novelWriter

User Guide

Release 2.3b1

Veronica Berglyd Olsen

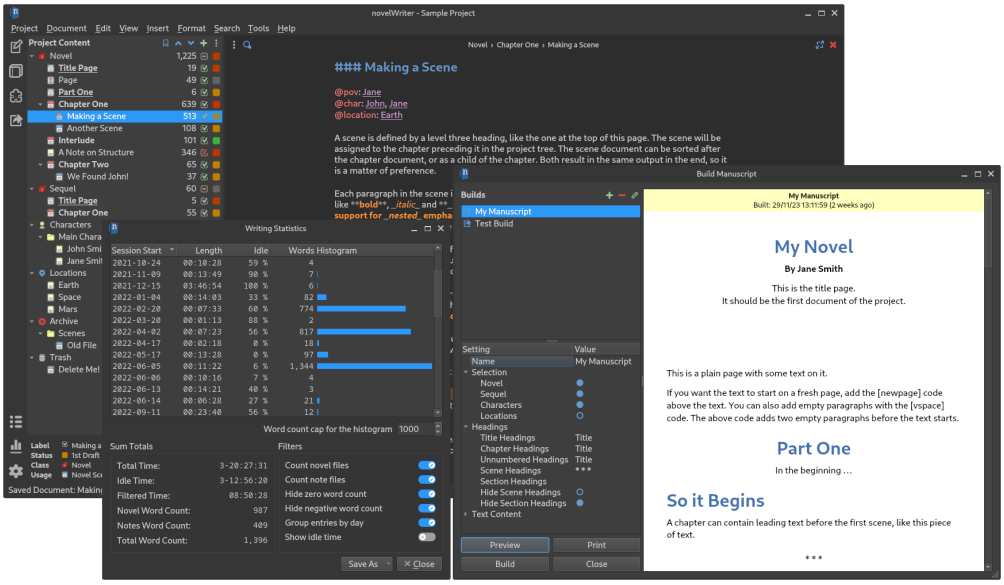
Friday, 16 February 2024 at 18:42

CONTENTS

Release Version: 2.3b1

Updated: Friday, 16 February 2024 at 18:42

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 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 with pip. See *Getting Started* for more details.

Useful Links

- Website: https://novelwriter.io
- Documentation: https://docs.novelwriter.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 its 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 available are limited to those relevant for this purpose. It is *not* suitable for technical writing, and it is *not* a full-featured Markdown editor.

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

There are two kinds of documents in your project: *Novel Documents* are documents that are part of your story. The other kind of documents are *Project Notes*. These are intended for your notes about your characters, your world building, and so on.

You can at any point split the individual documents by their headers up into multiple documents, or merge multiple documents into single documents. This makes it easier to use variations of the Snowflake method for writing. You can start by writing larger structure-focused documents, like one per act for instance, and later effortlessly split these up into scenes by their headers.

Below are some key features of novelWriter.

Focus on writing
: The aim of the user interface is to let you 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 panel itself is visible, and the project structure view is hidden away.

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, your character notes, plot notes, or any other document you may need to reference while writing. It is not intended as a preview panel for the document you're editing, but if you wish, you can also use it for this purpose.

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 into a manuscript, 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 of the project tree. You can also start out with a few documents and then later split them into multiple documents based on their headers.

Multi-novel project support
: As of novelWriter 2.0, you can have multiple Novel type root folders in a project. This allows

you to keep a series of individual novels with the same characters and world building in the same project, and create manuscripts for them individually.

Keep track of your plot elements
All notes in your project can be assigned a *tag* that 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 specific sections of it.

Get an overview of your story
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 chapter or scene, 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 an alternative view to the project tree.

Get an overview of your story elements
Under the document viewer panel you will find a series of tabs that shows the different story elements you have created tags for. The tabs are sorted into Characters, Plots, etc, depending on which categories you are using in your story. This panel can be hidden when you don't need it to free up space.

Building your manuscript
Whether you want to assemble a manuscript, or export all your notes, or generate an outline of your chapters and scenes with a synopsis, you can use the *Build Manuscript* tool to do so. 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

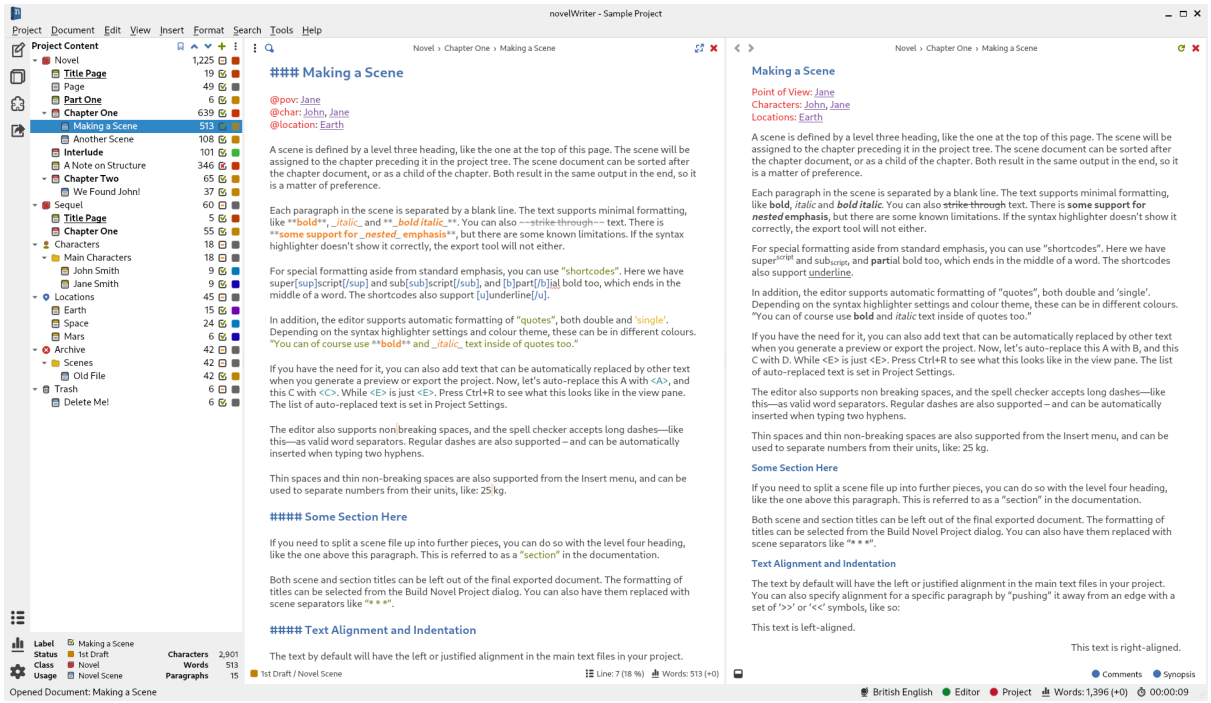


Fig. 1: novelWriter with light colour theme

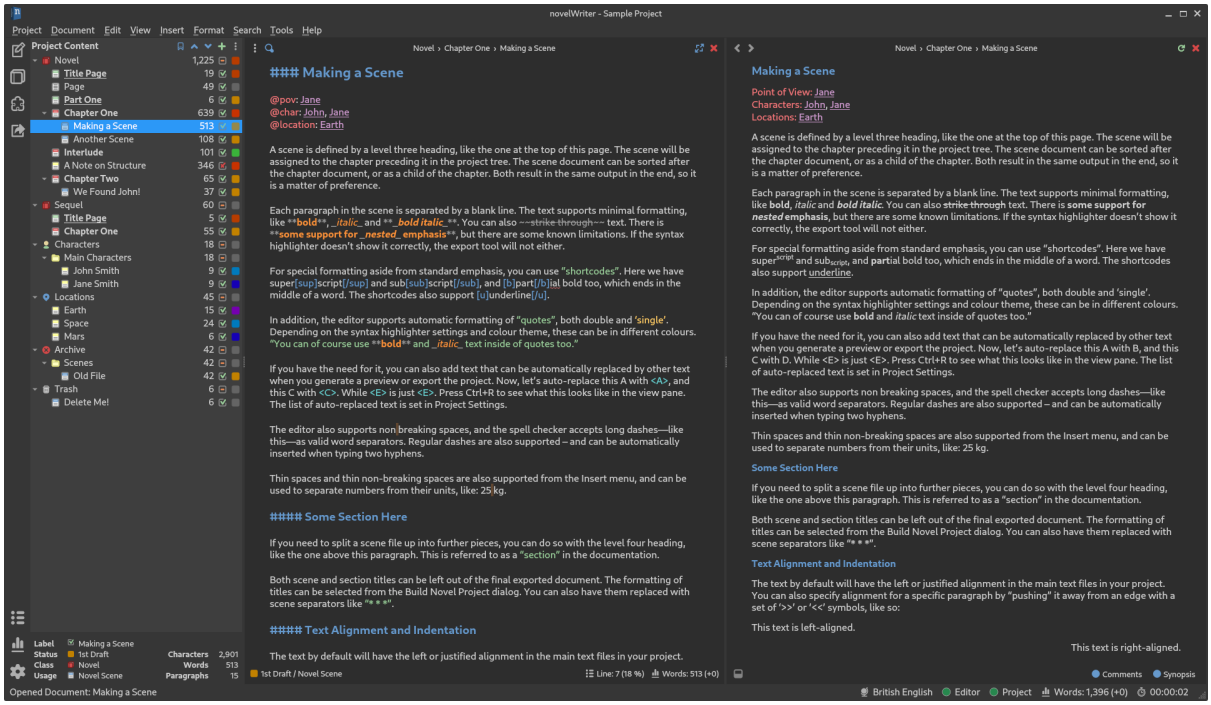


Fig. 2: novelWriter with dark colour theme

CHAPTER

TWO

OVERVIEW

novelWriter is built as a cross-platform application using Python 3 as the programming language, and Qt 5 for the user interface.

novelWriter is built for Linux first, and this is where it works best. However, it also runs fine on Windows and MacOS due to the cross-platform framework it's built on. The author of the application doesn't own a Mac, so on-going Mac support is dependent on user feedback and user contributions.

Spell checking in novelWriter is provided by a third party library called Enchant. Please see the section on *Spell Check Dictionaries* for how to install spell checking languages.

For install instructions, see *Getting Started*.

2.1 Using novelWriter

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

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

Project Views **– Recommended Reading**
: This chapter will give you a more detailed explanation of how you can use the user interface components to organise and view your project work.

The Editor and Viewer **– Recommended Reading**
: This chapter will give you a more detailed explanation of how the text editor and viewer work.

Formatting Your Text **– Essential Information**
: This chapter 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 from how you use headings. Tags and references are implemented by special keywords.

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

Typographical Notes **– Optional**
: This chapter 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 chapter 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 chapter.

2.2 Organising Your Projects

In addition to managing 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 chapter explains how you organise the content of your project, and how to set up automated backups of your work.

Novel Structure – **Essential Information**
: This chapter covers the way your novel's structure is encoded into the text documents. It explains how the different levels of headings are used, and some special formatting for different kinds of headings.

Tags and References **- Recommended Reading**
: This chapter explains how you organise your notes, and how the Tags and References system works. This system lets you cross-link your documents in your project, and display these references in the application interface.

Building the Manuscript **- Recommended Reading**
: This chapter explains how the *Manuscript Build* tool works, how you can control the way chapter titles are formatted, and how scene and section breaks are handled.

CHAPTER

THREE

GETTING STARTED

Ready-made packages and installers for novelWriter are available for all major platforms, including Linux, Windows and MacOS. See below for install instructions for each platform.

You can also install novelWriter from the Python Package Index (PyPi). See *Installing from PyPi*. Installing from PyPi does not set up icon launchers, so you will either have to do this yourself, or start novelWriter from the command line.

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.

3.1 Installing 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 as having multiple installations has been known to cause problems.

Note: The novelWriter installer is not signed because Microsoft doesn't currently provide a way for non-profit open source projects to properly sign their installers. The novelWriter project doesn't have the funding to pay for commercial software signing certificates. You will therefore see an additional warning about this when you download the installer.

3.2 Installing 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 the Pre-Release PPA instead, add the `ppa:vkbo/novelwriter-pre` repository.

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, as it is Ubuntu-specific.

