novelWriter

User Guide

Release 2.8

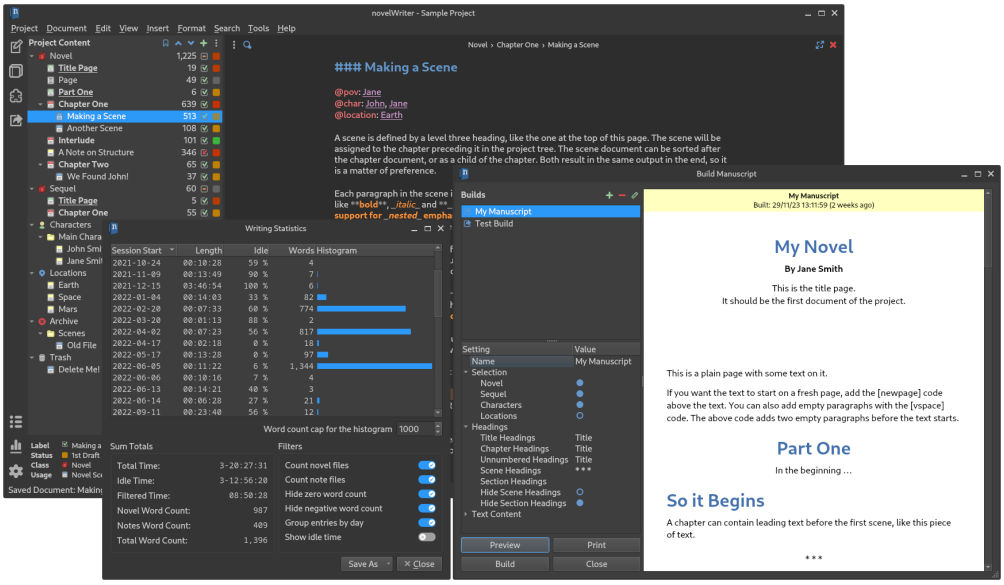
Veronica Berglyd Olsen

Sunday, 14 December 2025 at 20:27

CONTENTS

Release Version: 2.8
Updated: Sunday, 14 December 2025 at 20:27

novelWriter is an open source plain text editor designed for writing novels assembled from individual 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.



Useful Links

- Website: https://novelwriter.io
- Documentation: https://docs.novelwriter.io
- Public Releases: https://releases.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

INTRODUCTION

In a nutshell, novelWriter is a plain text editor that lets you organise one or more novels, and associated notes, as many smaller documents. You can at any time generate standard document formats from these plain text documents. Whether it is an outline of your story, a draft, a complete manuscript, or even a collection of your character notes or other notes.

1.1 Why Plain Text?

The idea is to let you be creative without having to deal with formatting while you are writing, or be distracted by it.

Of course, you probably need *some* form of minimal formatting for your text. At the very least you need emphasis. Most people are familiar with adding emphasis using `_underscores_` and `**asterisks**`. This formatting standard comes from Markdown and is supported by novelWriter. It also uses Markdown formatting for defining document headings, which is how you distinguish between chapters and scenes.

For those special cases where you need more complex formatting, a set of shortcodes are available. To make these codes easier to use, a dropdown button bar is available in the editor panel with standard format buttons. So don't worry. You don't have to learn any of these codes.

1.2 Adding Meta Data

In addition to the body text of your story, novelWriter allows you to enter some additional meta data into your text documents to indicate things like which characters are present in a chapter or scene, whose point of view we're seeing, what location the events take place in, and so on.

Since the editor is plain text, this is done on special lines of text starting with an `@` character. The editor will show an auto-complete menu to help you write these lines. We will talk more about this later.

You can also add your own author's comments in your text, without these comments becoming a part of the story itself. A comment line starts with a `%` character. There are different types of comments, and an auto-complete menu can help you here too. More about this later as well.

Limitations

Please keep in mind that novelWriter 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. It is also *not* a full-featured Markdown editor.

In addition, novelWriter is not intended as a tool for organising research for writing, and therefore lacks formatting features you may need for this purpose. The notes feature is mainly intended for character profiles and plot outlines. It is recommended to use a proper note-taking tool for research. This is anyway more practical as you may use the same research for multiple projects.

1.3 Contributing

This project relies on contributions to add new features. In particular, adding translations into other languages, or updating translations for existing languages, is particularly useful.

For code contributions, please read the official Contributing Guide.

For translations, please use the project page on Crowdin. For each language, there are two translation sets. One for the application itself, labelled "Main GUI". This is a fairly large dataset to translate. A smaller set called "Project Exports" is also available. The latter is what makes a language available in **Project Settings** in the application. It is a fairly small set to translate, and you can choose to translate only this set if you wish. If the language you wish to contribute to is not available in the list, you can message the maintainer on the Crowdin page to add new languages to the project.

CHAPTER

TWO

ORGANISING YOUR PROJECT

Your project is organised into a set of top level folders called “Root Folders”, which each have specific meaning in the project. Your project documents and notes are stored under these root folders. All the content of your project is available in the **Project Content** panel on the left side of the main window.

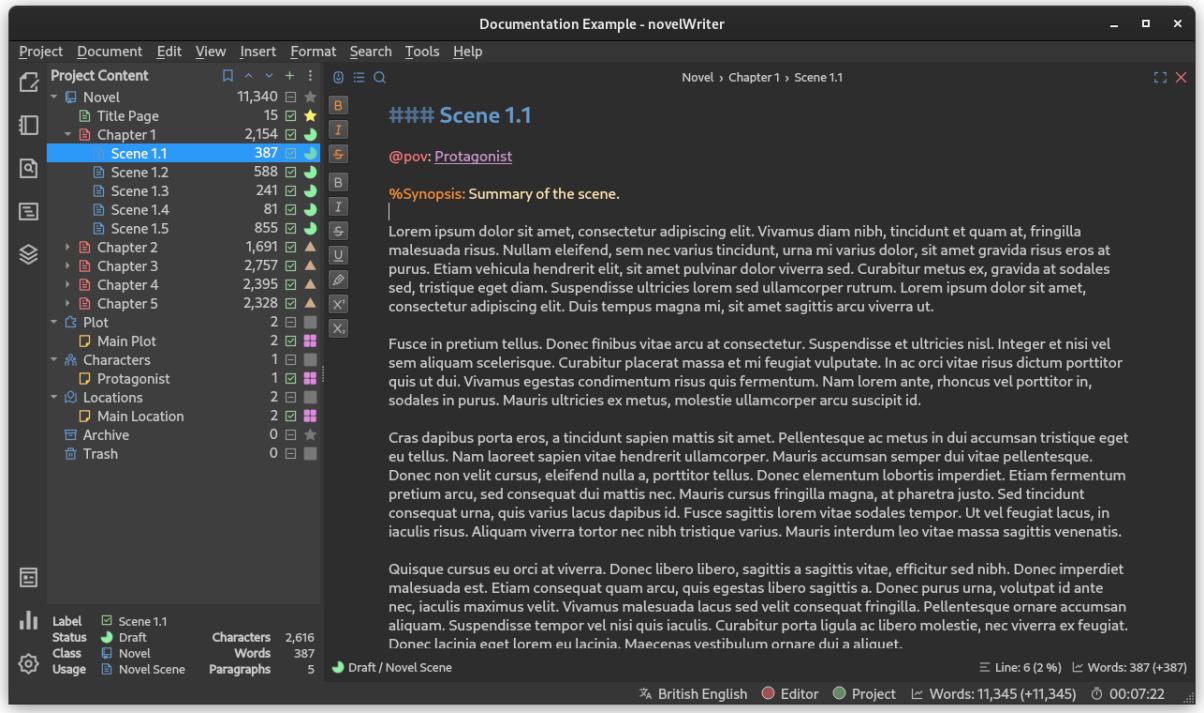


Fig. 1: The **Project Content** tree populated with example documents.

