatically when building the manuscript.

`### Title Text`
: Heading level three. For novel documents, the header level indicates the start of a new scene. Scene numbers or scene separators can be inserted automatically when building the manuscript, so you can use the title field as a working title for your scenes if you wish.

`#### Title Text`
: Heading level four. For novel documents, the header level indicates the start of a new section. Section titles can be replaced by separators or removed completely when building the manuscript.

For headers level one and two, adding a `!` modifies the behaviour of the heading:

`#! Title Text`
: This tells the build tool that the level one heading is intended to be used for the novel's main title, like for instance on the front page. When building the manuscript, this will use a different styling and will exclude the title from for instance a Table of Contents in Libre Office.

`##! Title Text`
: This tells the build tool to not assign a chapter number to this chapter title if automatic chapter numbers are being used. Such titles are useful for a prologue for instance. See *Unnumbered Chapter Headings* for more details.

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

8.3 Text Paragraphs

A text paragraph is indicated by a blank line. That is, you need two line breaks to separate two fragments of text into two paragraphs. Single line breaks are treated as line breaks within a paragraph.

In addition, the editor supports a few additional types of whitespaces:

- 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*.

Non-breaking spaces are highlighted by the syntax highlighter with an alternate coloured background, depending on the selected theme.

Tip: Non-breaking spaces are for instance the correct type of space to separate a number from its unit. Generally, non-breaking spaces are used to prevent line wrapping algorithms from adding line breaks where they shouldn't.

8.4 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.

8.5 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 building the manuscript or viewing the document.

If the first word of a comment is `Synopsis:` (with the colon included), 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 renderred by default (this can be overridden), seen in the document 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 building 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.

8.6 Tags and References

The document editor supports a set of keywords used for setting tags, and making references between documents. The tag can be set once per section defined by a heading. Setting it multiple times under the same heading will just override the previous setting. References can be set anywhere within a section, and are collected according to their category.

`@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 *Note References* section. They can also be inserted at the cursor position in the editor via the *Insert* menu.

8.7 Paragraph Alignment and Indentation

All documents have the text by default aligned to the left or justified, depending on your Preferences.

You can override the default text alignment on individual paragraphs by specifying alignment tags. These tags are double angle brackets. Either >> or <<. You put them either before or after the paragraph, and they will "push" the text towards the edge the brackets point towards. This should be fairly intuitive.

Indentation uses a similar syntax. But here you use a single > or < to push the text away from the edge.

Examples:

Table 1: Text Alignment and Indentation

Syntax	Description
`>> Right aligned text`	The text paragraph is right-aligned.
`Left aligned text <<`	The text paragraph is left-aligned.
`>> Centred text <<`	The text paragraph is centred.
`> Left indented text`	The text has an increased left margin.
`Right indented text <`	The text has an increased right margin.
`> Left/right indented text <`	The text has an both margins increased.

Note: The text editor will not show the alignment and indentation live. But the viewer will show them when you open the document there. It will of course also be reflected in the document generated from the build tool as long as the format supports paragraph alignment.

8.8 Vertical Space and Page Breaks

Adding more than one line break between paragraphs will *not* increase the space between those paragraphs when building the project. To add additional space between paragraphs, add the text `[VSPACE]` on a line of its own, and the build tool will insert a blank paragraph in its place.

If you need multiple blank paragraphs just add a colon and a number to the above code. For instance, writing `[VSPACE:3]` will insert three blank paragraphs.

Normally, the build tool will insert a page break before all headers of level one and for all headers of level two for novel documents, i.e. chapters, but not for project notes.

If you need to add a page break somewhere else, put the text `[NEW PAGE]` on a line by itself before the text you wish to start on a new page.

Page breaks are automatically added to partition, chapter and unnumbered chapter headers of novel documents. If you want such breaks for scenes and sections, you must add them manually.

Note: The page break code is applied to the text that follows it. It adds a "page break before" mark to the text when exporting to HTML or Open Document. This means that a `[NEW PAGE]` which has no text following it, it will not result in a page break.

CHAPTER

NINE

KEYBOARD SHORTCUTS

Most features in novelWriter are available as keyboard shortcuts. This is a reference list if these shortcuts. Most of them are also listed in the application.