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 noble 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: You may need to use the Ubuntu 20.04 (focal) packages for Debian 11 or earlier. The newer Ubuntu packages use a different compression algorithm that may not be supported.

Tip: If you get an error message like `gpg: failed to create temporary file` when importing the key from the Ubuntu keyserver, try creating the folder it fails on, and import the key again:

```
sudo mkdir /root/.gnupg/
```

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.

Note: novelWriter generally don't support Python versions that have reached end of life. If your Linux distro still uses older Python versions and novelWriter won't run, you may want to try the AppImage instead.

3.3 Installing on MacOS

You can install novelWriter with both its Python and library dependencies embedded using the DMG application image file from the main website, or from the Releases page on GitHub. Installing it should be straightforward.

- Download the DMG file and open it. Then drag the novelWriter icon to the *Applications* folder on the right. This will install it into your *Applications*.
- The first time you try to launch it, it will say that the bundle cannot be verified, simply press the *Open* button to add an exception.
- If you are not presented with an *Open* button in the dialog launch the application again by right clicking on the application in Finder and selecting *Open* from the context menu.

The context menu can also be accessed by option-clicking if you have a one button mouse. This is done by holding down the option key on your keyboard and clicking on the application in Finder.

Note: The novelWriter DMG is not signed because Apple doesn't currently provide a way for non-profit open source projects to properly sign their installers. The novelWriter project doesn't have the funding to pay for commercial software signing certificates.

3.4 Installing 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. You can download Python from python.org. It is recommended that you install the latest version. If you are on Windows, also make sure to select the "Add Python to PATH" option during installation.

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, and then you may get a "Not Found" error when running the `novelwriter` command.

CHAPTER

FOUR

TIPS & TRICKS

This is a list of hopefully helpful little tips on how to get the most out of novelWriter.

Note: This section will be expanded over time. If you would like to have something added, feel free to contribute, or start a discussion on the project's Discussions Page.

4.1 Managing the Project

Merge Multiple Documents Into One

If you need to merge a set of documents in your project into a single document, you can achieve this by first making a new folder for just that purpose, and drag all the files you want merged into this folder. Then you can right click the folder, select *Transform* and *Merge Documents in Folder*.

In the dialog that pops up, the documents will be in the same order as in the folder, but you can also rearrange them here of you wish. See *Splitting and Merging Documents* for more details.

4.2 Layout Tricks

Create a Simple Table

The formatting tools available in novelWriter don't allow for complex structures like tables. However, the editor does render tabs in a similar way that regular word processors do. You can set the width of a tab in *Preferences*.

The tab key should have the same distance in the editor as in the viewer, so you can align text in columns using the tab key, and it should look the same when viewed next to the editor.

This is most suitable for your notes, as the result in exported documents cannot be guaranteed to match.

4.3 Organising Your Text

Add Introductory Text to Chapters

Sometimes chapters have a short preface, like a brief piece of text or a quote to set the stage before the first scene begins.

If you add separate files for chapters and scenes, the chapter file is the perfect place to add such text. Separating chapter and scene files also allows you to make scene files child documents of the chapter (added in novelWriter 2.0).

Distinguishing Soft and Hard Scene Breaks

Depending on your writing style, you may need to separate between soft and hard scene breaks within chapters. Like for instance if you switch point-of-view character often.

In such cases you may want to use the scene heading for hard scene breaks and section headings for soft scene breaks. the *Build Manuscript* tool will let you add separate formatting for the two when you generate your manuscript. You can for instance add the common "* * *" for hard breaks and select to hide section breaks, which will just insert an empty paragraph in their place. See *Build Settings* for more details.

4.4 Other Tools

Convert Project to/from yWriter Format

There is a tool available that lets you convert a yWriter project to a novelWriter project, and vice versa.

The tool is available at peter88213.github.io/yw2nw

CHAPTER

FIVE

CUSTOMISATIONS

There are a few ways you can customise novelWriter yourself. 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 your operating system, it may or may not load all installed spell check dictionaries automatically.

5.1.1 Linux and MacOS

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

5.1.2 Windows

For Windows, English is included with the installation. For other languages you have to download and add dictionaries yourself.

Install Tool

A small tool to assist with this can be found under *Tools* > *Add Dictionaries*. It will import spell checking dictionaries from Free Office or Libre Office extensions. The dictionaries are then installed in the install location for the Enchant library.

Manual Install

If you prefer to do this manually or want to use a different source than the ones mentioned above, You need to get compatible dictionary files for your language. You need two files 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.

You can find the various dictionaries on the Free Desktop website.

Note: The Free Desktop link points to a repository, and what may look like file links inside the dictionary folder are actually links to web pages. If you right-click and download those, you get HTML files, not dictionaries!

In order to download the actual dictionary files, right-click the "plain" label at the end of each line and download that.

5.2 Syntax and GUI Themes

Adding your own GUI and syntax themes is relatively easy, although it requires that you manually edit config files with colour values. 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.
- The existing icon themes are stored in novelwriter/assets/icons.

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, just `themes` for GUI themes, and `icons` for icon 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.

New in version 2.0: The `icontheme` value was added to GUI themes. Make sure you set this value in existing custom themes. Otherwise, novelWriter will try to guess your icon theme, and may not pick the most suitable one.

5.2.1 Custom GUI and Icons Theme

A GUI theme `.conf` file consists of the following 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
```

(continues on next page)

(continued from previous page)

```
[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`, or to an icon theme in your custom icons directory. The setting must match the icon theme's folder name.

The Palette values correspond to 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 following 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
```

(continues on next page)

(continued from previous page)

```
doublequotes    =   0,   0,   0
singlequotes    =   0,   0,   0
hidden          =   0,   0,   0
shortcode       =   0,   0,   0
keyword         =   0,   0,   0
value           =   0,   0,   0
optional        =   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 of the format `r, g, b` where each is an integer from to 255. Omitted values default to black, except `background` which defaults to white,

New in version 2.2: The *shortcode* syntax colour entry was added, so you need to update your custom themes if you made any before version 2.2.

New in version 2.3: The *optional* syntax colour entry was added, so you need to update your custom themes if you made any before version 2.3.

CHAPTER

SIX

GLOSSARY

Context Menu
: A context menu is a menu that pops up when you right click something in the user interface. In novelWriter, you can often also open a context menu by pressing the keyboard shortcut `Ctrl+.`.

Headings
: Each level of headings in *Novel Documents* have a specific meaning in terms of the structure of the story. That is, they determine what novelWriter considers a partition, a chapter, a scene or a text section. For *Project Notes*, the header levels don't matter. For more details on headings in novel documents, see *Importance of Headings*.

Keyword
: A keyword in novelWriter is a special command you put in the text of your documents. They are not standard Markdown, but are used in novelWriter to add information that is interpreted by the application. For instance, keywords are used for *tags* and *references*.

  Keywords must always be on their own line, and the first character of the line must always be the `@` character. The keyword must also always be followed by a `:` character, and the values passed to the command are added after this, separated by commas.

Novel Documents
: These are documents that are created under a "Novel" *root folder*. They behave differently than *Project Notes*, and have some more restrictions. For instance, they can not exist in folders intended only for project notes. See the *Novel Structure* chapter for more details.

Project Index
: The project index is a record of all headings in a project, with all their meta data like synopsis comments, *tags* and *references*. The project index is kept up to date automatically, but can also be regenerated manually from the *Tools* menu or by pressing `F9`.

Project Notes
: Project Notes are unrestricted documents that can be placed anywhere in your project. You should not use these documents for story elements, only for notes. Project notes are the source files used by the Tags and References system. See the *Tags and References* chapter for more details on how to use them.

Reference
: A reference is one of a set of *keywords* that can be used to link to a *tag* in another document. The reference keywords are specific to the different *root folder* types. A full overview is available in the *Tags and References* chapter.

Root Folder
: A "Root Folder" is a top level folder of the project tree in novelWriter. Each type of root folder has a specific icon to identify it. For an overview of available root folder types, see *Project Roots*.

Tag

A tag is a user defined value assigned as a tag to a section of your *Project Notes*. It is optional, and can be defined once per heading. It is set using the *keyword* syntax `@tag: value`, where `value` is the user defined part. Each tag can be referenced in another file using one of the *reference* keywords. See *Tags and References* chapter for more details.

CHAPTER

SEVEN

HOW IT WORKS

The main features of novelWriter are listed in the *Key Features* chapter. In this chapter, 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 more detail.

7.1 GUI Layout and Design

The user interface of novelWriter is intended to be as minimalistic as practically possible, while at the same time provide useful 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, it is more or less redundant.

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* chapter.

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 side 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 Manuscript* tool, and a set of project-related tools and quick access to settings at the bottom.

New in version 2.2: A number of new formatting options were added in 2.2 to allow for some special formatting cases. At the same time, a small formatting toolbar was added in the editor. It is hidden by default, but can be opened by pressing the three dots icon in the top right corner.

7.1.1 Project Tree and Editor 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 contains 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 purpose if you wish as it will apply the formatting tags you have

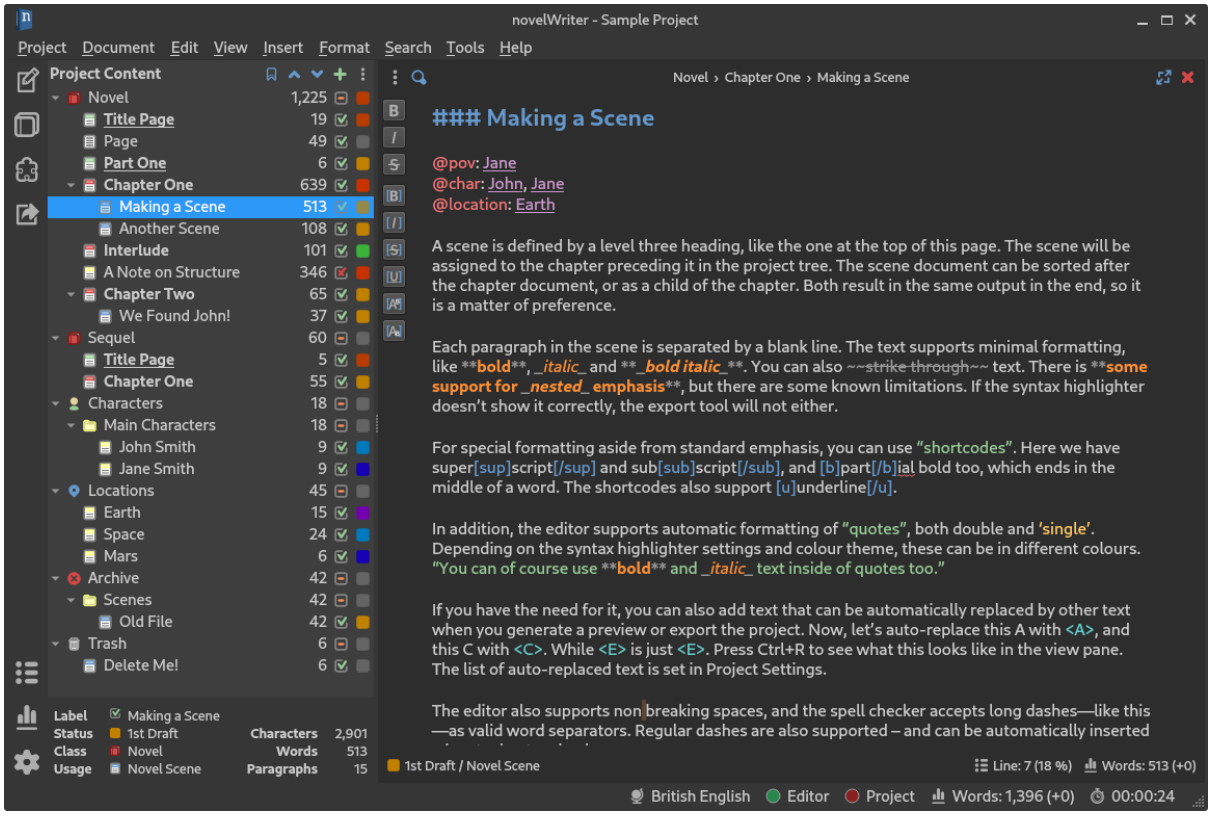


Fig. 1: A screenshot of the Project Tree and Editor View.