Each line in the project tree shows the name of each item, its word count (or alternatively character count), an icon for *Active and Inactive Documents*, and a custom icon for *Importance and Status* of each item. These latter two are covered later in this section.

You can add, view and edit documents in the project tree by right-clicking on them. Some features are also located in the buttons along the top, next to the **Project Content** label.

2.1 How Root Folders Work

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. Each type of root folder has a distinct icon.

The documents that make up your story go into a root folder of type **Novel**. Your notes go into the other root folders. These other root folder types are separated into types depending on what kind of notes go into them. This is not only for organisation. It also matters to how you can reference these notes later. We will come back to this in the *Tags and References* section.

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.

The intended usage of each type of root folder is listed below. However, aside from the **Novel** folder, no restrictions are applied by novelWriter on what you put in them. You can use them however you want.

2.1.1 Root Folder Types

Novel (Story)
: This is where you put the documents that are part of your story. You can create multiple Novel folders if you wish, but various parts of the application assumes each Novel folder belongs to only one novel.

  The Novel folder is somewhat special in that it can contain documents for chapters, scenes and story partitions. How this is indicated is covered in the section *Chapters and Scenes*.

Plot (Notes)
: This is where you can keep notes and outlines of your story plots. Such notes can be particularly useful if you have outlines for sub plot. You can make references to these subplots from the scene documents, which makes it easier to track story progress.

Characters (Notes)
: Character notes go in this root folder type. For your main characters, you may want to make one document for each character. For smaller characters you can put multiple into the same document. In your chapters and scenes you can reference these character notes as point-of-view or focus characters.

Locations (Notes)
: The locations where your story takes place can be documented here. This, together with Plot and Characters are the key story elements to track, and to reference from your chapter and scene documents.

Timeline (Notes)
: If the story has multiple plot timelines or jumps in time within the same plot, this folder type can be used to track this.

Objects (Notes)
: Important objects in the story, for instance physical objects that change hands often, can be tracked here.

Entities (Notes)
: Does your plot have many powerful organisations or companies? Or other entities that are part of the plot? They can be organised here.

Custom (Notes)
: The custom root folder type can be used for tracking anything else not covered by the above options.

Templates
: Any document added under this root folder will be made available as template options when creating new documents. See *Document Templates* for more details.

Archive
: If you don't want to delete a document, or put it in the **Trash** folder where it may be deleted, but still want it out of your main project, you can put it in this folder. The contents of the document will be ignored by the scanner that looks for tags, and it will be ignored in any outline view and in your manuscript.

Trash
: This folder behaves like you expect. Anything dropped in here can be deleted permanently from the project, and the content doesn't show up anywhere else in novelWriter.

The root folder types are closely tied to the tags and reference system. Each folder type for novel and notes corresponds to one or more categories of tags that can be used to reference the content in them. See *Tags and References* for more details.

Tip: The root folders have standard names, but you can rename them to whatever you want.

2.2 Regular Folders

You can add regular folders anywhere you want in the project. 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.

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.

2.3 Documents

You can add documents anywhere you want in your project structure. You can even add documents as child items of other documents, just as if they were folders. This makes it easy to associate a set of scenes with their chapter. You can also do this in your notes, where you for instance may have a hierarchy of your locations.

The name on a document in the project tree is not linked to any headings in the document text. Think of the document name as a file name. You can rename a document, or any other item in the project, at any time.

Documents come in two types:

Novel Documents
: These are the documents that make up your story or novel. They can only be added under a root folder of type **Novel**. You can technically also add them under **Archive**. See *Chapters and Scenes* for more details on how these documents are handled by novelWriter.

Project Notes
: These are the documents where you keep your notes. You can add them anywhere in your project, including under **Novel** type folders. If you do add them there, they are not treated as a part of the story by default.

You can convert between the two types of documents where both types are allowed. You can also convert folders into documents, which may sometimes be convenient too.

Another convenient feature is that documents can be split into sub-documents by its headings, or multiple documents merged into one. This is particularly useful if you start out with larger structural documents, like one containing all chapters and scenes in an act, and then split those when you start writing. See *Split and Merge Documents* for more details.

2.3.1 Document Templates

If you wish to create template documents to be used when creating new documents, like for instance a character note template, you can add a **Templates** root folder to your project. Any document added to this root folder will show up in the **Add Item** menu in the project tree toolbar. When selected, a new document is created with its content copied from the chosen template.

If the first line of the template file is a title line, the title text will be replaced with the text of the document label when it is first created.

New in version 2.3.

2.4 Active and Inactive Documents

A document can be set as "Active" or "Inactive", which alters the icon in the third column of the project tree. These are mostly intended for your convenience as they will indicate whether the document is meant to be included in the manuscript or not. You can think of an inactive status as a whole-document out-take. It allows you to take it out without moving it to **Archive**.

Inactive documents are by default excluded from your manuscript, but you can override this if you wish. See *Document Selection* for more details.

2.5 Importance and Status

Each document or folder in your project can have either a "Status" or "Importance" label set. These are labels and icons that you control and define yourself, and novelWriter doesn't use them for anything. You can modify these labels in **Project Settings**. See *Status and Importance* for more details.

The "Status" labels are intended to tag a novel document as for instance a draft or as completed, and the "Importance" labels are 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" label depends on which root folder it lives in. If it's in a **Novel** type folder, it uses the "Status" label, otherwise it uses an "Importance" label.

CHAPTER

THREE

CHAPTERS AND SCENES

Since novelWriter uses a plain text format, the structure of your novel must follow a certain set of simple rules. For documents in a **Novel** type root folder, it is the heading that determines if the document is a chapter or a scene.

The formatting of headings is based on Markdown. A heading is indicated by a line starting with one or more # characters. It accepts up to four of these. You can use multiple headings in the same document, but it is the first heading that determines which icon and information is displayed in the project tree.

Note: You can use the same heading levels for your notes in the other root folders, but they aren't treated as chapters or scenes, so there you are free to use them as you want.