9.1 Main Shortcuts

The main shorcuts are as follows:

Table 1: Keyboard Shortcuts

Shortcut	Description
F1	Open the online user manual.
F2	If in the project tree, edit an item's label.
F3	Find next occurrence of search word in current document.
F5	Open the *Build Novel Project* dialog.
F6	Open the *Writing Statistics* dialog.
F7	Re-run spell checker.
F8	Toggle *Focus Mode*.
F9	Re-build the project index.
F11	Toggle full screen mode.
Return	If in the project tree, open a document for editing.
Alt1	Switch focus to the project tree. On Windows, use Ctrl Alt 1.
Alt2	Switch focus to document editor. On Windows, use Ctrl Alt 2.
Alt3	Switch focus to document viewer. On Windows, use Ctrl Alt 3.
Alt4	Switch focus to outline view. On Windows, use Ctrl Alt 4.
AltLeft	Move backward in the view history of the document viewer.
AltRight	Move forward in the view history of the document viewer.
Ctrl.	Open the context menu in the project tree or the document editor.
Ctrl,	Open the *Preferences* dialog.
Ctrl/	Toggle block format as comment.
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.
Ctrl5	Change block alignment to left-aligned.
Ctrl6	Change block alignment to centred.
Ctrl7	Change block alignment to right-aligned.

continues on next page

Table 1 – continued from previous page

Shortcut	Description
Ctrl8	Add a left margin to the block.
Ctrl9	Add a right margin to the block.
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	Strikethrough selected text, or word under cursor.
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.
CtrlH	Open the search and replace tool 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*.
CtrlL	Open the *Quick Links* menu in the Project Tree.
CtrlN	Open the Create New Item menu in the Project Tree.
CtrlO	Open selected document.
CtrlQ	Exit novelWriter.
CtrlR	If in the project tree, open a document for viewing. If in the editor, 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.
CtrlUp	Move item one step up in the project tree.
CtrlDown	Move item one step down in the project tree.
CtrlDel	Delete next word in editor.
CtrlBackspace	Delete previous word in editor.
Ctrl'	Wrap selected text, or word under cursor, in single quotes.
Ctrl"	Wrap selected text, or word under cursor, in double quotes.
CtrlReturn	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.
CtrlShiftA	Select all text in current paragraph.
CtrlShiftG	Find previous occurrence of search word in current document.
CtrlShiftI	Import text to the current document from a text file.
CtrlShiftO	Open a project.
CtrlShiftR	Close the document viewer.
CtrlShiftS	Save the current project.
CtrlShiftW	Close the current project.
CtrlShiftZ	Undo move of project tree item.
CtrlShiftDel	If in the project tree, move an item to Trash.
ShiftF1	Open the local user manual (PDF) if it is available.
ShiftF3	Find previous occurrence of search word in current document.

continues on next page

Table 1 – continued from previous page

Shortcut	Description
ShiftF6	Open the *Project Details* dialog.

Note: On macOS, replace Ctrl with Cmd.

9.2 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, Ctrl_	Insert a horizontal bar (quotation dash).
CtrlK, ~	Insert a figure dash (same width as a number).
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, .	Insert an ellipsis.
CtrlK, Ctrl'	Insert a prime.
CtrlK, Ctrl"	Insert a double prime.
CtrlK, Space	Insert a non-breaking space.
CtrlK, ShiftSpace	Insert a thin space.
CtrlK, CtrlSpace	Insert a thin non-breaking space.
CtrlK, *	Insert a list bullet.
CtrlK, Ctrl-	Insert a hyphen bullet (alternative bullet).
CtrlK, Ctrl*	Insert a flower mark (alternative bullet).
CtrlK, %	Insert a per mille symbol.
CtrlK, CtrlO	Insert a degree symbol.
CtrlK, CtrlX	Insert a times sign.
CtrlK, CtrlD	Insert a division sign.
CtrlK, G	Insert a @tag keyword.
CtrlK, V	Insert a @pov keyword.
CtrlK, F	Insert a @focus keyword.
CtrlK, C	Insert a @char keyword.
CtrlK, P	Insert a @plot keyword.
CtrlK, S	Insert a synopsis comment.
CtrlK, T	Insert a @time keyword.
CtrlK, L	Insert a @location keyword.
CtrlK, O	Insert an @object keyword.

continues on next page

Table 2 – continued from previous page

Shortcut	Description
`CtrlK`, `E`	Insert an `@entity` keyword.
`CtrlK`, `X`	Insert a `@custom` keyword.