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's header, or by pressing `F8`. When *Focus Mode* is enabled, all the user interface elements other than the document editor itself are hidden away.

7.1.2 Novel Tree and Editor View

When in *Novel Tree View* mode, the project tree is replaced by an overview of your novel structure for a specific Novel *root folder*. 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 in the same document.

If you have multiple Novel root folders, the header of the novel view becomes a dropdown 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 triggered, or you manually press `Ctrl+S` to save changes. (You can adjust the auto-save interval in *Preferences*.) You can also regenerate the whole novel view by pressing the refresh button at the top of the side panel.

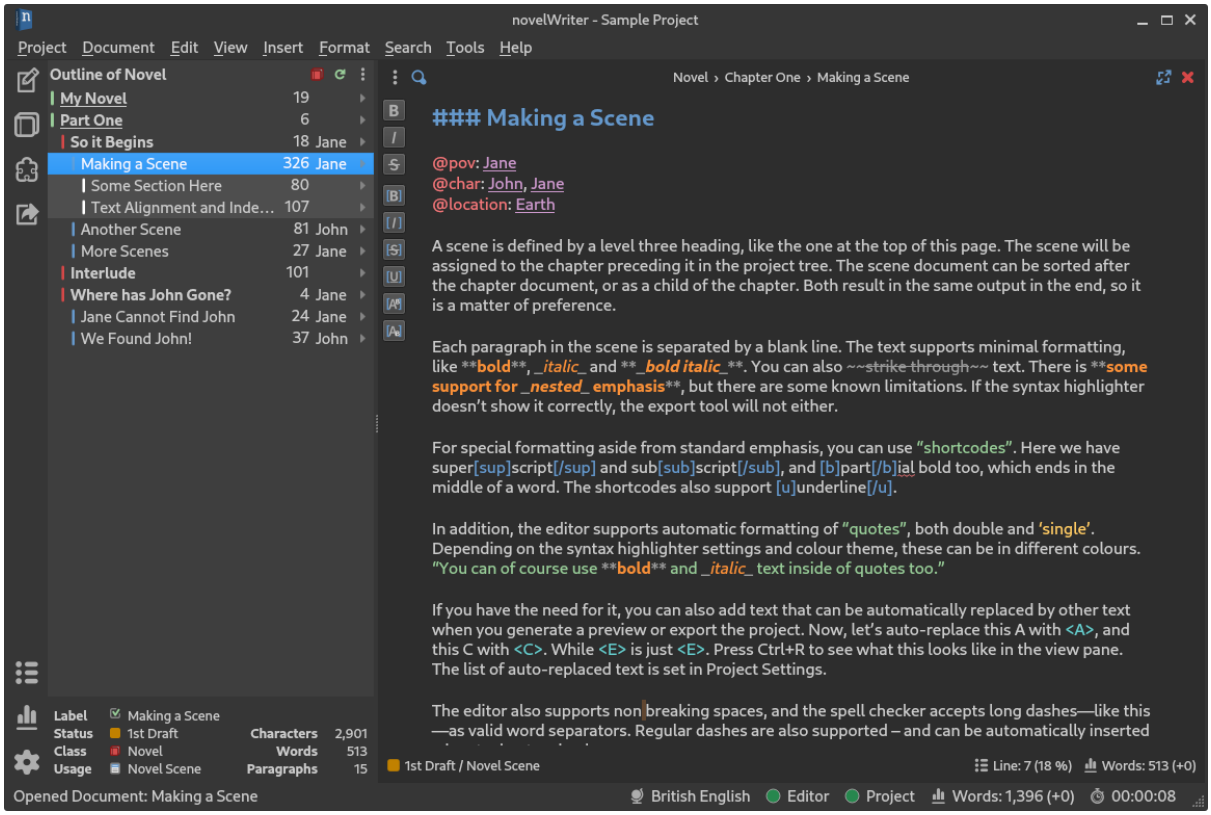


Fig. 2: A screenshot of the Novel Tree View.

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

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

7.1.3 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 your column order and size between sessions, and for each individual project.

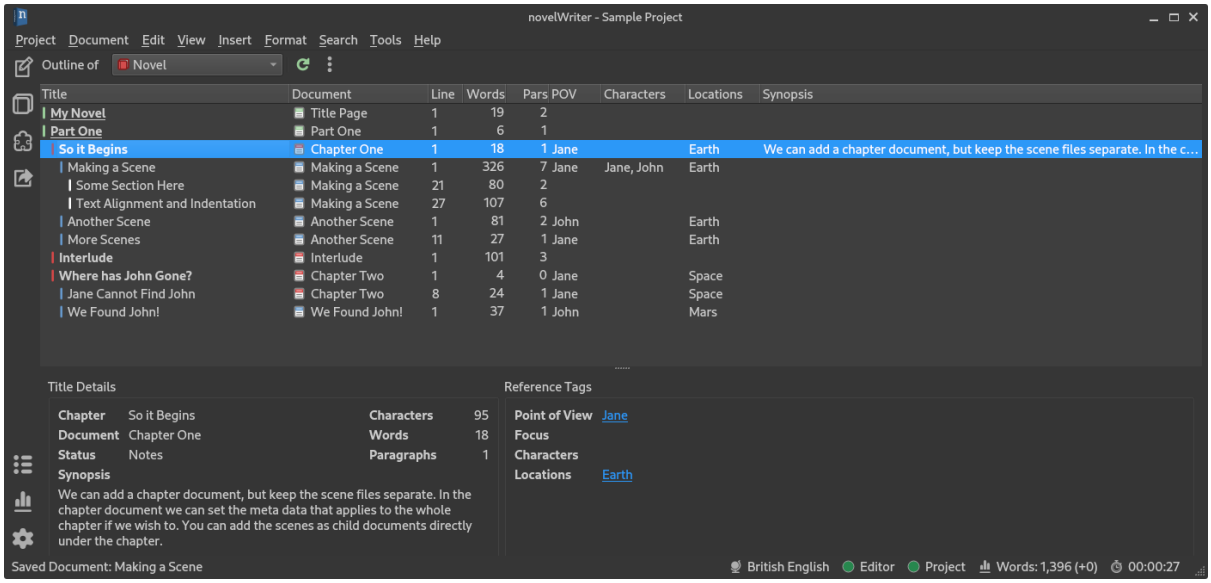


Fig. 3: A screenshot of the Novel Outline View.

7.1.4 Colour Themes

By default, novelWriter will use the colour theme provided by the Qt library, which is determined by the Fusion style setting. You can also choose between a standard dark and light theme that have neutral colours from *Preferences*. Other colour themes are also available. More themes can be contributed to novelWriter on GitHub.

Switching the GUI colour theme does not affect the colours of the editor and viewer. They have separate colour themes called *Editor Themes*. They are separated because there are a lot more options to choose from for the editor and viewer.

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

7.2 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 sub-folders or sub-documents, 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 novel 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.

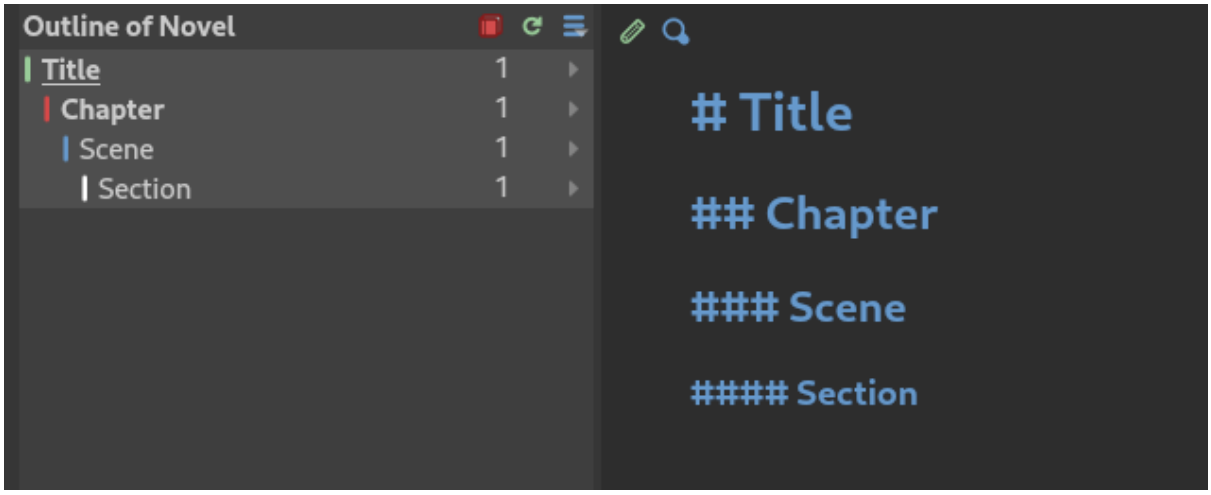


Fig. 4: An illustration of how header levels correspond to the novel structure.

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 *project notes*, the header levels have no structural meaning, and you are free to use them however you want. See *Novel Structure* and *Tags and References* for more details.

New in version 2.0: You can add documents as child items of other documents. 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 so that you can rearrange scene documents freely without affecting chapter placement.

7.3 Building the Manuscript

The project can at any time be assembled into a range of different formats through the *Build Manuscript* 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. The Open Document format is supported by most Office type applications. In addition, printing is also possible. Print to PDF is available from the print dialog.

In addition, you can export the content of the project to a JSON file. This is useful if you want to write your own custom 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 with either HTML formatted text, or with the raw text as typed into the novel documents.

See *Building the Manuscript* for more details.

New in version 2.1: You can now define multiple build definitions in the *Build Manuscript* tool. This allows you to define specific settings for various types of draft documents, outline documents, and manuscript formats. See *Building the Manuscript* for more details.

7.4 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 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, which means there is a smaller risk of losing data if the application or your computer crashes.

Secondly, having multiple small files means it is very easy to synchronise 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, as a rule, manually edit files within it. If you really must manually edit the text files, e.g. with some automated task you want to perform, you need to rebuild the *Project Index* when you open the project again.

Editing text files in the `content` folder is less risky as these 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

EIGHT

PROJECT VIEWS

This chapter covers in more detail the different project views available in novelWriter.

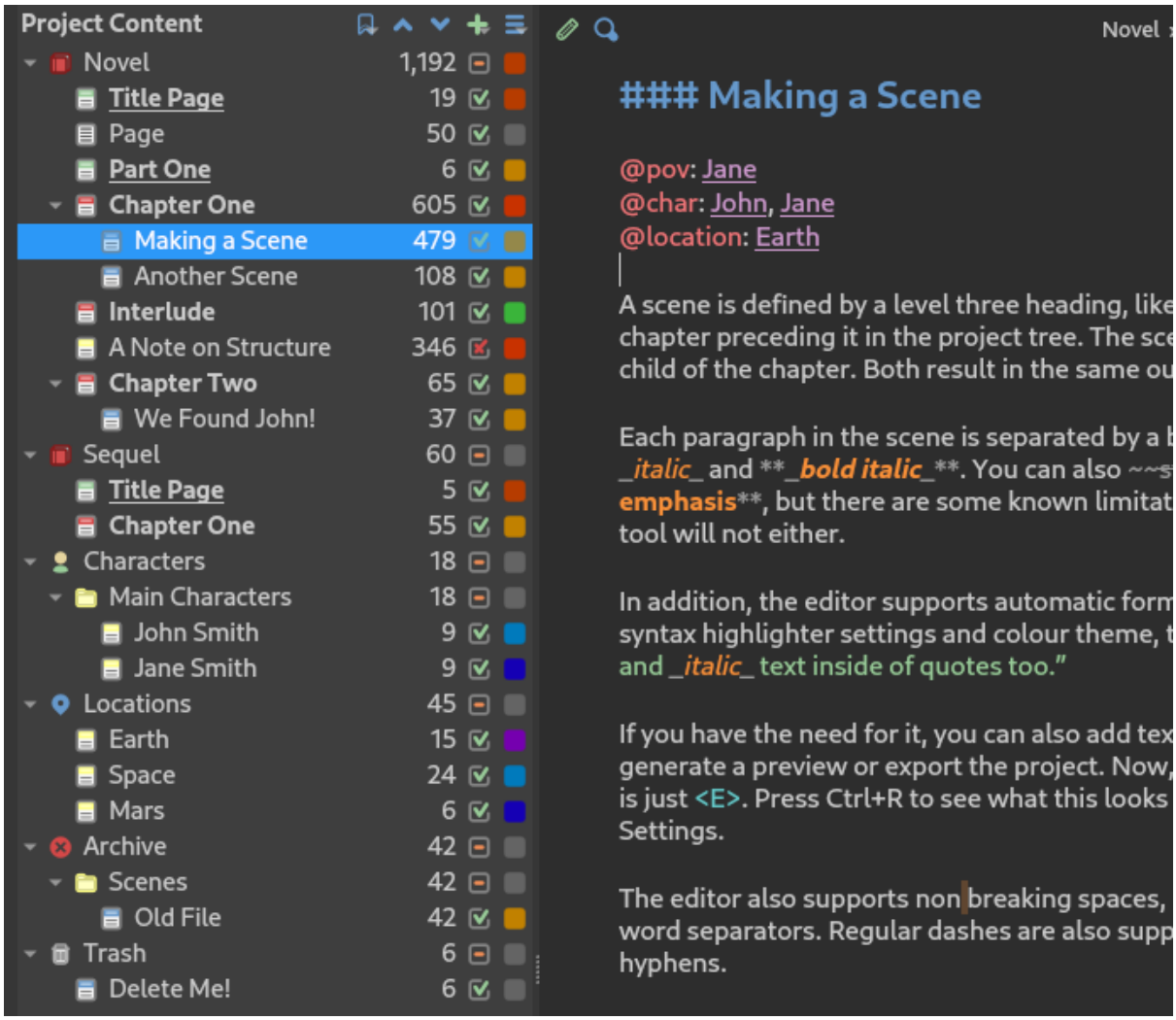


Fig. 1: The Project Tree as it appears when loading a sample project.