3.1 Heading Levels

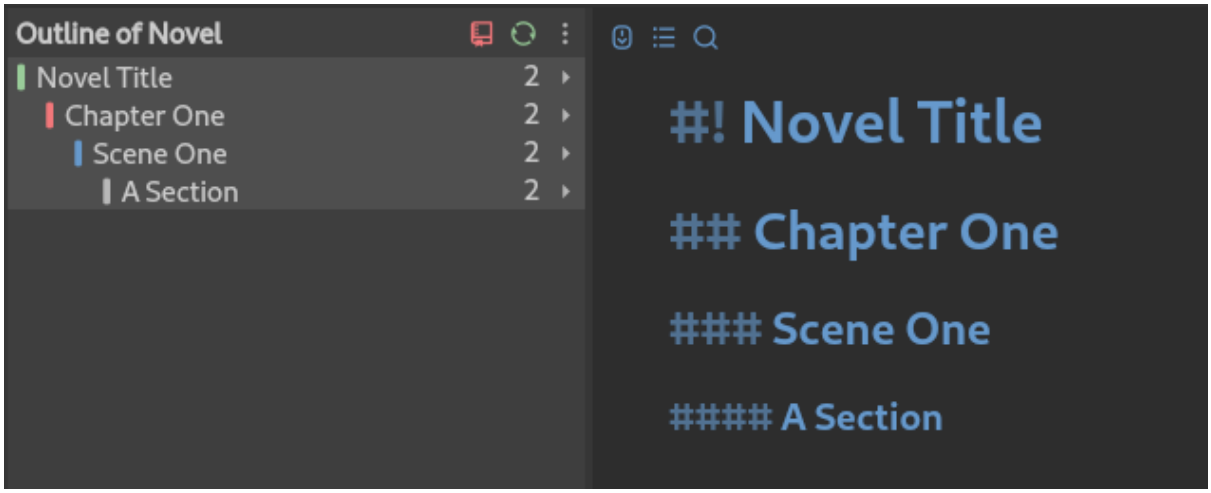


Fig. 1: An illustration of how heading levels correspond to the novel structure.

Four levels of headings are understood for novel documents. You can pick and choose from these as you want, but if your story has chapters, you should use these headings to indicate them. If you also add scene headings, you have better control of how your scene separators are formatted in your manuscript. The chapter and scenes headings are also displayed in the *Novel View* and *Outline View*.

`# Title Text`
: This is a heading level one. This heading indicates the start of a new partition. Partitions are for when you want to split your story into "Part 1", "Part 2", etc. You can also choose to use them for

splitting the text up into acts, and then hide these headings in your manuscript so that they are not included in the output.

`## Chapter Title`
: This is a heading level two. This heading indicates the start of a new chapter. Chapter numbers can be inserted automatically when building the manuscript, so you don't have to do this in the title. See *Automatic Numbering* for more details.

`### Scene Title`
: This is a heading level three. This heading 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, but you must provide a minimal title.

`#### Section Title`
: This is a heading level four. This heading indicates the start of a new section. Section titles can be replaced by separators or ignored completely when building the manuscript. The meaning of a section is really whatever you want it to be. You can use it to split your scenes up into chunks, or into separate documents.

For headings level one through three, adding a `!` modifies the meaning of the heading. The alternative meaning of the heading is only relevant when you generate your manuscript, but you may want to keep the use cases in mind while writing.

`#! Title Text`
: This tells the **Manuscript Build** tool that the level one heading is intended to be used for the novel or notes folder's main title, like for instance the novel title on the cover page. When building the manuscript, this will use a different styling of the title, which you can modify independently from how partition titles are styled. See *The Title Page* for more details.

`##! Chapter Title`
: This tells the **Manuscript Build** tool to not assign a chapter number to this chapter title if automatic chapter numbers are enabled. Such titles are useful for prologues and epilogues for instance. See *Unnumbered Chapters* for more details.

`###! Scene Title`
: This is an alternative scene heading that can be formatted differently in the **Manuscript Build** tool. It is intended for separating "soft" and "hard" scene breaks. Aside from this, it behaves identically to a regular scene heading. See *Hard and Soft Scenes* for more details.

The formatting of these headings can be customised quite extensively in the *Manuscript Tool*, which is covered in a separate part of the documentation.

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

Page breaks can be automatically added before titles, partition, chapter and scene headings from the **Manuscript Build** tool when you build your project to a format that supports page breaks. If you want page breaks in other places, you have to specify them manually. See *Vertical Space and Page Breaks* for more details.

CHAPTER

FOUR

BASIC FORMATTING

The basic text formatting syntax of novelWriter is based on Markdown. It is only a subset of the Markdown syntax though. Lists, images, and links are not supported.

That said, URLs in the text should automatically be highlighted and become clickable. However, only URLs starting with "http" or "https" are recognised. In the editor, you must hold down the `Ctrl` key when clicking a URL to follow it.

4.1 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.

It is important that you actually follow this rule. You should not, for instance, mimic indented paragraphs manually in the editor. This, and a lot of other formatting options that can be applied to text paragraphs in the *Manuscript Tool* depend on paragraphs being separated by blank lines.

Correct

```
### Scene

This is a text paragraph.

This is another text paragraph.
```

Incorrect

```
### Scene

This is a text paragraph.
    This is meant to be another text paragraph.
```

If you do as shown in the "Incorrect" example, novelWriter will understand this as a single paragraph with two lines.

4.2 Text Emphasis with Markdown

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

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

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

`~~text~~`
: Strike through text.

In Markdown guides it is often recommended to differentiate between strong emphasis 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 comments and paragraphs. Headings and meta data tags don't allow for formatting, and any formatting markup will be displayed 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.

CHAPTER

FIVE

COMMENTS AND NOTES

You can add comments to your text that are not a part of the story. Regular comments are intended for you to add notes to yourself inside the text, which may be useful when you revise your drafts. However, there are several types of comments you can use.

This section covers the basic comment types. There are a couple of advanced features that use comment syntax too, but they are covered later.

5.1 Plain Comments

A plain comment is a line or paragraph that starts with the character % as its first character. You can put them wherever you like in your documents, and you can choose to include or exclude them from your manuscript.

For the most part, novelWriter completely ignores these comments. They are not included in your word or character counts either, and are only displayed in the document viewer panel if you enable them.

Example

```
### Scene

A regular text paragraph in the scene.

% A comment you've added for your own notes.

Another regular text paragraph in the scene.
```

5.2 Synopsis or Description Comments

A special kind of comments are **Synopsis** and **Short Description** comments. They are different from plain comments in that they can be displayed alongside other information about a scene or a character or other story element described in a note. As with plain comments, they can be included in your manuscript, but they are formatted differently than plain comments.

Note: A summary or description comment can be used once, and only once, for each heading as they are considered a description of the content of the text under that heading. If you add two such comments under the same heading, the last one will be used.

5.2.1 Synopsis

A **Synopsis** comment is intended for adding a summary of your chapters and scenes.

Example

```
### Scene

%Synopsis: A summary of the content of the scene.

The actual scene text.
```

5.2.2 Short Description

A **Short Description** comment behaves exactly the same as a synopsis comment, but is intended as a description of a story element, like a character.

Example

```
# Characters

## Darth Vader

%Short: A Sith Lord that used to be a Jedi.

Your text about the character.

## Luke Skywalker

%Short: A Jedi. The son of Darth Vader.

Your text about the character.
```

Note: The `%Synopsis:` and `%Short:` comment prefixes are interchangeable, but when you include them in the manuscript, they are labelled based on the prefix, so the latter may make more sense for a Character note than the former.

5.3 Footnote Comments

Footnotes are added with a shortcode, paired with a matching comment for the actual footnote text. The matching is done with a key that links the two. If you insert a footnote from the **Insert** menu, a unique key is generated for you. Shortcodes in general are covered in more detail in *Formatting with Shortcodes*.

The insert footnote feature will add the footnote shortcode marker at the position of your cursor in the editor panel, and create the associated footnote comment right after the paragraph. It will then move the cursor there so you can immediately start typing the footnote text.

The footnote comment can be anywhere in the document, so if you wish to move them to, say, the bottom of the text, you are free to do so.

Footnote keys are only required to be unique within a document, so if you copy, move or merge text, you must make sure the keys are not duplicated. If you use the automatically generated keys from the **Insert** menu, they are unique among all indexed documents. They are not guaranteed to be unique against footnotes in the **Archive** or **Trash** folder though, but the chance of accidentally generating the same key twice in a project is relatively small.