CHAPTER

TEN

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.

10.1 Special Notes on Symbols

This section contains additional notes on the available special symbols.

10.1.1 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.

In addition, "Figure Dash" is available. The Figure Dash is a dash that has the same width as the numbers of the same font, for most fonts. It helps to align numbers nicely in columns when you need to use a dash in them.

10.1.2 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 regular straight double quotes and 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.

10.1.3 Single and Double Prime

Both single and double prime symbols are available in the *Insert* menu. These symbols are the correct symbols to use for unit symbols for feet, inches, minutes and seconds. The usage of these is described in more detail on the Wikipedia Prime page. They look very similar to single and double straight quotes, and may be renderred similarly by the font, but they have different codes. Using these correctly will also prevent the auto-replace and dialogue highlighting features misunderstanding their meaning in the text.

10.1.4 Modifier Letter Apostrophe

The auto-replace feature will consider any right-facing single straight quote as a quote symbol, even if it is 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.

10.1.5 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.

CHAPTER

ELEVEN

PROJECT FORMAT CHANGES

Most of the changes to the file formats over the history of novelWriter have no impact on the user-side of things. The project files are generally updated automatically. However, some of the changes require minor actions from the user.

The key changes in the formats are listed below, as well as the user actions required, where applicable.

A full project file format specification is available in the online documentation.

> **Caution:** When you update a project from one format version to the next, the project can no longer be opened by a version of novelWriter prior to the version where the new file format was introduced. You will get a notification about any updates to your project file format and will have the option to decline the upgrade.

11.1 Format 1.5 Changes

This project format was introduced in novelWriter version 2.0 RC 2.

This is a modification of the 1.4 format. It makes the XML more consistent in that meta data have been moved to their respective section nodes as attributes, and key/value settings now have a consistent format. Logical flags are saved as yes/no instead of Python True/False, and the main heading of the document is now saved to the item rather than in the index. The conversion is done automatically the first time a project is loaded. No user action is required.

11.2 Format 1.4 Changes

This project format was introduced in novelWriter version 2.0 RC 1. Since this was a release candidate, it is unlikely that your project uses it, but it may be the case if you've installed a pre-release.

This format changes the way project items (folders, documents and notes) are stored. It is a more compact format that is simpler and faster to parse, and easier to extend. The conversion is done automatically the first time a project is loaded. No user action is required.

11.3 Format 1.3 Changes

This project format was introduced in novelWriter version 1.5.

With this format, the number of document layouts was reduced from 8 to 2. The conversion of document layouts is performed automatically when the project is opened.

Due to the reduction of layouts, some features that were previously controlled by these layouts will be lost. These features are instead now controlled by syntax codes, so to recover these features, some minor modification must be made to select documents by the user.

The manual changes the user must make should be very few as they apply to document layouts that should be used only a few places in any given project. These are as follows:

Title Pages

- The formatting of the level one title on the title page must be changed from `# Title Text` to `#! Title Text` in order to retain the previous functionality. See *Headings*.
- Any text that was previously centred on the page must be manually centred using the new text alignment feature. See *Paragraph Alignment and Indentation*.

Unnumbered Chapters

- Since the specific layout for unnumbered chapters has been dropped, such chapters must all use the `##! Chapter Name` formatting code instead of `## Chapter Name`. This also includes chapters marked by an asterisk: `## *Chapter Name`, as this feature has also been dropped. See *Headings*.

Plain Pages

- The layout named "Plain Page" has also been removed. The only feature of this layout was that it ensured that the content always started on a fresh page. In the new format, fresh pages can be set anywhere in the text with the `[NEW PAGE]` code. See *Vertical Space and Page Breaks*.

11.4 Format 1.2 Changes

This project format was introduced in novelWriter version 0.10.

With this format, the way auto-replace entries were stored in the main project XML file changed.

11.5 Format 1.1 Changes

This project format was introduced in novelWriter version 0.7.