8.1 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 sub-documents. 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 Manuscript* tool, you can include or exclude documents 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 on how to uses these labels. 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 `Ctrl+L`.
- 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 `Ctrl+N`.
- The last button is a menu of further actions you can apply to the project tree.

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.

Tip: If you want to set the label of a document to be the same as a header within it, you can right-click a header in the document when it is open in the editor and select *Set as Document Name* from the context menu.

8.1.1 Splitting and Merging Documents

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

Splitting Documents

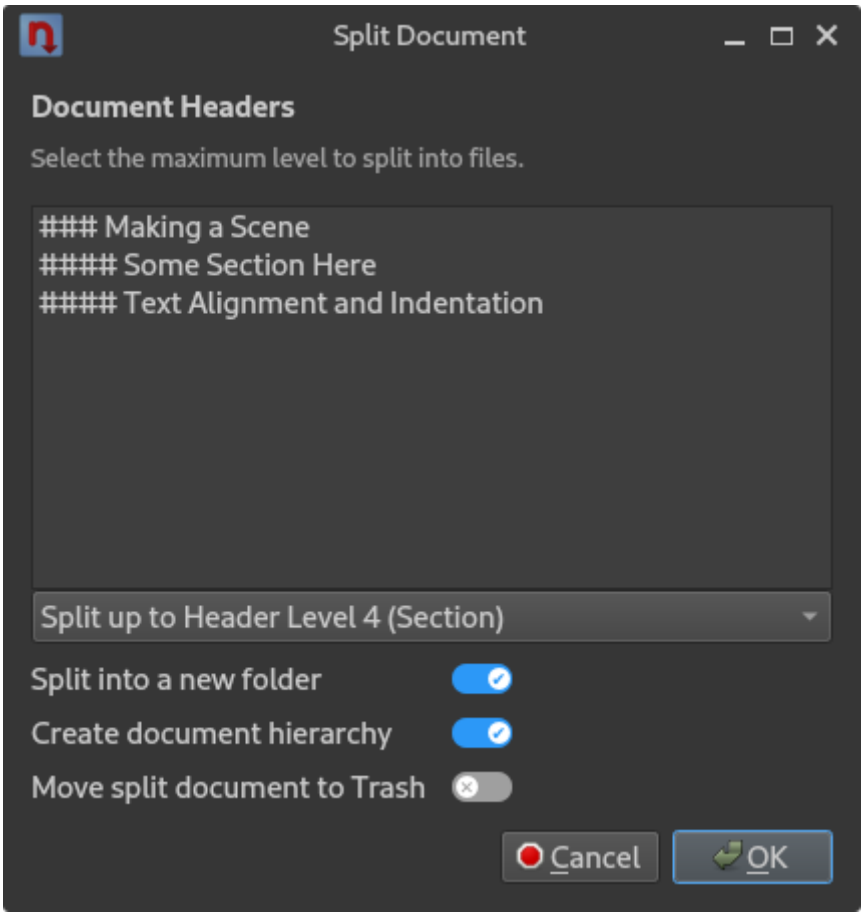


Fig. 2: The *Split Document* dialog.

The *Split Document by Header* option will open a dialog that allows you to split the selected document into multiple new documents based on the headings it contains. You can select at which heading level the split is to be performed from the dropdown box. The list box will preview which headings will be split into new documents.

You are given the option to create a folder for these new documents, and whether or not to create a hierarchy of documents. That is, put sections under scenes, and scenes under chapters.

The source document *is not* deleted in the process, but you have the option to let the tool move the source document to the *Trash* folder.

Merging Documents

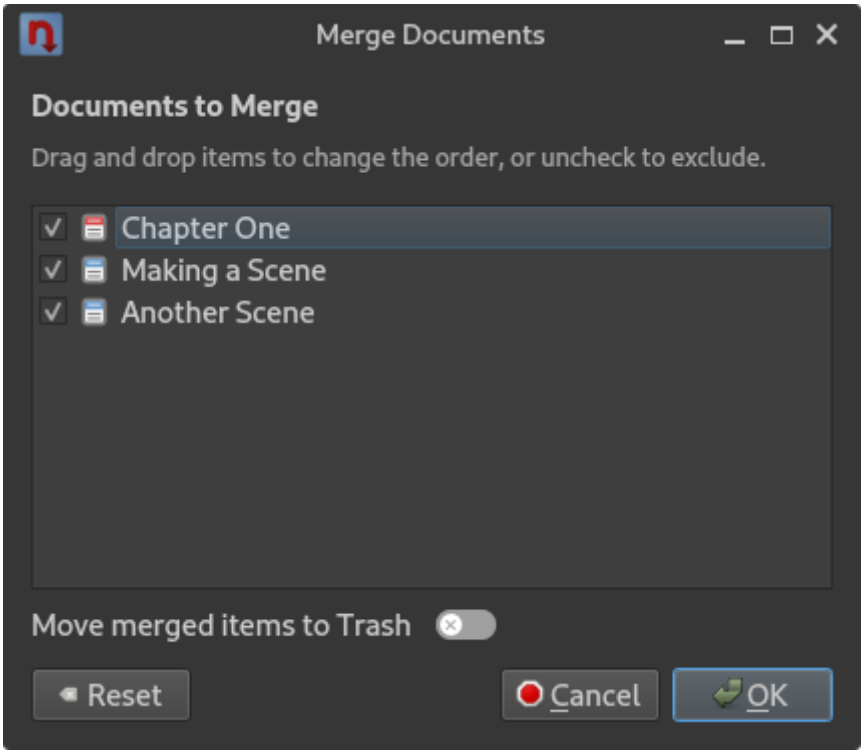


Fig. 3: The *Merge Documents* dialog.

You have two options for merging documents that are child elements of another document. You can either *Merge Child Items into Self* and *Merge Child Items into New*. The first option will pull all content of child items and merge them into the parent document, while the second option will create a new document in the process.

When merging documents in a folder, you only have the latter process is possible, so only the choice *Merge Documents in Folder* is available.

In either case, the *Merge Documents* dialog will let you exclude documents you don't want to include, and it also lets you reorder them if you wish.

8.1.2 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. novelWriter 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 *project notes*, as for instance a main, major, or minor character or story element.

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

8.1.3 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 *Manuscript Build* tool.

New in version 2.2: You can now select multiple items in the project tree by holding down the `Ctrl` or `Shift` key while selecting items.

You can drag and drop documents and regular folders, but not root folders. If you select multiple items, they can only be dragged and dropped if they are siblings. That is, they have the same parent item in the project. This is due to the way drag and drop is implemented in the user interface framework novelWriter is built upon.

Documents and their folders can be rearranged freely within their root folders. If you move a Novel document 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 and dropped at all. If you want to reorder them, you can move them up or down with respect to each other from the arrow buttons at the top of the project tree, or by pressing `Ctrl+Shift+Up` or `Ctrl+Shift+Down` when they are selected.

8.2 The Novel Tree

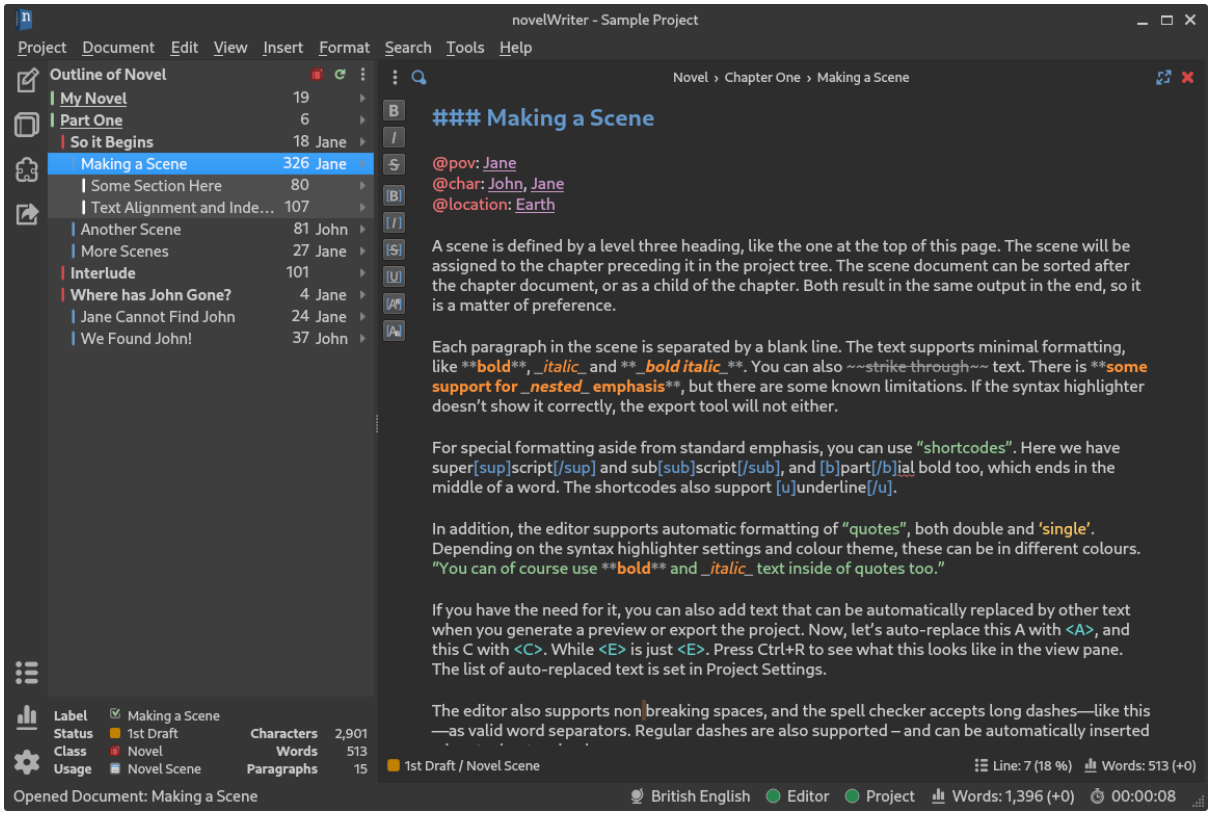


Fig. 4: A screenshot of the Novel Tree View.

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 View*. 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 themselves. Any such 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 when the document is saved.