Example

```
### Scene

This is a text paragraph with a footnote[footnote:fn1] in the middle.

%Footnote.fn1: This is the text of the footnote.
```

New in version 2.5.

5.4 Ignored Text

If you want to completely ignore some of the text in your documents, but are not ready to delete it, you can add `%~` before the text paragraph or line. This will cause novelWriter to skip the text entirely when generating previews or building manuscripts.

This is a better way of removing text than converting them to regular comments, as you may want to include regular comments in your previews or draft manuscript.

You can toggle the ignored text feature on and off for a paragraph by pressing `Ctrl+Shift+D` on your keyboard with your cursor somewhere in the paragraph.

Example

```
### Scene

%~ This text is ignored.

This text is a regular paragraph.
```

CHAPTER

SIX

TAGS AND REFERENCES

One of the core features of novelWriter is its **Tags and References** system. 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.

In novelWriter there are no forms or tables to fill in to define characters, locations or other elements of your story. Instead, you create documents in one of the root folders for notes. Within these documents you can set **tags**, like for instance for your main character. If you then want to annotate a scene with this character as its point-of-view, you create a **reference** to the tag.

Tip: If you find the Tags and Reference system difficult to follow just from reading this chapter, you can create a new project in the **Welcome** dialog's New Project form and select "Create an example project" from the "Pre-fill project" option. The example project contains several examples of tags and references.

6.1 How to Use Tags

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

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

The basic format of a tag is `@tag: TagName`.

An alternative format of a tag is `@tag: TagName | Display Name`.

`tagName` (Required)
: This is a unique identifier of your choosing. It is the value you use later for making references back to the heading in the document. The tag must be unique.

`Display Name` (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 directly into chapter titles. By default, the `tagName` value is used in such headings, but if you use a shortened format internally in your project, you can use the display name to specify a more suitable format for your chapter title.

Note: 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 **References** panel under the document viewer when a document is open in the viewer. See *Novel Outline View* and *Document References* for more details.

The editor will indicate to you that the keyword is correctly used and that the tag is allowed, that is, the tag is unique, by adding a colour highlighting to it. An invalid tag should have a wiggly line under it, and will not receive the colour that valid tags do.

The tag is the only part of notes that novelWriter uses. The rest of the document content is there for you to use in whatever way you wish.

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.

New in version 2.6: You can now add tags also to Novel Documents. These can be used for cross-referencing between chapters and scenes, and also from notes if desired.

Example

Example of a note document for a character with a tag set:

```
# 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". You will then be able to reference "Jane" in any of your other documents using the reference keywords for characters.

The character "Jane" will also show up in the **Character** tab in the **Reference** panel below the document viewer.

Note: It is the root folder type that defines what category of story elements the tag is indexed under. See *How Root Folders Work* for more details.

6.2 How to Use References

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

References are set with a special keyword, with 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 the point-of-view character. The target must be a note tag in a **Character** type root folder.

`@char`
: For 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 an **Entities** type root folder.

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

`@mention`
: For anything, anyone or anyplace mentioned, but not present in the current section. It is intended for those cases where you reveal details about a character or place in a scene without otherwise being a part of it. This can be useful when checking for consistency later. Any tag in any root note folder can be listed under `@mention`.

`@story`
: This is used when referencing a Novel Document, like a scene or chapter, from somewhere else in your project. It is possible to also set tags in documents in a **Novel** type folder, and this is the keyword you use to reference those.

When tags and references are used correctly, it will be indicated by highlight colours in the editor.

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 note automatically in the correct type of root folder, with the new tag defined.

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.

Your 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

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 ...
```

6.2.1 Auto-Completion in the Editor

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. Notes for 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

SEVEN

ALIGNMENT AND INDENTATION

The Markdown standard doesn't have commands for aligning text, so novelWriter adds its own syntax for this. It also has syntax for indentation, which is similar to Markdown block quotes.

7.1 Paragraph Alignment and Indentation

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

Example

Table 1: Text Alignment and Indentation

Syntax	Description
`>> Right aligned text`	The text paragraph is right-aligned.
`Left aligned text <<`	The text paragraph is left-aligned.
`>> Centred text <<`	The text paragraph is centred.
`> Left indented text`	The text has an increased left margin.
`Right indented text <`	The text has an increased right margin.
`> Left/right indented text <`	The text has 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 **Manuscript Build** tool as long as the format supports paragraph alignment.

7.2 Alignment with Line Breaks

If you have line breaks in the paragraph, the markers for all the lines are combined and used for the entire paragraph. For the following text, all lines will be centred:

Example

```
>> I am the very model of a modern Major-General <<
I've information vegetable, animal, and mineral
I know the kings of England, and I quote the fights historical
From Marathon to Waterloo, in order categorical
```

If you have multiple conflicting alignments on a paragraph, only one is applied. The order of precedence is:

1. Left alignment
2. Right alignment
3. Centred text
4. Justified text

Note: It is strongly recommended that you keep the **Preserve Hard Line Breaks** setting enabled in your manuscript build settings. This setting assumes all single line breaks in your text are intended. Turning this off makes adding line breaks more complicated, but it is still possible. See *Alignment with Forced Line Breaks*.

7.3 Alignment with First Line Indent

If you have first line indent enabled in your manuscript build settings, you probably want to disable it for text in verses. Adding any alignment tags on a paragraph will cause the first line indent to be switched off for that paragraph.

Example

The following text will always be aligned against the left margin:

```
I am the very model of a modern Major-General <<
I've information vegetable, animal, and mineral
I know the kings of England, and I quote the fights historical
From Marathon to Waterloo, in order categorical
```

7.4 Alignment with Forced Line Breaks

If you turn off **Preserve Hard Line Breaks** in your manuscript build settings, you can still force line breaks in paragraphs using the `[br]` shortcode. For clarity in the text, you can add a line break after it as well. It doesn't result in two line breaks.

Keep in mind that when the text is processed, the lines on either side of a `[br]` shortcode are combined, and any trailing hard line break is *ignored*. This means that when such a paragraph is processed, these line breaks count as the same line. This affects how alignment tags are handled. For instance, this text becomes centred instead of left aligned.

```
>> I am the very model of a modern Major-General[br]
I've information vegetable, animal, and mineral[br]
I know the kings of England, and I quote the fights historical[br]
From Marathon to Waterloo, in order categorical <<
```

Since this is understood as one line, this is the only way you can actually centre this paragraph.

> **Caution:** Due to this difference in how text with `[br]` tags are processed, it is generally better to stick with the **Preserve Hard Line Breaks** setting enabled. It ensures a better correspondence between what you see in the editor and what output you get.

See also *Forced Line Break*.

CHAPTER

EIGHT

ADVANCED FORMATTING

Standard Markdown formatting is somewhat limited, so novelWriter has some additional formatting codes for special use cases. These codes are all based on brackets, and some allow an additional value to be set after a colon.

This section covers all these formatting codes.

8.1 Formatting with Shortcodes

For basic formatting, like emphasis, you should use the standard Markdown style formatting tags descried in *Text Emphasis with Markdown* whenever possible.

For additional formatting options, you can use shortcodes. Shortcodes is a form of in-line codes that wrap the section of text to be formatted. Shortcodes can be nested to apply multiple formats to the same piece of text.