With this format, the `content` folder was introduced in the project storage. Previously, all novelWriter documents were saved in a series of folders numbered from `data_0` to `data_f`.

It also reduces the number of meta data and cache files. These files are automatically deleted if an old project is opened. This was also when the Table of Contents file was introduced.

11.6 Format 1.0 Changes

This is the original file format and project structure. It was in use up to version 0.6.3.

CHAPTER

TWELVE

NOVEL PROJECTS

New projects 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 novelWriter project requires a dedicated folder for storing its files on the local file system. See *How Data is Stored* for further details on how files are organised.

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 or by clicking the *Remove* button.

Project-specific settings are available in *Project Settings* in the *Project* menu. See further details below in the *Project Settings* section. Details about the project, including word counts, and a table of contents with word and page counts, is available through the *Project Details* dialog.

12.1 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 novel documents go into a root folder of type *Novel*. Project notes 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 the project tree tool bar.

Each root folder has one or more reference keyword associated with it that is used to reference them from other documents and notes. The intended usage of each type of root folder is listed below. However, aside from the *Novel* folder, no restrictions are applied by the application on what you put in them. You can use them however you want.

Tip: You can make multiple root folders of each kind if you wish to split up your notes.

Note: It is not the notes themselves that are referenced by the listed reference keywords, but tags set within the notes. See the examples below and in *Note References*.

Novel
: This is the root folder of all text that goes into the final novel or novels. This class of documents have other rules and features than the project notes. 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 brief 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 you want 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, or focusing on a particular character's storyline. Tags in this folder can be referenced using the `@pov` keyword for point-of-view characters, `@focus` for a focus character, or the `@char` keyword for any 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 *Note References*.

Note: You can rename root folders to whatever you want. However, this doesn't change the reference keyword.

Example of a character note:

```
# Jane Doe

@tag: Jane

Some information about the character Jane Doe.
```

Example of a novel scene referencing the above character:

```
### Chapter 1, Scene 1

@pov: Jane

When Jane woke up that morning ...
```

12.1.1 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.

A document or a folder can be deleted from the *Project* menu, or by pressing `CtrlShiftDel`. Root folders can only be deleted when they are empty.

12.1.2 Archived Documents

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 *Archive* root folder.

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.

12.1.3 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 give 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.

12.1.4 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 reopening 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 project inconsistencies the next time you open the project.

12.1.5 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 add child documents to other documents. This is particularly useful when you create chapters and scenes. If you add separate scene documents, you should also add separate chapter documents, even if they only contain a chapter heading. You can then add scene documents as child items to the chapters.

12.2 Project Documents

New documents can be created from the tool bar in the Project Tree, or by pressing `CtrlN`. This will open the create new item menu and let you choose between a number of pre-defined documents and folders. You will be prompted for a label for the new item. You can always rename an item by selecting *Rename Item* from the *Project* menu, or by pressing `F2`.

Other settings for project items are available from the context menu that you can activate by right-clicking on an item in the Project Tree. The *Transform* submenu includes options for converting, splitting, or merging items.

12.2.1 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 or novel tree.

The word counts are not updated in real time, but run in the background every few 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`.

12.3 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.

12.3.1 Settings Tab

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

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

The *Novel Title* and *Authors* settings are used when building the manuscript, for some formats.

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.

12.3.2 Status and Importance Tabs

Each document or folder of type *Novel* can be given a *Status*_ label accompanied by a coloured icon, and each document or folder of the remaining types can be given an *Importance* label.

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 story. 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.

See also *Document Importance and Status*.

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

12.3.3 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 manuscript builds.

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 matching 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.

12.4 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 project files in the project folder (files not strictly a part of the project are ignored). The zip archives are stored in a subfolder of the backup path. The subfolder will have the same name as the *Project Name* as defined in *Project Settings*.

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

Note: For the backup to be able to run, the *Project Name* 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.

12.5 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, provided that the session lasted either more than 5 minutes, or that the total word count changed. 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`, or find it on the Sidebar.

The tool will show a list of all your sessions, and a set of filters to apply to the data. 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.

As of version 1.2, the log file also stores how much of the session time was spent idle. The definition of idle here is that the novelWriter main window loses focus, or the user hasn't made any changes to the currently open document in five minutes. The number of minutes can be altered in *Preferences*.