8.3 Project Outline View

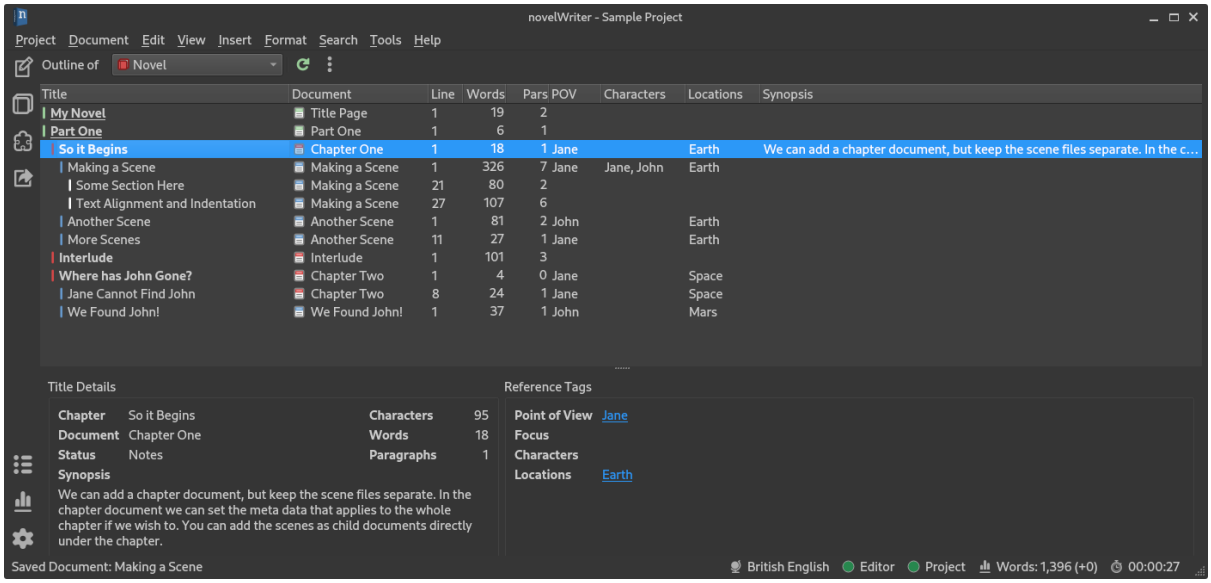


Fig. 5: A screenshot of the Novel Outline View.

The project's *Outline View* is available as another view option from the sidebar. 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 the heading is defined. Double-clicking an entry will open the corresponding document in the editor and switch to *Project Tree View* mode.

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, this view 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 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

NINE

THE EDITOR AND VIEWER

This chapter covers in more detail how the document editor and viewer panels work.

9.1 Editing a Document

Fig. 1: A screenshot of the Document Editor panel.

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* chapter.

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 a tools button that opens a toolbar with a few buttons for applying text formatting, and a search button to open the search dialog.

Both the document editor and viewer will show the label of the currently open 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 thus reveal its location there, making it easier to find in a large project.

Any *references* in the editor can be opened in the viewer by moving the cursor to the label and pressing `Ctrl+Return`. You can also control-click them with your mouse.

9.1.1 Editor Auto-Completer

If you type the character `@` on a new line, a context menu will appear showing the different available keywords. The list will shorten as you type. Once a keyword command has been selected or typed, the editor may suggest further content based on your project content. See *The References Auto-Completer* for more details.

New in version 2.2: The auto-completer feature was added.

9.2 Viewing a Document

Any document in the project tree can also be viewed in parallel in a right hand side document viewer. To view a document, press `Ctrl+R`, 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 the one currently being edited. However, If you *are* viewing the same document, pressing `Ctrl+R` again will update the document with your latest changes. You can also press the reload button in the top–right corner of the viewer panel, next to the close button, to achieve the same thing.

In the viewer *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 button on the left side of the footer area to reveal it.) This panel contains a References tab with links to all documents referring back to the one you're currently viewing, if any has been defined. If you have created root folders and tags for various story elements like characters and plot points, these will appear as additional tabs in this panel.

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`.

Fig. 2: A screenshot of the Document Viewer panel.

New in version 2.2: The reference panel was redesigned and the additional tabs added.

9.3 Search & Replace

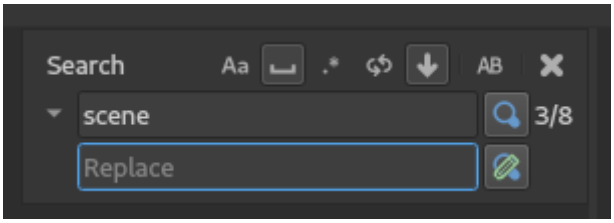


Fig. 3: A screenshot of the Document Editor search box.

The document editor has a search and replace tool that can be activated with `Ctrl+F` for search mode or `Ctrl+H` for search and replace mode.

Pressing `Return` while in the search box will search for the next occurrence of the word, and `Shift+Return` 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.

See also:

For more information on the capabilities of the Regular Expression option, see the Qt documentation for the QRegularExpression class.

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

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 `Ctrl+Z` once after the auto-replacement will undo it without undoing the character you typed before it.

CHAPTER

TEN

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 strike through text, as well as four levels of headings. Form some further complex formatting needs, a set of shortcodes can be used.

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 to whole paragraphs. See *Text Paragraphs* for more details.

10.1 Syntax Highlighting

The editor has a syntax highlighter feature that is meant to help you know when you've used the formatting 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.

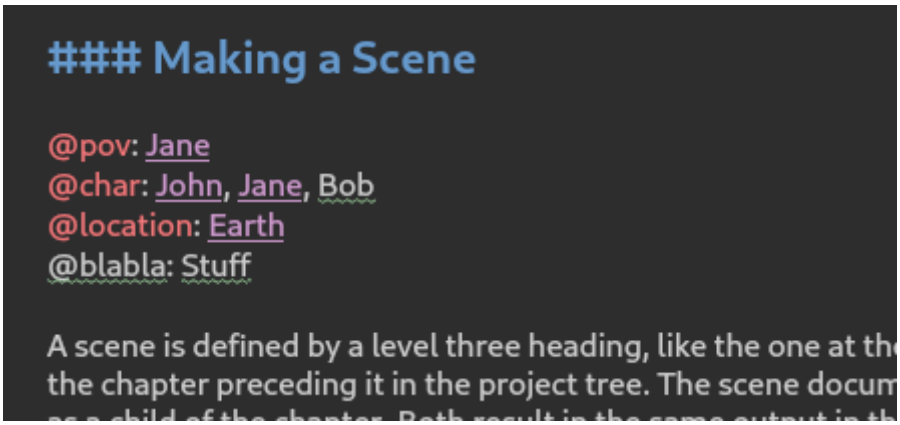


Fig. 1: An example of the colour highlighting of references. "Bob" is not defined, and "@blabla" is not a valid reference type.

When you use the commands to set tags and references, these also change colour. Correct commands have a distinct 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 from *Preferences*.

10.2 Headings

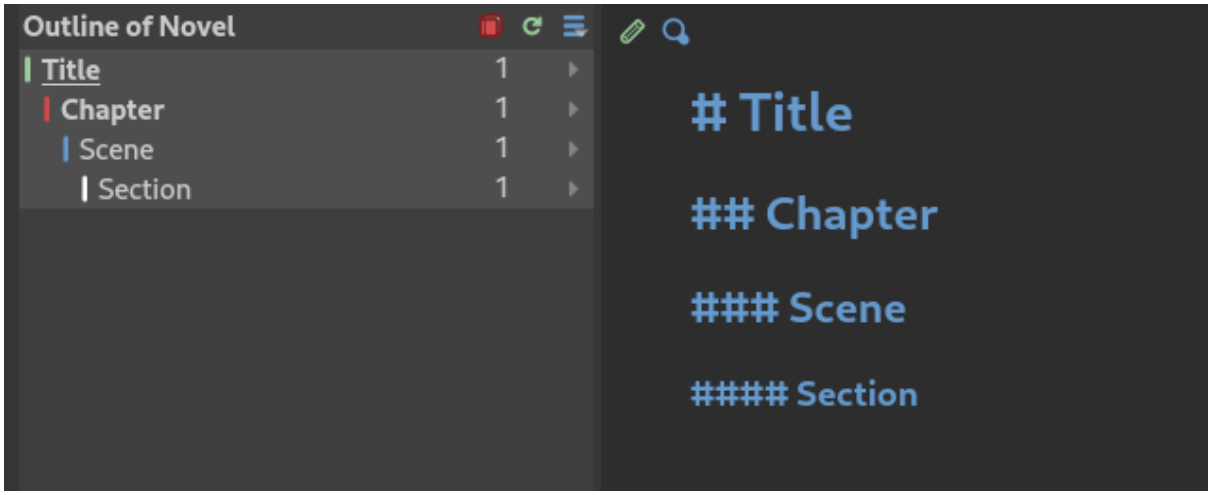


Fig. 2: An illustration of how header levels correspond to the novel structure.

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 ignored 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.

10.3 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 white spaces:

- A non-breaking space can be inserted with `Ctrl`+`K`, `Space`.
- Thin spaces are also supported, and can be inserted with `Ctrl`+`K`, `Shift`+`Space`.
- Non-breaking thin space can be inserted with `Ctrl`+`K`, `Ctrl`+`Space`.

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.

10.4 Text Emphasis

A minimal set of text emphasis styles are supported for text paragraphs.

`_text_`
: The text is rendered as emphasised text (italicised).

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

`~~text~~`
: Strike through 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 strike through 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.
5. Text emphasis can only be used in plain paragraphs. Comments, titles, and meta data tags don't allow for formatting, and any formatting markup will be rendered as-is.

Tip: novelWriter supports standard escape syntax for the emphasis markup characters in case the editor misunderstands your intended usage of them. That is, *, _ and \~ will generate a plain *, _ and ~, respectively, without interpreting them as part of the markup.

10.5 Extended Formatting with Shortcodes

For additional formatting options, you can use shortcodes. Shortcodes is a form of in-line codes that can be used to change the format of the text that follows and opening code, and last until that formatting region is ended with a closing code.

These shortcodes are intended for special formatting cases, or more complex cases that cannot be solved with simple Markdown-like formatting codes. Available shortcodes are listed below.

Table 1: Shortcodes Formats

Syntax	Description
`[b]text[/b]`	Text is rendered as bold text.
`[i]text[/i]`	Text is rendered as italicised text.
`[s]text[/s]`	Text is rendered as strike through text.
`[u]text[/u]`	Text is rendered as underlined text.
`[sup]text[/sup]`	Text is rendered as superscript text.
`[sub]text[/sub]`	Text is rendered as subscript text.

Unlike Markdown style codes, these can be used anywhere within a paragraph. Even in the middle of a word if you need to. You can also freely combine them to form more complex formatting.

The shortcodes are available from the *Format* menu and in the editor toolbar, which can be activated by clicking the three dots in the editor header.

New in version 2.2.

10.6 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 in a special manner 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 rendered 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.

%Short: text ...
: This is a short description comment. It is identical to the synopsis comment, but is intended to be used for project notes. The text shows up in the Reference panel below the document viewer in the last column labelled *Short Description*.

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

10.7 Tags and References

The document editor supports a set of keywords used for setting tags, and making references between documents.

Tags use the command `@tag:` to define a tag. 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.

@tag: value
: A tag command followed by the tag value, like for instance the name of a character.

References can be set anywhere within a section, and are collected according to their category. References are in the form:

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

Tags and references are covered in detail in the *Tags and References* chapter. The keywords can be inserted at the cursor position in the editor via the *Insert* menu. If you start typing an `@` on a new line, and auto-complete menu will also pop up suggesting keywords.

10.8 Paragraph Alignment and Indentation

All documents have the text by default aligned to the left or justified, depending on your settings in *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 2: 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 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.

10.9 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.

If you want page 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.

Example:

```
This is a text paragraph.

[VSPACE:2]

This is another text paragraph, but there will be two empty paragraphs
in-between them.

[NEWPAGE]

This text will always start on a new page if the build format has pages.
```

CHAPTER

ELEVEN

KEYBOARD SHORTCUTS

Most features in novelWriter are available as keyboard shortcuts. This is a reference list of those shortcuts. Most of them are also listed in the application's user interface.

Note: On MacOS, replace `Ctrl` with `Cmd`.