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 displayed as bold text.
`[i]text[/i]`	Text is displayed as italicised text.
`[s]text[/s]`	Text is displayed as strike through text.
`[u]text[/u]`	Text is displayed as underlined text.
`[m]text[/m]`	Text is displayed as highlighted text.
`[sup]text[/sup]`	Text is displayed as superscript text.
`[sub]text[/sub]`	Text is displayed as subscript text.
`[footnote:key]`	A reference to a *footnote comment*.

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 left-most icon button in the editor header.

Note: Shortcodes are not processed until you generate a preview or generate a manuscript document. So there is no highlighting of the text between the formatting markers. There is also no check that your

markers make sense. You must ensure that you have both the opening and closing formatting markers where you want them.

New in version 2.2.

8.1.1 Forced Line Break

Inserting `[br]` in the text will ensure a line break is always inserted in that place, even if you turn off **Preserve Hard Line Breaks** in your manuscript build settings.

You can add a manual line break after it too, for a better visual representation in the editor, but keep in mind that this line break is removed before the text is processed, so the text on either side of the `[br]` shortcode will be considered as belonging to the same line. This can affect how alignment is treated. See *Alignment with Forced Line Breaks* for more details.

8.2 Vertical Space and Page Breaks

You can apply page breaks to partition, chapter and scene headings for novel documents from the **Manuscript Build** tool. If you need to add a page break or additional vertical spacing in other places, there are special codes available for this purpose.

Adding more than one line break between paragraphs will **not** increase the space between those paragraphs when generating a manuscript document. To add additional space between paragraphs, add the text `[vspace]` on a line of its own, and the **Manuscript 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.

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

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]` code which has no text following 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
between them.

[new page]

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

8.3 Inserting Word Counts in the Text

The cover page of a manuscript normally has the word count stated on it. Any statistics value collected by novelWriter can be inserted into any document using a special shortcode. You can insert the code for any of the available statistics values from the **Insert** menu under **Word/Character Count**.

The value inserted is the actual count for your entire manuscript, so it is not populated until you run the **Manuscript Build** tool. Until then they will show up as "0" in the viewer panel.

The available codes are:

Table 2: Stats Shortcodes

Code	Description
`[field:allChars]`	Characters
`[field:textChars]`	Characters in Text
`[field:titleChars]`	Characters in Headings
`[field:paragraphCount]`	Paragraphs
`[field:titleCount]`	Headings
`[field:allWordChars]`	Characters, No Spaces
`[field:textWordChars]`	Characters in Text, No Spaces
`[field:titleWordChars]`	Characters in Headings, No Spaces
`[field:allWords]`	Words
`[field:textWords]`	Words in Text
`[field:titleWords]`	Words in Headings

Example

This is an example cover page. A similar page is automatically generated when you create a new project.

```
Jane Smith[br]
42 Main Street[br]
1234 Capital City <<

[vspace:5]

#! Example

>> **By Jane Smith** <<

>> Word Count: [field:textWords] <<
```

CHAPTER

NINE

FRONT AND BACK MATTER

Front and back matter documents are documents that go before and after your main story text. They can include pages like your cover page, content tables, prologues, epilogues, etc. These special pages and sections are supported to some extent by novelWriter.

9.1 The Title Page

It is recommended that you add a document at the very top of each **Novel** root folder with the novel title in it. You should modify the level 1 heading 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.

You can also add the author name and address above this if this is required by the manuscript format you use, and additional space added before the title.

Example

This is the title page novelWriter generates automatically for a new project as of version 2.6:

```
Jane Doe[br]
Address Line 1[br]
Address Line 2 <<

[vspace:5]

#! My Novel

>> **By Jane Doe** <<

>> Word Count: [field:textWords] <<
```

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.

The default title page inserts the word count for text only, but you can add other counts too. See *Inserting Word Counts in the Text* for more details.

9.2 Additional Pages

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 *Vertical Space and Page Breaks* for more details.

9.3 Unnumbered Chapters

If you use the automatic numbering feature for your chapters, but you want to keep some special chapters separate from this, you can add an `!` to the level 2 heading formatting code to tell the build tool to skip these chapters when adding numbers.

Unnumbered chapters are useful for prologue and epilogue chapters, and also for interlude chapters if you use those in your text. There is a separate formatting feature for such chapter titles in the **Manuscript Build** tool. See the *Building the Manuscript* page for more details.

Example

```
##! Unnumbered Chapter Title

Chapter Text
```

CHAPTER

TEN

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.

10.1 Your Project

How do I create a project from a template?

On the Welcome dialog's **Create New Project** form, you can select to "Prefill Project" from the content of a different project. This feature is most useful if you copy a project you have dedicated to be a template project. If you have a structure and settings you want to use for every new project, this is the best solution.

How do I merge multiple documents into one?

If you need to merge a selection 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 documents 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 rearrange them here of you wish. See *Split and Merge Documents* for more details.

How do I share status or importance labels between projects?

The status or importance labels you have defined in a project can be exported from **Project Settings** from the respective configuration tabs. You can then import these labels in another project.

How do I 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.

How do I distinguishing between 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 different scene headings for hard and soft scene breaks. The **Build Manuscript** tool will let you define a different format for scenes using the ### and ###! heading codes when you generate your manuscript. You can for instance add the common "* * *" for hard breaks and select to hide soft scene breaks, which will just insert an empty paragraph in their place. See *Build Settings* for more details.

New in version 2.4.

How can I set the language for my manuscript?

The language set for manuscript documents like Open Document and Word Document is taken from your project language setting in **Project Settings**. Only some languages are available here, but you can override the language information by setting the "Override Document Language" value in your Manuscript Build settings to a valid language code. The code uses the IETF BCP 47 language tag format.

Examples: `en` for English, `de` for German, etc.

If you want to help add new languages to novelWriter, see: *Contributing*.

New in version 2.8.

10.2 The Application

How can I increase the line height?

Unfortunately, in this case, novelWriter uses the plain text editor component of the underlying framework (called Qt) which doesn't support changing the line height. However, you can choose an editor font with larger leading (the spacing between the lines). A good font for this case is Noto, which comes in both Sans-Serif and Serif.

10.3 Layout Tricks

How do I create a 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. Especially if you don't use the same font in your manuscript as in the editor.

How do I force a line break when line breaks are ignored in my manuscript?

In the **Manuscript Build Settings** you can choose to ignore line breaks within paragraphs in your text. However, some times you still need those breaks. Like for instance on the cover page where you may need to add your name and address. In such cases, you can add `[br]` where you want line breaks. These breaks cannot be ignored by any settings and will always be respected.

New in version 2.6.

How do I turn off first line indent for a specific paragraph?

If you have first line indent enabled, but have a specific paragraph that you don't want indented, you can disable the indentation by explicitly adding text alignment. For instance by adding << to the end to left-align it. Aligned paragraphs are not indented.

See *Alignment and Indentation* for more details.

10.4 Other Tools

How do I convert my project to/from the 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

ELEVEN

THE MAIN WINDOW

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 by default have an editor toolbar like many other applications do. This reduces clutter, and since the documents are formatted with style tags, it is not needed most of the time. Still, a small formatting toolbar can be popped out by clicking the left-most button in the header of the document editor. It gives quick access to standard formatting codes.

Most formatting features supported are also available through keyboard shortcuts, as well as available in the main menu under **Format**, so you don't have to look up formatting codes every time you need them. For reference, a list of all shortcuts can be found in the *Keyboard Shortcuts* section.