CHAPTER

THIRTEEN

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.

13.1 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 *Formatting Your Text* section for more details about the markdown syntax.

Note: The header levels are not only important when generating the manuscript, they are also used by the indexer when building the outline tree in the Outline as well as the Novel Tree. Each heading also starts a new region where new references and tags can be defined.

The syntax for the four basic header types, and the two special header types, is listed in section *Headings*. The meaning of the four levels for the structure of your novel is as follows:

Header Level 1: Partition
: This header level signifies that the text refers to a top level partition. This is useful when you want to split the manuscript up into books, parts, or acts. These headings are not required. The novel title itself should use the special header level explained in *Headings*.

Header Level 2: Chapter
: This header level 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.

Header Level 3: Scene
: This header level 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 build your manuscript.

Header Level 4: Section
: This header level 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 you can filter them out of the final manuscript just like with scene titles.

Page breaks are automatically added before level 1 and 2 headers when you build your project to a format that supports page breaks, or when you print the document directly from the build tool. If you want page breaks in other places, you have to specify them manually. See *Vertical Space and Page Breaks*.

Tip: There are multiple options of how to process novel titles when building 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 *Building the Manuscript* page for more details.

13.1.1 Novel Title and Front Matter

It is recommended that you add a document at the very top of each Novel root folder with the novel title as the first line. You should modify the level 1 header format code with an `!` in order to render it as a document title that is excluded from any automatic Table of Content in a manuscript build document, like so:

```
#! My Novel
```

The title is by default centred on the page. You can add more text to the page as you wish, like for instance the author's name and details.

If you want an additional page of text after the title page, starting on a fresh page, you can add `[NEW PAGE]` on a line by itself, and continue the text after it. This will insert a page break before the text. See also *Vertical Space and Page Breaks*.

13.1.2 Unnumbered Chapter Headings

If you use the automatic numbering feature for your chapters, but you want to keep some special chapters separate from this, you cam add a `!` to the level 2 header formatting code to tell the build tool to skip these chapters.

```
##! Unnumbered Chapter Title
```

There is a separate formatting feature for such chapters in the *Build Novel Project* tool as well. See the *Building the Manuscript* page for more details. When building a document of a format that supports page breaks, also unnumbered chapters will have a page break added just like for normal chapters.

Note: Previously, you could also disable the automatic numbering of a chapter by adding an * as the first character of the chapter title itself. This feature has been dropped in favour of the current format in order to keep level 1 and 2 headers consistent. Please update your chapter headings if you've used this syntax.

13.2 Note References

Each text partition, indicated by a heading of any level, can contain references to tags set in the project notes of the project. The references are gathered by the indexer and used to generate the Outline View. This section covers how to make references to tags. See *Tags in Notes* for how to define tags the references can point to.

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 a *Character* type root folder.

`@focus`
: The character that has the focus for the current section. This can be used in cases where the focus is not a point-of-view character. The target must be a note tag in a *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 or focus character if those references are used.

`@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 a *Object* type root folder.

`@entity`
: Entities present in the current section. The target must be a note tag in a *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 custom folder are for any other category of notes you may want to use.

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.

Example of a novel document with references to characters and plots:

```
## Chapter 1

@pov: Jane

### Scene 1

@char: John, Sam
@plot: Main

Once upon a time ...
```

13.3 Document Layout

All documents in the project can have a layout format set. Previously, there were multiple layouts available to change how the documents where formatted on export. These have now been reduced to just two layouts: *Novel Document* and *Project Note*.

Novel documents can only live in a *Novel* type root folder. You can also move them to *Archive* and *Trash* of course. Project notes can be added anywhere in the project.

The project tree can distinguish between the different layouts and header levels of the documents using coloured icons, and optionally add emphasis on the label (See the *Preferences*.) For novel documents, the heading level of the first heading is recorded, and indicated by the icon.

Tip: You can always start writing with a coarse setup with one or a few documents, and then later use the split tool to split the documents into separate chapter and scene documents. You can split a document on any of the four header levels.

CHAPTER

FOURTEEN

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 that are written and maintained just like all other text in your project.

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 View 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.

14.1 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 of your choosing. 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 opened in the viewer. See *Note References* for how to reference notes.