11.1 Main Window Shortcuts

Shortcut	Description
`F1`	Open the online user manual
`F5`	Open the *Build Manuscript* tool
`F6`	Open the *Writing Statistics* tool
`F8`	Toggle *Focus Mode*
`F9`	Re-build the project index
`F11`	Toggle full screen mode
`Ctrl+,`	Open the *Preferences* dialog
`Ctrl+E`	Switch focus to the document editor
`Ctrl+T`	Switch focus to the project/novel tree
`Ctrl+Q`	Exit novelWriter
`Ctrl+Shift+,`	Open the *Project Settings* dialog
`Ctrl+Shift+O`	Open a project
`Ctrl+Shift+S`	Save the current project
`Ctrl+Shift+T`	Switch focus to the outline view
`Ctrl+Shift+W`	Close the current project
`Shift+F1`	Open the local user manual (PDF) if it is available
`Shift+F6`	Open the *Project Details* dialog

11.2 Project Tree Shortcuts

Shortcut	Description
F2	Edit the label of the selected item
Return	Open the selected document in the editor
Alt+Up	Jump or go to the previous item at same level in the tree
Alt+Down	Jump or go to the next item at same level in the tree
Alt+Left	Jump to the parent item in the tree
Alt+Right	Jump to the first child item in the project tree
Ctrl+.	Open the context menu on the selected item
Ctrl+L	Open the *Quick Links* menu
Ctrl+N	Open the *Create New Item* menu
Ctrl+O	Open selected document
Ctrl+R	Open the selected document in the viewer
Ctrl+Up	Move selected item one step up in the tree
Ctrl+Down	Move selected item one step down in the tree
Ctrl+Shift+Del	Move the selected item to Trash
Ctrl+Shift+Z	Undo the last move of a project item, if possible

11.3 Document Editor Shortcuts

11.3.1 Text Search Shortcuts

Shortcut	Description
F3	Find the next occurrence of the search word
Ctrl+F	Open the search bar and search for the selected word, if any is selected
Ctrl+G	Find the next occurrence of the search word
Ctrl+H	Open the search tool and populate with the selected word (Mac Cmd+=)
Ctrl+Shift+1	Replace selected occurrence of the search word, and move to the next
Ctrl+Shift+G	Find the previous occurrence of the search word
Shift+F3	Find the previous occurrence of the search word

11.3.2 Text Formatting Shortcuts

Shortcut	Description
Ctrl+'	Wrap selected text, or word under cursor, in single quotes
Ctrl+"	Wrap selected text, or word under cursor, in double quotes
Ctrl+/	Toggle block format as comment
Ctrl+0	Remove block formatting for block under cursor
Ctrl+1	Change block format to header level 1
Ctrl+2	Change block format to header level 2
Ctrl+3	Change block format to header level 3
Ctrl+4	Change block format to header level 4
Ctrl+5	Change block alignment to left-aligned
Ctrl+6	Change block alignment to centred
Ctrl+7	Change block alignment to right-aligned
Ctrl+8	Add a left margin to the block
Ctrl+9	Add a right margin to the block
Ctrl+B	Format selected text, or word under cursor, with strong emphasis (bold)
Ctrl+D	Strike through selected text, or word under cursor
Ctrl+I	Format selected text, or word under cursor, with emphasis (italic)
Ctrl+Shift+/	Remove block formatting for block under cursor
Ctrl+Shift+D	Toggle block format as ignored text

11.3.3 Other Editor Shortcuts

Shortcut	Description
F7	Re-run the spell checker
Ctrl+.	Open the context menu at the current cursor location
Ctrl+A	Select all text in the document
Ctrl+C	Copy selected text to clipboard
Ctrl+K	Activate the insert commands (see list in *Insert Shortcuts*)
Ctrl+R	Open or reload the current document in the viewer
Ctrl+S	Save the current document
Ctrl+V	Paste text from clipboard to cursor position
Ctrl+W	Close the current document
Ctrl+X	Cut selected text to clipboard
Ctrl+Y	Redo latest undo
Ctrl+Z	Undo latest changes
Ctrl+Backspace	Delete the word before the cursor
Ctrl+Del	Delete the word after the cursor
Ctrl+F7	Toggle spell checking
Ctrl+Return	Open the tag or reference under the cursor in the viewer
Ctrl+Shift+A	Select all text in the current paragraph
Ctrl+Shift+I	Import text to the current document from a text file

11.3.4 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 `Ctrl+K`, followed by a key or key combination for the inserted content.

Shortcut	Description
`Ctrl+K, Space`	Insert a non-breaking space
`Ctrl+K, _`	Insert a long dash (em dash)
`Ctrl+K, .`	Insert an ellipsis
`Ctrl+K, '`	Insert a modifier apostrophe
`Ctrl+K, *`	Insert a list bullet
`Ctrl+K, %`	Insert a per mille symbol
`Ctrl+K, ~`	Insert a figure dash (same width as a number)
`Ctrl+K, -`	Insert a short dash (en dash)
`Ctrl+K, 1`	Insert a left single quote
`Ctrl+K, 2`	Insert a right single quote
`Ctrl+K, 3`	Insert a left double quote
`Ctrl+K, 4`	Insert a right double quote
`Ctrl+K, C`	Insert a `@char` keyword
`Ctrl+K, E`	Insert an `@entity` keyword
`Ctrl+K, F`	Insert a `@focus` keyword
`Ctrl+K, G`	Insert a `@tag` keyword
`Ctrl+K, H`	Insert a short description comment
`Ctrl+K, L`	Insert a `@location` keyword
`Ctrl+K, O`	Insert an `@object` keyword
`Ctrl+K, P`	Insert a `@plot` keyword
`Ctrl+K, S`	Insert a synopsis comment
`Ctrl+K, T`	Insert a `@time` keyword
`Ctrl+K, V`	Insert a `@pov` keyword
`Ctrl+K, X`	Insert a `@custom` keyword
`Ctrl+K, Ctrl+Space`	Insert a thin non-breaking space
`Ctrl+K, Ctrl+_`	Insert a horizontal bar (quotation dash)
`Ctrl+K, Ctrl+'`	Insert a prime
`Ctrl+K, Ctrl+"`	Insert a double prime
`Ctrl+K, Ctrl+*`	Insert a flower mark (alternative bullet)
`Ctrl+K, Ctrl+-`	Insert a hyphen bullet (alternative bullet)
`Ctrl+K, Ctrl+D`	Insert a division sign
`Ctrl+K, Ctrl+O`	Insert a degree symbol
`Ctrl+K, Ctrl+X`	Insert a times sign
`Ctrl+K, Shift+Space`	Insert a thin space

11.4 Document Viewer Shortcuts

Shortcut	Description
`Alt+Left`	Move backward in the view history
`Alt+Right`	Move forward in the view history
`Ctrl+C`	Copy selected text to clipboard
`Ctrl+Shift+A`	Select all text in the current paragraph
`Ctrl+Shift+R`	Close the document viewer

CHAPTER

TWELVE

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.

12.1 Special Notes on Symbols

This section contains additional notes on the available special symbols.

12.1.1 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 `Ctrl+Z`, 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.

12.1.2 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.

12.1.3 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 rendered 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.

12.1.4 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 rendered 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 Manuscript* 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 syntax highlighting.

12.1.5 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

THIRTEEN

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 in this chapter, 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.

13.1 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.

13.2 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.

13.3 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 eight to two. 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 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*.

13.4 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.

13.5 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.

13.6 Format 1.0 Changes

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

CHAPTER

FOURTEEN

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 bare bone project suited to your needs.

A novelWriter project requires a dedicated folder for storing its files on the local file system. If you're interested in the details, you can have a look at the chapter *How Data is Stored*.

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.

14.1 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.

The root folder system is closely connected to how the Tags and References system works. For more details, see the *Tags and References* chapter.

Novel
: This is the root folder type for 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 type 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 type. These are especially important if you want to use the *Outline View* to see which character appears where, 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 type of 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 character present.

Locations
: The locations folder type 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 folder type can be used to track this. Tags in this type of folder can be references using the `@time` keyword.

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

Custom
: The custom root folder type can be used for tracking anything else not covered by the above options. Tags in this folder type 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 *How to Use References*.

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

New in version 2.0: As of version 2.0, you can make multiple root folders of each kind to split up your project.

14.1.1 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 `Ctrl+Shift+Del`. Root folders can only be deleted when they are empty.

14.1.2 Archived Documents

If you don't want to delete a document, or put it in the *Trash* folder where it may be deleted accidentally, but still want it out of your main project tree, you can create an *Archive* root folder and move it there.

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

14.1.3 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.

The discovered files are scanned for metadata that give clues as to where the document may previously have been located in the project. The project loading routine 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 type. If it cannot figure out which root folder is correct, the document will be added to the *Novel* root folder. Finally, if the *Novel* folder is missing, one will be created.

If the title of the document can be recovered, the word "Recovered:" will be added as a prefix to indicate that it may need further attention. If the title cannot be determined, the document will be named after its internal key, which is a string of characters and numbers.

14.1.4 Project Lockfile

To prevent lost documents caused by file conflicts when novelWriter projects are synchronised via 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 given the option to ignore this warning, and continue opening the project at your own risk.

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

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

14.1.5 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 a project, like for instance when you create a manuscript from it, these folders are ignored. Only the order of the documents themselves matter.

The folders are there purely as a way for you 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.

New in version 2.0: As of version 2.0 it is possible to 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.

14.2 Project Documents

New documents can be created from the toolbar in the *Project Tree*, or by pressing `Ctrl+N`. 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 tree. The *Transform* submenu includes options for converting, splitting, or merging items. See *Splitting and Merging Documents* for more details on the latter two.

14.2.1 Word Counts

A character, word and paragraph count is maintained for each document, as well as for each section of a document following a *heading*. 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 trees.

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 *project index*. If the counts seem wrong, a full project word recount can be initiated by rebuilding the project's index. Either from the *Tools* menu, or by pressing `F9`.

14.3 Project Settings

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

14.3.1 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 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 language is set to, you can override the default setting here. You can also override the automatic backup setting. The project language can also be changed from the *Tools* menu.

14.3.2 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 labels are there purely for your 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.

14.3.3 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 is 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.

14.4 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 name and path of the backups. Without it, the backup will not run at all, but it will produce a warning message.

14.5 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. For more details about the log file, see *How Data is Stored*.

A tool to view the content of the log 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 or Excel.

New in version 1.2: 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

FIFTEEN

NOVEL STRUCTURE

This chapter covers the structure of a novel project.

There are two different types of documents in a project, *Novel Documents* and *Project Notes*. Novel documents can only live in a *Novel* type root folder. You can also move them to *Archive* and *Trash* of course.

The *Project Tree* can distinguish between the different header levels of the novel documents using coloured icons, and optionally add emphasis on the label, set in *Preferences*.

15.1 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 headings 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 markup 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 View* as well as in the *Novel Tree*. Each heading also starts a new region where new Tags and References can be defined. See *Tags and References* for more details.

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 `#!` covered 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 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 *Manuscript 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.

15.1.1 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*.

15.1.2 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 an `!` 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 *Manuscript Build* 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.

CHAPTER

SIXTEEN

TAGS AND REFERENCES

In novelWriter there are no forms or tables to fill in to define the characters, locations and other elements of your story. Instead, you can mark your *project notes* as representing these story elements by creating a *tag*. Whenever you want to link a piece of your story to a note defining a story element, like a character, you create a *reference* back to that tag. You can also cross-link your project notes in the same way.

This is perhaps one of the features that makes novelWriter different from other, similar applications. It is therefore not always obvious to new users how this is supposed to work, so this chapter hopes to explains in more detail how to use the tags and references system.

Tip: If you find the Tags and Reference system difficult to follow just from reading this chapter, you can create a new project in novelWriter and select to "Fill the project with example files" in the *New Project Wizard*. The example project contains several examples of tags and references.

16.1 Metadata in novelWriter

The structure of your novelWriter project is inferred from the *headings* within the documents, not the documents themselves. See *Importance of Headings* for more details. Therefore, metadata is also associated with headings, and not the documents directly.

If you split your project into separate documents for each scene, this distinction may not matter. However, there are several benefits to using documents at a larger structural scale when starting your project. For instance, it may make more sense to define all your scenes, and even chapters, in a single document at first, or perhaps a document per act. You can later split these documents up using the document split feature. See *Splitting and Merging Documents* for more details.