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.

11.1 Project Tree and Editor View

When **Project Tree View** in the sidebar is selected, the work area 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 side 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. For instance if you need to check that the formatting tags behave as you expect. However, 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.

The project tree will highlight with a different background colour the document that is currently open in the editor.

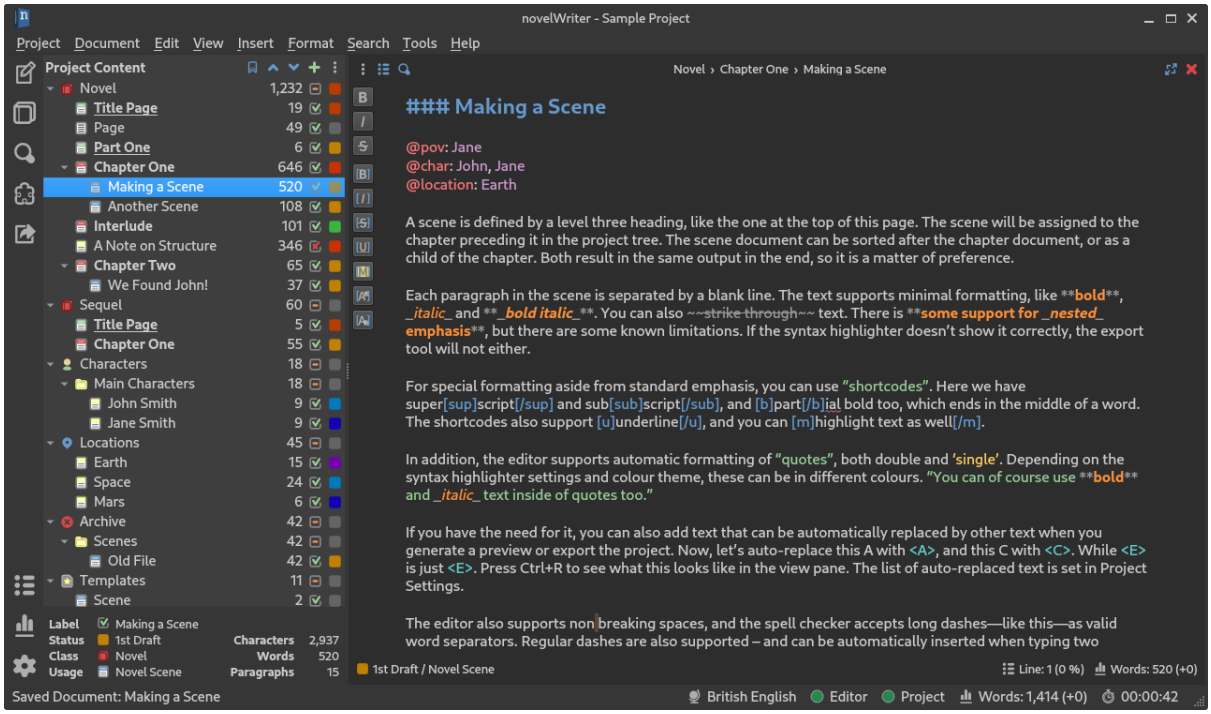


Fig. 1: A screenshot of the Project Tree and Editor View.

11.1.1 Drag & Drop

The project tree allows drag & drop so 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.

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** type root folder, its "Importance" setting will be replaced by a "Status" setting. See *Importance and Status* for more details. 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 using the arrow buttons at the top of the project tree, or by pressing `Ctrl+Up` or `Ctrl+Down` when they are selected.

Tip: You can drag and drop documents onto the editor or viewer panel to open them.

New in version 2.6.

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

New in version 2.2.

11.2 Novel View and Editor View

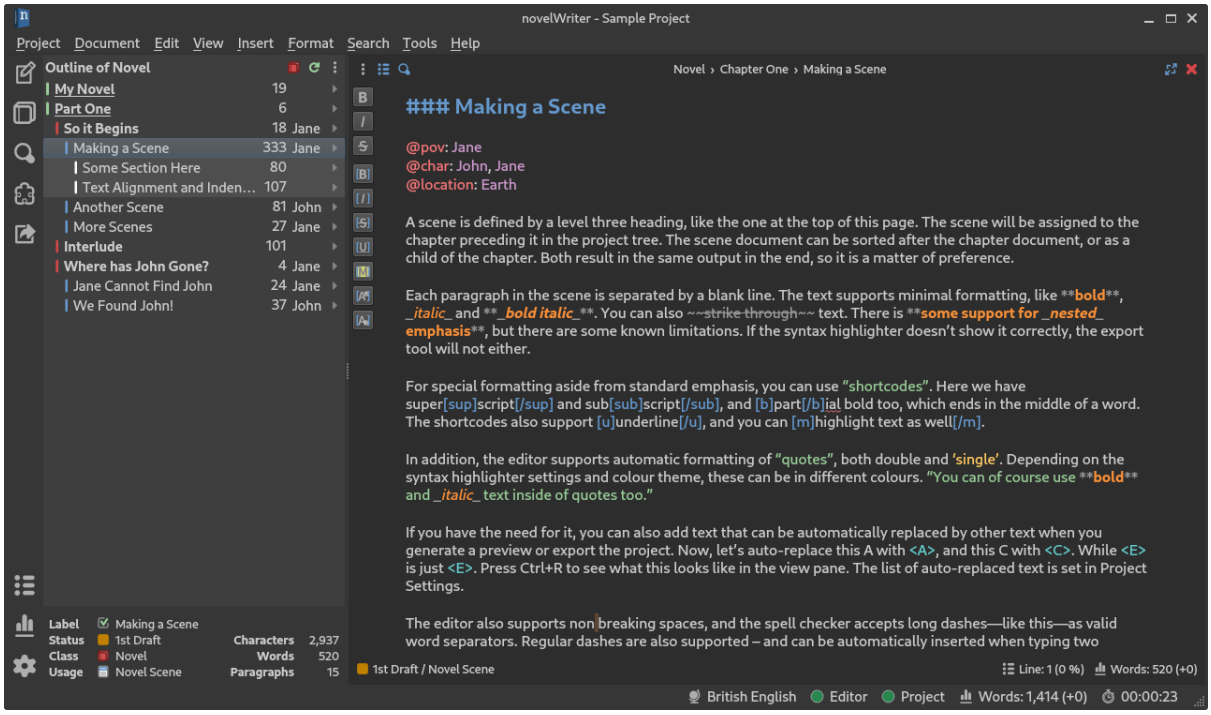


Fig. 2: A screenshot of the Novel Tree and Editor View.

When **Novel Tree View** in the sidebar is selected, the project tree is replaced by an overview of your novel structure for a specific **Novel** type root folder. Instead of showing individual documents, the tree now shows all headings of your text. This includes multiple headings within the same document.

Each heading is indented according to the heading level, not its parent/child relationship to other elements of your project. 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** type 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 in the novel view header.

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

Note: You cannot reorganise the entries in the novel view, or add any new documents, as that would imply restructuring the content of the documents 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.