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 added to the manuscript, or an outline document. If you want to compile a single document of all your notes, you can do this from the *Build Novel Project* tool.

One note can also reference another note 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, or of course if you just want to highlight that two notes are related.

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.

Example of a project note with two headers, with separate tags, and with references to other notes:

```
# Main Characters

## Jane Doe

@tag: Jane
@location: Earth

Something about Jane ...

## John Doh

@tag: John
@location: Mars

Something about John ...
```

CHAPTER

FIFTEEN

BUILDING THE MANUSCRIPT

You can at any time build a manuscript, an outline of your notes, or any other type of document from the text in your project. All of this is handled by the *Build Novel Project* tool. You can activate it from the sidebar, the *Tools* menu, or by pressing `F5`.

Note: This tool is scheduled to receive a full update in novelWriter 2.1, adding many new features requested by users. See the 2.1 Milestone for an overview.

15.1 Header Formatting

The titles for the five types of headings (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, unless the heading formatting code has the added `!`. In the latter case, the counter is *not* incremented. This is useful for for instance Prologue and Epilogue chapters.

`%chw%`
: Behaves like `%ch%`, but the number is represented as a number word. You can select between a number of different languages.

`%chi%`
: Begaves like `%ch%`, but represented as a lower case Roman number from 1 to 4999.

`%chI%`
: Behaves like `%ch%`, but represented as an upper case Roman number from 1 to 4999.

`%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.

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.

15.2 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, resulting in a gap in the text.

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.

15.3 File Selection

Which documents and notes are selected for the build can be controlled from the options on the left side of the dialog window. In addition, you can select to include the synopsis comments, regular comments, keywords, and even exclude the body text itself if you just want an outline.

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 set the documents as "inactive" in the project tree. *Build Novel Project* tool has a switch to collectively exclude inactive documents.

15.4 Printing

The print button allows you to print the content in the preview window. You can either print to one of your system's printers, or print directly to a file as PDF. You can also print to file from the regular print dialog. The direct to file option is just a shortcut.

Note: The paper format should in all cases default to whatever your system default is. Of you want to change it, you have to select it from the *Print Preview* ` dialog.

15.5 Export Formats

Currently, six formats are supported.

Open Document Format
: The Build tool can produce either an `.odt` file, or an `.fodt` file. The latter is just a flat version of the document format as a single XML file. Most rich text editors support the former, and a few the latter.

novelWriter HTML
: The HTML format writes a single `.htm` file with minimal style formatting. The HTML document is suitable for further processing by document conversion tools like Pandoc, for importing in word processors, or for printing from browser.

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/GitHub Markdown
: The Markdown format comes in both Standard and GitHub flavour. The *only* difference in terms of novelWriter functionality is the support for strikethrough text, which is not supported by the Standard flavour, but *is* supported by the GitHub flavour.

15.6 Additional Formats

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 included.

The text body is saved in a two-level list. The outer list contains one entry per 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

SIXTEEN

FILE LOCATIONS

novelWriter will create a few files on your system outside of the application folder itself. These file locations are described in this document.

16.1 Configuration

The general configuration of novelWriter, including everything that is in *Preferences*, is saved in one central configuration file. The location of this file depends on your operating system. The system paths are provided by the Qt QStandardPaths class and its ConfigLocation value.

The standard paths are:

- Linux: `~/.config/novelwriter/novelwriter.conf`
- macOS: `~/Library/Preferences/novelwriter/novelwriter.conf`
- Windows: `C:\Users\<USER>\AppData\Local\novelwriter\novelwriter.conf`

Here, ~ corresponds to the user's home directory on Linux and macOS, and <USER> is the user's username on Windows.

Note: These are the standard operating system defined locations. If your system has been set up in a different way, these locations may also be different.

16.2 Application Data

novelWriter also stores a bit of data that is generated by the user's actions. This includes the list of recent projects form the *Open Project* dialog. Custom themes should also be saved here. The system paths are provided by the Qt QStandardPaths class and its AppDataLocation.

The standard paths are:

- Linux: `~/.local/share/novelwriter/`
- macOS: `~/Library/Application Support/novelwriter/`
- Windows: `C:\Users\<USER>\AppData\Roaming\novelwriter\`

Here, ~ corresponds to the user's home directory on Linux and macOS, and <USER> is the user's username on Windows.