The implication here is that you can treat each heading as an independent element of your notes that can be referenced somewhere else. In order to make it possible to reference a header section, you need to assign it a tag.

16.2 How to Use Tags

A "tag" in novelWriter is a word or phrase that you define as belonging to a heading. Tags are set by using the `@tags` *keyword*.

The general format of a tag is `@tag: tagName`.

The full format of a tag is `@tag: tagName | displayName`.

`tagName` (Required)
: This is a unique identifier of your choosing. It is the value you use later for making references back to this document, or section of the document.

`displayName` (Optional)
: This is an optional display name used for the tag. When you build your manuscript, you can for instance insert the point of view character name into chapter headings. By default, the `tagName` value is used in headings, but if you use a shortened format internally in your project, you can use this to specify a more suitable format for your manuscript.

You can only set *one* tag per heading, and the tag has to be unique across *all* documents in the project.

After a tag has been defined, it can be referenced in novel documents, or cross-referenced in other notes. Tags will also show up in the *Outline View* and in the back-reference panel when a document is opened in the viewer.

The syntax highlighter will indicate to you 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 novelWriter uses. The rest of the document content is there for you to use in whatever way you 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 *Manuscript Build* tool.

New in version 2.2: Tags are no longer case sensitive. The tags are by default displayed with the capitalisation you use when defining the tag, but you don't have to use the same capitalisation when referencing it later.

New in version 2.3: Tags can have an optional display name for manuscript builds.

Example of a heading with a tag for a character of the story:

```
# Character: Jane Doe

@tag: Jane | Jane Doe

Some information about the character Jane Doe.
```

When this is done in a document in a *Root Folder* of type "Characters", the tag is automatically treated as an available character in your project with the value "Jane", and you will be able to reference it in any of your other documents using the reference keywords for characters. It will also show up in the Character tab in the Reference panel below the document viewer, and in the reference auto-completer menu in the editor when you fill in references. See *Viewing a Document* and *The References Auto-Completer*.

It is the root folder type that defines what category of story elements the tag is indexed under. See the *Project Roots* section for an overview of available root folder types. They are also covered in the next section.

16.3 How to Use References

Each heading of any level in your project can contain references to tags set in project notes. The references are gathered by the indexer and used to generate the *Outline View*, among other things.

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 reference 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.

Note: 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.

Tip: If you add a reference in the editor to a tag that doesn't yet exist, you can right-click it and select *Create Note for Tag*. This will generate a new project note automatically with the new tag defined. In order for this to be possible, a root folder for that category of references must already exist.

One note can also reference another note in the same way novel documents do. When the note is opened in the document viewer, the references become clickable links, making it easier to follow connections in the plot. You can follow links in the document editor by clicking them with the mouse while holding down the `Ctrl` key. Clicked links are always opened in the view panel.

Project notes don't show up in the *Outline View*, 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 *Manuscript Build* tool, the cross-references become clickable links in the exported HTML document as well.