11.3 Novel Outline View

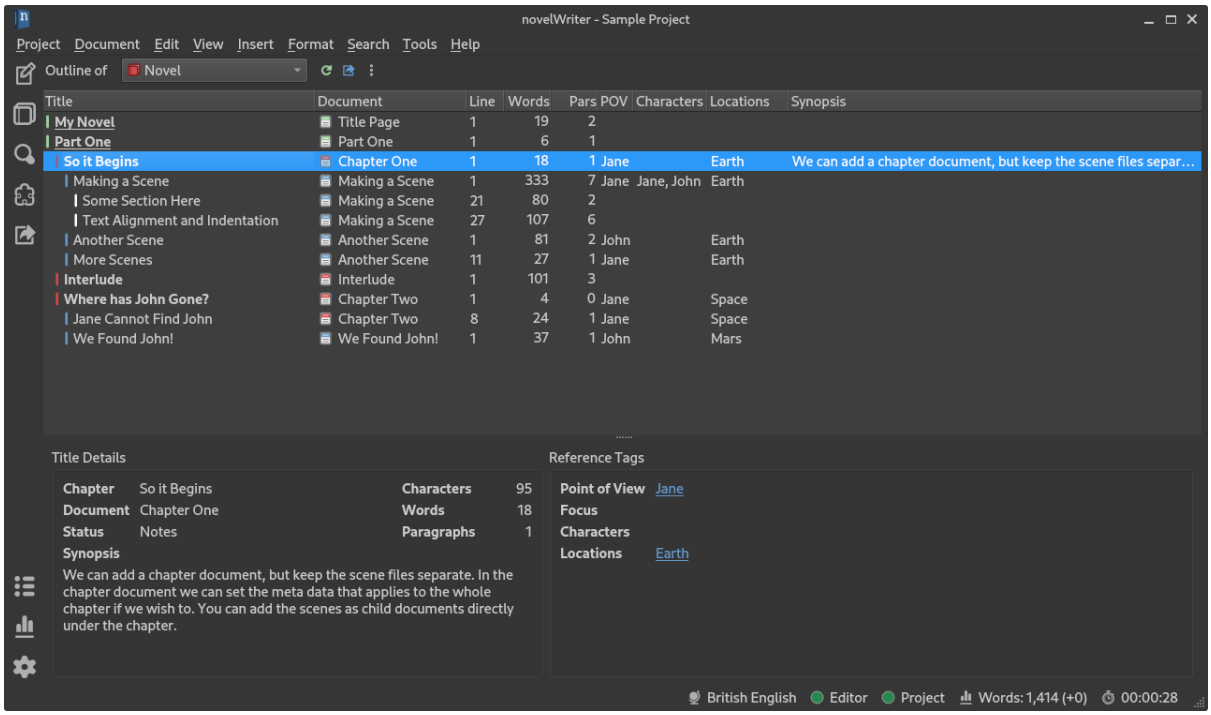


Fig. 3: A screenshot of the Novel Outline View.

When **Novel Outline View** in the sidebar is selected, the tree, editor and viewer are replaced by a table that shows the entire novel structure with all the tags and references listed. You can select which novel folder to display from the dropdown menu. You can optionally choose to show a combination of all novel folders.

Pretty much all collected meta data is available in this view. You can select which columns to display from the menu button. This includes various meta data and information extracted from your *Tags and References*. The order of the columns can also be changed 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 *Synopsis or Description Comments*.

11.4 Project Search

A global search tool is available from the side bar. It allows you to search through your entire project. The tool does not provide a replace feature. There is a search and replace tool available in the document editor that acts on the open document. See *Search & Replace* for more details.

New in version 2.4.

11.5 Switching Focus

If the project or novel view does not have focus, pressing `Ctrl+T` switches focus to whichever of the two is visible. If one of them already has focus, the key press will switch between them instead.

Likewise, pressing `Ctrl+E` will switch focus to the document editor or viewer, or if any of them already have focus, it will switch focus between them.

These two shortcuts make it possible to jump between all these GUI elements without having to reach for the mouse or touchpad.

See *Keyboard Shortcuts* for more details.

11.6 Colour Themes

By default, novelWriter uses a light colour theme. You can also choose between a standard dark theme that have neutral colours, or a series of other included themes, from **Preferences**.

If you wish, you *can* create your own colour themes, and even have them added to the application. See *Colour Themes* for more details.

Switching the GUI colour theme does not affect the colours of the editor and viewer. They have separate themes selectable from the "Document colour theme" setting in **Preferences**. They are separated because there are a lot more options to choose from for the editor and viewer.

Note: If you switch between light and 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.

CHAPTER

TWELVE

MANAGING PROJECTS

Your text in novelWriter is organised into projects. Each project is meant to contain one novel and associated notes. If you have multiple novels in a series, with the same characters and shared notes, it is also possible to keep all of them in the same project by creating multiple **Novel** root folders. See *How Root Folders Work* for more details.

12.1 Creating A New Project

You can create a new project from the **Project** menu by selecting **Create or Open Project**. This will open the **Welcome** dialog, where you can select the *New* button that will assist you in creating a project. This dialog is also displayed when you start novelWriter.

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

A list of recently opened projects is maintained, and displayed in the **Welcome** dialog. A project can be removed from this list by selecting it and pressing the `Del` key or by right-clicking it and selecting the **Remove Project** option.

Fig. 1: The project list (left) and new project form (right) of the **Welcome** dialog.

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's novel text, including word counts, and a table of contents with word and page counts, is available through the **Novel Details** dialog. Statistics about the project is also available in the **Manuscript Build** tool.

12.1.1 Template Projects

From the Welcome dialog you can also create a new from another existing project. If you have a specific structure you want to use for all your new projects, you can create a dedicated project to be used as a template, and select to copy an existing project from the "Prefill Project" option from the **New Project** form.

12.2 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.

12.2.1 General Settings

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

The **Project Name** can be edited here. It is used for the main window title and for generating backup files. So keep in mind that if you do change this setting, the backup file names will change too.

You can also change the **Author** and **Project Language** setting. These are only used when building the manuscript, for some formats. The language setting is also used when inserting text into documents in the viewer, like for instance labels for keywords and special comments.

If your project is in a different language than your main spell checking language is set to, you can override the default setting here. The project language can also be changed from the **Tools** menu.

You can also override the automatic backup setting for the project if you wish.

12.2.2 Status and Importance

Each document or folder of type **Novel** can be given a "Status" label accompanied by a coloured icon with an optional shape selected from a list of pre-defined shapes. Each document or folder of the remaining types can be given an "Importance" label with the same customisation options.

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.

Both status and importance labels can be exported and imported so you can share them between projects, or define a standard set for all your writing projects. When you import labels to a project, they are always added as *new* labels.

See also *Importance and Status*.

Note: Status or importance level currently in use cannot be deleted, but they can be edited.

12.2.3 Text Auto-Replace

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.

12.3 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**. The path defaults to a folder named "Backups" in your home directory.

Backups can 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** 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.

CHAPTER

THIRTEEN

THE EDITOR AND VIEWER

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

13.1 Editing a Document

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

To edit a document, double-click it in the project tree, press the `Return` key while having it selected, or drag and drop it onto the editor panel. This will open the document in the document editor.

The editor has a maximise button, which 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, a drop down menu for navigating between headings, 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.

13.1.1 Spell Checking

A third party library called Enchant is used for spell checking in the editor. The controls for spell checking are found in the **Tools** menu. You can also set spell checking language in **Project Settings**.

This spell checking library comes with support for custom words that you can add by selecting "Add Word to Dictionary" from the context menu when a word is highlighted by the spell checker as misspelled. The custom words are managed on a per-project basis, and the list of words can be edited from the **Project Word List** tool available from the **Tools** menu.