Note: These are the standard operating system defined locations. If your system has been set up in a different way, these locations may also be different.

CHAPTER

SEVENTEEN

HOW DATA IS STORED

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

17.1 Project Structure

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 when working on the same files on both Windows and Unix-based operating systems.

17.1.1 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 you do 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.

The project XML file is indent-formatted, and is 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, and various meta data entries incremented, on each save.

17.2 Project Documents

All the project documents are saved in a folder in the main project folder named `content`. Each document has a file handle based on a 52 bit random number, represented as a hexadecimal string. The documents are saved with a filename assembled from this handle and the file extension `.nwd`.

If you wish to find the file system 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, referenced by their label, 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. This is particularly useful when using a versioning system.

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 characters may become mangled when the file is opened in novelWriter again.

Editing these files is generally not recommended outside of special circumstances, whatever they may be. The reason for this is that the index will not be automatically updated when doing so, which means novelWriter doesn't know you've altered the file. If you do edit a file in this manner, you should rebuild the index when you next open the project in novelWriter.

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.

17.2.1 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 then 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.

17.3 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 and the custom words list. The JSON files within this folder can safely be ignored as they will be automatically regenerated if lost.

17.3.1 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.

The integrity of the index is checked when the file is loaded. It is possible to corrupt the index if the file is manually edited and manipulated, so the check is important to avoid sudden crashes of novelWriter. If the file contains errors, novelWriter will automatically build it anew. If the check somehow fails and novelWriter keeps crashing, you can delete the file manually and rebuild the index. If this too fails, you have likely encountered a bug.

17.3.2 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.

17.3.3 Custom Word List

A file named `meta/wordlist.txt` contains all the custom words you've added to the project for spell checking purposes. The content of the file can be edited from the *Tools* menu. If you lose this file, all your custom spell check words will be lost too.

17.3.4 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.

17.4 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. The folder will be removed entirely in novelWriter 2.1.

CHAPTER

EIGHTEEN

RUNNING TESTS

The novelWriter source code is well covered by tests. The test framework used for the development is `pytest` with the use of an extension for Qt.

18.1 Dependencies

The dependencies for running the tests can be installed with:

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

This will install a couple of extra packages for coverage and test management. The minimum requirement is `pytest` and `pytest-qt`.

18.2 Simple Test Run

To run the tests, you simply need to execute the following from the root of the source folder:

```
pytest
```

Since several of the tests involve opening up the novelWriter GUI, you may want to disable the GUI for the duration of the test run. Moving your mouse while the tests are running may otherwise interfere with the execution of some tests.

You can disable the renderring of the GUI by setting the flag `export QT_QPA_PLATFORM=offscreen`, or alternatively run the tests with the `xvfb` package, like so:

```
xvfb-run pytest
```

18.3 Advanced Options

Adding the flag `-v` to the `pytest` command will increase verbosity of the test execution.

You can also add coverage report generation. For instance to HTML:

```
xvfb-run pytest -v --cov=novelwriter --cov-report=html
```

Other useful report formats are `xml`, and `term` for terminal output.

You can also run tests per subpackage of novelWriter with the `-m` command. The available subpackage groups are `base`, `core`, and `gui`. Consider for instance:

```
xvfb-run pytest -v --cov=novelwriter --cov-report=html -m core
```

This will only run the tests of the "core" package, that is, all the classes that deal with the project data of a novelWriter project. The "gui" tests, likewise, will run the tests for the GUI components, and the "base" tests cover the bits in-between.

You can also filter the tests with the `-k` switch. The following will do the same as `-m core`:

```
xvfb-run pytest -v --cov=novelwriter --cov-report=html -k testCore
```

All tests are named in such a way that you can filter them by adding more bits of the test names. They all start with the word "test". Then comes the group: "Core", "Base", "Dlg", "Tool", or "Gui". Finally comes the name of the class or module, which generally corresponds to a single source code file. For instance, running the following will run all tests for the document editor:

```
xvfb-run pytest -v --cov=novelwriter --cov-report=html -k testGuiEditor
```

To run a single test, simply add the full test name to the `-k` switch.