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 ...
```

16.3.1 The References Auto-Completer

An auto-completer context menu will show up automatically in the document editor when you type the character `@` on a new line. It will first suggest tag or reference keywords for you to add, and after the `:` has been added, suggest references from the list of tags you have already defined.

You can use the auto-completer to add multiple references with a `,` between them, and even type new ones. New references can be created by right-clicking on them and selecting *Create Note for Tag* from the menu.

New in version 2.2.

CHAPTER

SEVENTEEN

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 *Manuscript Build* tool. You can activate it from the sidebar, the *Tools* menu, or by pressing `F5`.

New in version 2.1: This tool is new for version 2.1. A simpler tool was used for earlier versions. The simpler tool only allows you to define a single set of options for the build, but otherwise has much the same functionality.

17.1 The Manuscript Build Tool

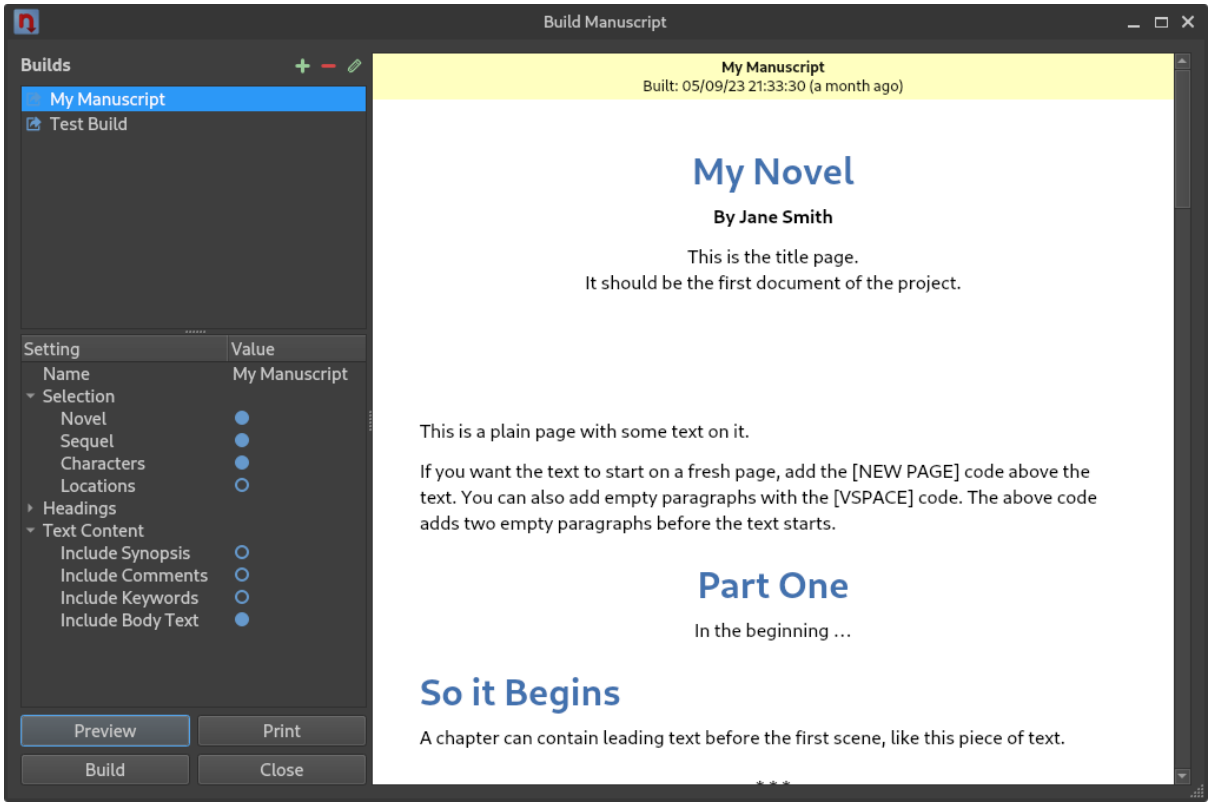


Fig. 1: The *Manuscript Build* tool main window.

The main window of the *Manuscript Build* tool contains a list of all the builds you have defined, a selection of settings, and a few buttons to generate preview, open the print dialog, or run the build to create a manuscript document.

17.2 Build Settings

Each build definition can be edited by opening it in the *Manuscript Build Settings* dialog, either by double-clicking or by selecting it and pressing the edit button in the toolbar.

Tip: You can keep the *Manuscript Build Settings* dialog open while testing the different options, and just hit the *Apply* button. You can test the result of your settings by pressing the *Preview* button in the main *Manuscript Build* window. When you're happy with the result, you can close the settings.

17.2.1 Document Selection

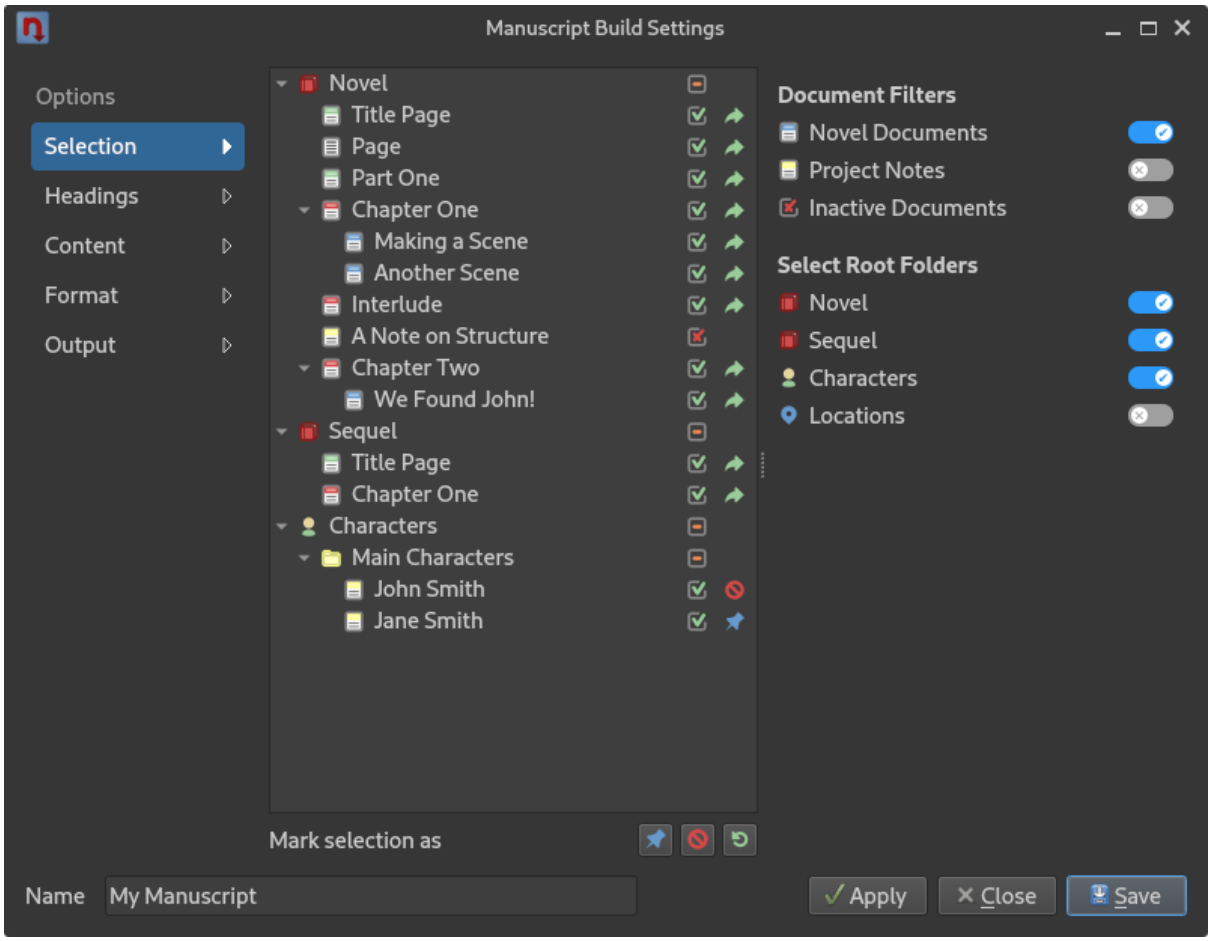


Fig. 2: The *Selections* page of the *Manuscript Build Settings* dialog.

The *Selections* page of the *Manuscript Build Settings* dialog allows you to fine tune which documents are included in the build. They are indicated by a green arrow icon in the last column. On the right you have some filter options for selecting content of a specific type, and a set of switches for which root folders to include.

You can override the result of these filters by marking one or more documents and selecting to explicitly include or exclude them by using the buttons below the tree view. The last button can be used to reset the override and return control to the filter settings.

In the figure, the green arrow icon and the blue pin icon indicates which documents are included, and the red forbidden icon indicates that a document is explicitly excluded.

17.2.2 Formatting Headings

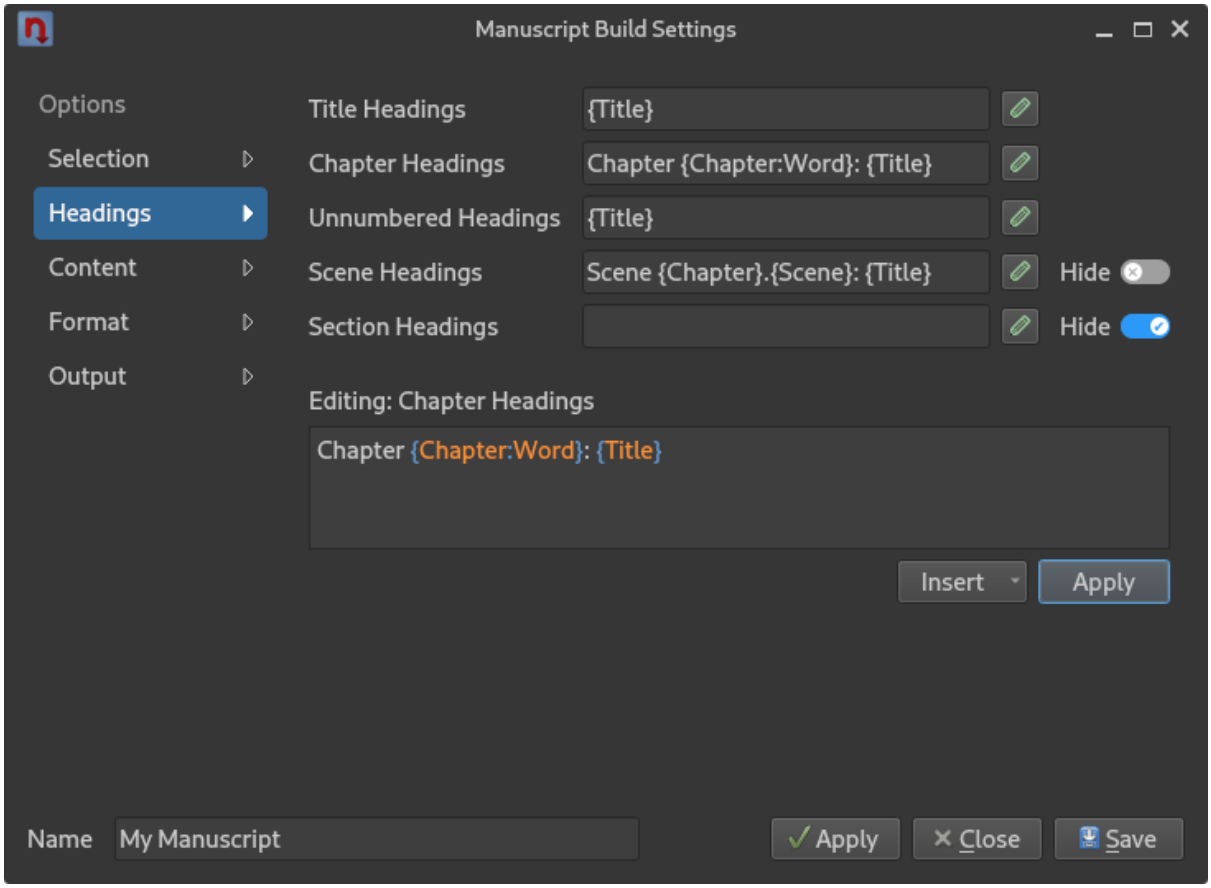


Fig. 3: The *Headings* page of the *Manuscript Build Settings* dialog.

The *Headings* page of the *Manuscript Build Settings* dialog allows you to set how the headings in your *Novel Documents* are formatted. By default, the title is just copied as-is, indicated by the `{Title}` format. You can change this to for instance add chapter numbers and scene numbers like shown int he figure above.

Clicking the edit button next to a format will copy the formatting string into the edit box where it can be modified, and where a syntax highlighter will help indicate which parts are automatically generated by the build tool. The *Insert* button is a dropdown list of these formats, and selecting one will insert it at the position of the cursor.

Any text you add that isn't highlighted in colours will remain in your formatted titles. `{Title}` will always be replaced by the text in the heading from your documents.

You can preview the result of these format strings by clicking *Apply*, and then clicking *Preview* in the *Manuscript Build* tool main window.

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. You can also switch on the *Hide* setting, which will ignore them completely. That is, there won't even be an extra gap inserted.

Alternatively, if you want a separator text between them, like the common * * *, you can enter the desired separator text as the format. If the format is any piece of static text, it will always be treated as a separator.

17.2.3 Output Settings

The *Content*, *Format* and *Output* pages of the *Manuscript Build Settings* dialog control a number of other settings for the output. Some of these only apply to specific output formats, which is indicated by the section headings on the settings pages.

17.3 Building Manuscript Documents

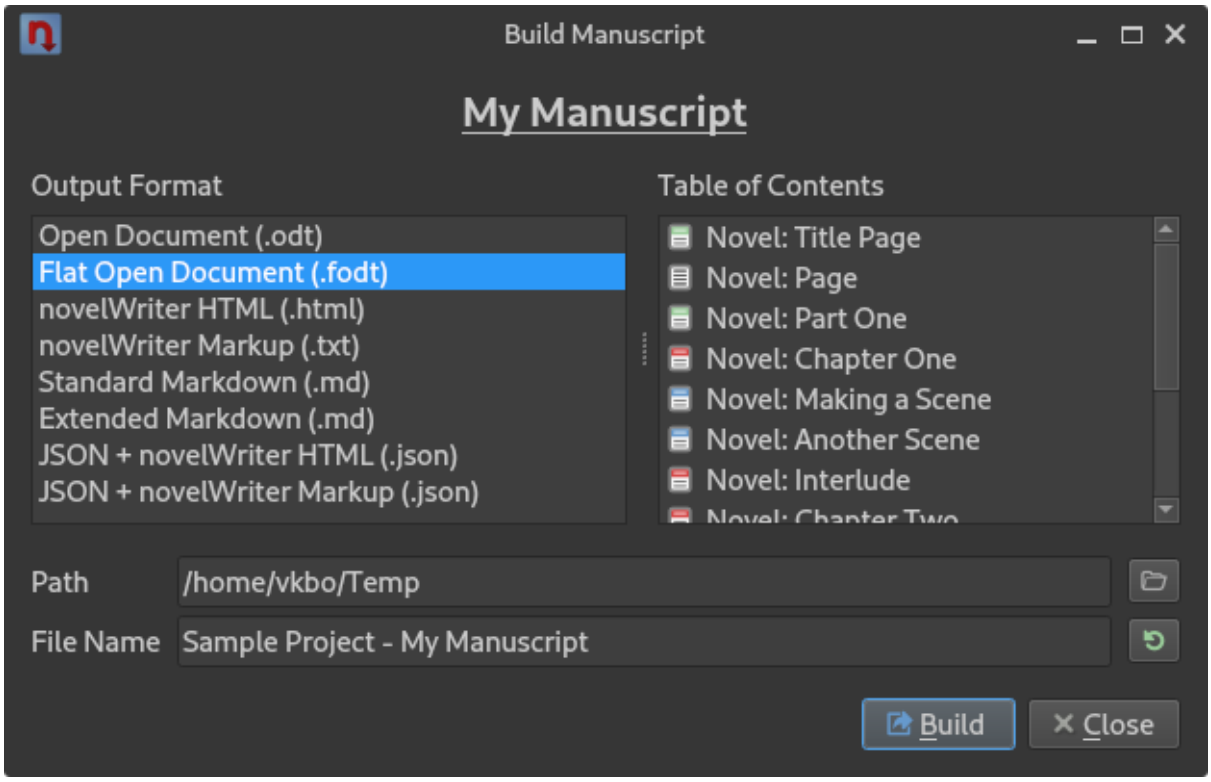


Fig. 4: The *Manuscript Build* dialog used for writing the actual manuscript documents.

When you press the *Build* button on the *Build Manuscript* tool main window, a special file dialog opens up. This is where you pick your desired output format and where to write the file.

On the left side of the dialog is a list of all the available file formats, and on the right, a list of the documents which are included based on the build definition you selected. You can choose an output path, and set a base file name as well. The file extension will be added automatically.

To generate the manuscript document, press the *Build* button. A small progress bar will show the build progress, but for small projects it may pass very fast.

17.3.1 File Formats

Currently, four document 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 only 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 Markup
: This is simply a concatenation of the project documents selected by the filters into a `.txt` file. 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/Extended Markdown
: The Markdown format comes in both Standard and Extended flavour. The *only* difference in terms of novelWriter functionality is the support for strikethrough text, which is not supported by the Standard flavour.

17.3.2 Additional Formats

In addition to the above document formats, the novelWriter HTML and Markup 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 used by the preview 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.

17.4 Print and PDF

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 select PDF as your output format from the printer icon on the print dialog.

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

CHAPTER

EIGHTEEN

FILE LOCATIONS

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

18.1 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.

18.2 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` value.

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

NINETEEN

HOW DATA IS STORED

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

19.1 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.

19.1.1 Main Project File

The project itself requires a dedicated folder for storing its files, where novelWriter will create its own "file system" where the project's 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 (except the index data), 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.

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 timestamp is set in the meta section on line 2, and various meta data entries incremented, on each save.

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

19.2 Project Documents

All the project documents are saved in a subfolder of 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 in the editor, 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 you name 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. 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 the file is lost from the main 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.

19.2.1 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 replaces it. This ensures that the previously saved data is only replaced when the new data has been successfully saved to the storage medium.

19.3 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. All files in this folder are JSON or JSON Lines files, although some other files may remain from earlier versions of novelWriter as they haven't all been JSON files in the past.

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.

19.3.1 The Project Index

Between writing sessions, the project index is saved in a JSON file in `meta/index.json`. This file is not critical. If it is lost, it can be completely 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.

19.3.2 Build Definitions

The build definitions from the *Manuscript Build* tool are kept in the `meta/builds.json` file. If this file is lost, all custom build definitions are lost too.

19.3.3 Cached GUI Options

A file named `meta/options.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.

19.3.4 Custom Word List

A file named `meta/userdict.json` 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.

19.3.5 Session Stats

The writing progress is saved in the `meta/sessions.jsonl` 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.

Each session is recorded as a JSON object on a single line of the file. Each session record is appended tot he file.

CHAPTER

TWENTY

RUNNING FROM SOURCE

This chapter 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 on many systems the command `python3` may be needed instead. Likewise, `pip` may need to be replaced with `pip3`.

Most of the custom commands for building packages of novelWriter, or building assets, are contained in the `pkgutils.py` script in the root of the source code. You can list the available commands by running:

```
python pkgutils.py help
```

20.1 Dependencies

novelWriter has been designed to rely on as few dependencies as possible. Only the Python wrapper for the Qt GUI libraries is required. The package for spell checking is optional, but recommended. Everything else is handled with standard Python libraries.

The following Python packages are needed to run all features of novelWriter:

- `PyQt5` – needed for connecting with the Qt5 libraries.
- `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.

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
```

Note: On Linux distros, the Qt library is usually split up into multiple packages. In some cases, secondary dependencies may not be installed automatically. For novelWriter, the library files for renderring

the SVG icons may be left out and needs to be installed manually. This is the case on for instance Arch Linux.

20.2 Build and Install from Source

If you want to install novelWriter directly from the source available on GitHub, you must first build the package using the Python Packaging Authority's build tool. It can be installed with:

```
pip install build
```

On Debian-based systems the tool can also be installed with:

```
sudo apt install python3-build
```

With the tool installed, run the following command from the root of the novelWriter source code:

```
python -m build --wheel
```

This should generate a `.whl` file in the `dist/` folder at your current location. The wheel file can then be installed on your system. Here with example version number 2.0.7, but yours may be different:

```
pip install --user dist/novelWriter-2.0.7-py3-none-any.whl
```

20.3 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 pkgutils.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.

20.4 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 the `pkgutils.py` script by running:

```
python pkgutils.py sample
```

20.5 Building the Documentation

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

```
pip install -r docs/source/requirements.txt
```

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 `pkgutils.py` script:

```
python pkgutils.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

TWENTYONE

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.

21.1 Dependencies

The dependencies for running the tests can be installed with:

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

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

21.2 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 `QT_QPA_PLATFORM=offscreen`:

```
export QT_QPA_PLATFORM=offscreen pytest
```

21.3 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:

```
export QT_QPA_PLATFORM=offscreen 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:

```
export QT_QPA_PLATFORM=offscreen 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`:

```
export QT_QPA_PLATFORM=offscreen 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:

```
export QT_QPA_PLATFORM=offscreen pytest -v --cov=novelwriter --cov-
↪report=html -k testGuiEditor
```

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

INDEX