Note: Generally, spell checking dictionaries are collected from your operating system, but on Windows they are not. See *Spell Check Dictionaries* for how to add spell checking dictionaries on Windows.

13.1.2 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`.

The rules for how the counts are made is covered in more detail in *Word and Text Counts*.

Tip: For some languages, character count is the more interesting statistics. You can select to display character count instead of word count in the user interface in **Preferences**.

New in version 2.7.

13.2 Search & Replace

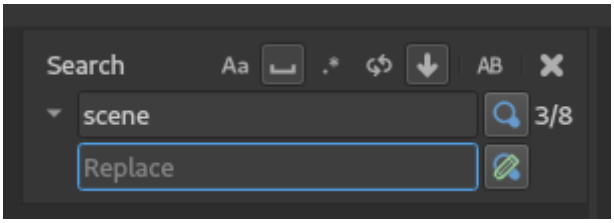


Fig. 2: A screenshot of the Document Editor search box.

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.

13.3 Auto-Replace as You Type

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

Tip: If you don't like auto-replacement, all symbols inserted by this feature are also available in the *Insert* menu, and via *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, four dashes with a horizontal bar, three dots 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.

13.4 Viewing a Document

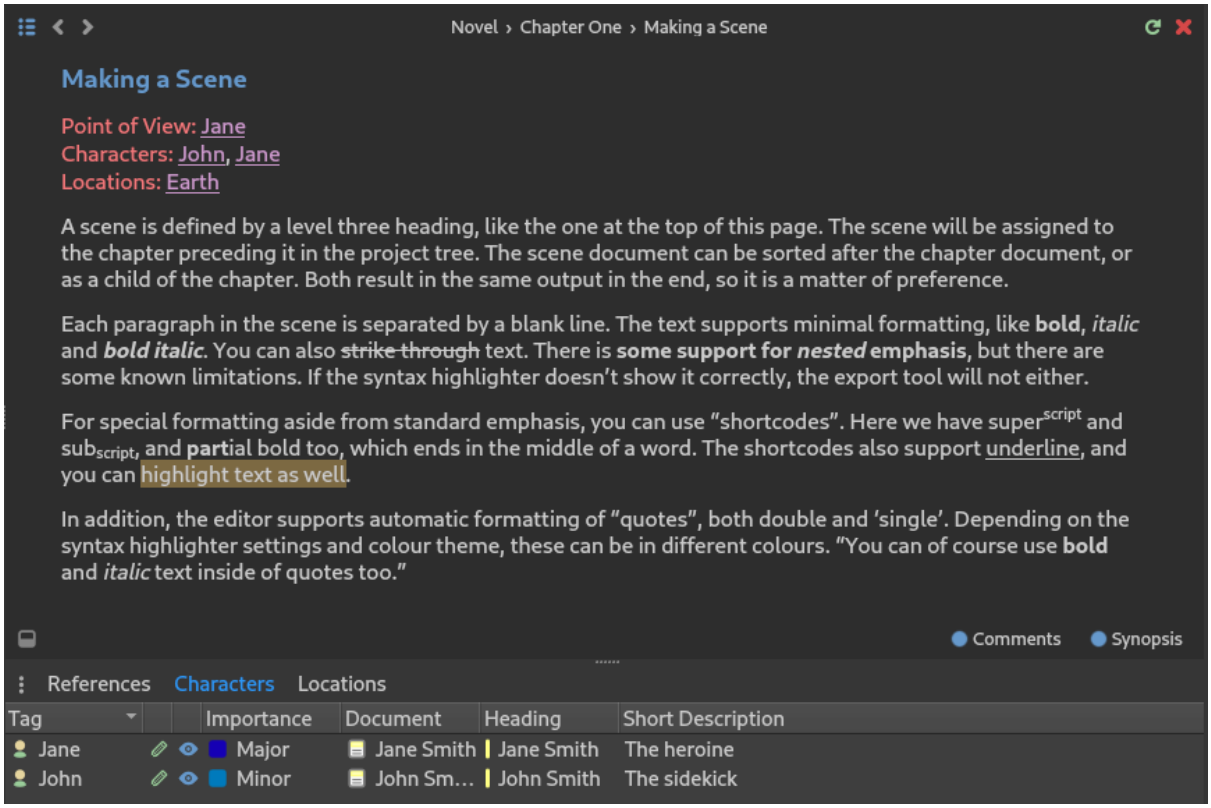


Fig. 3: A screenshot of the Document Viewer panel.

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, select **View Document** in the menu or context menu, or drag and drop the document onto the viewer panel. 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 from the editor 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. The left-most button is a dropdown menu for quickly navigation between headings in the document. The edit button on the right will open the viewed document in the editor.

13.4.1 Document References

At the bottom of the viewer panel you will find 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 project index. 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`.

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

CHAPTER

FOURTEEN

SPLIT AND MERGE 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.

14.1 Splitting Documents

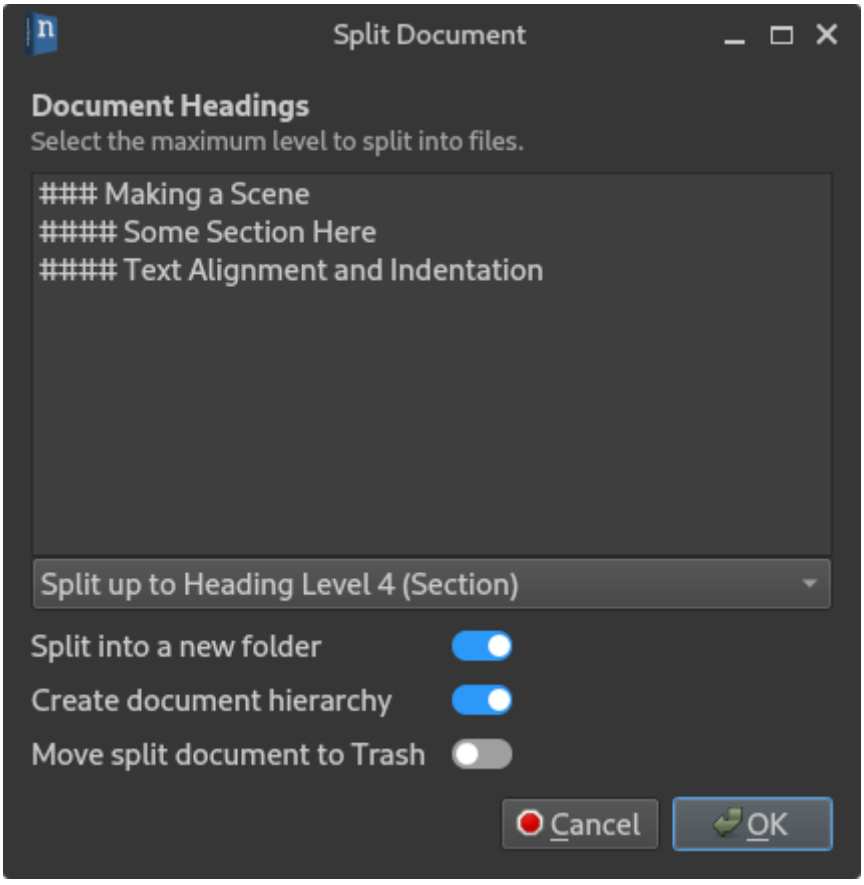


Fig. 1: The **Split Document** dialog.

The **Split Document by Headings** 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.

14.2 Merging Documents

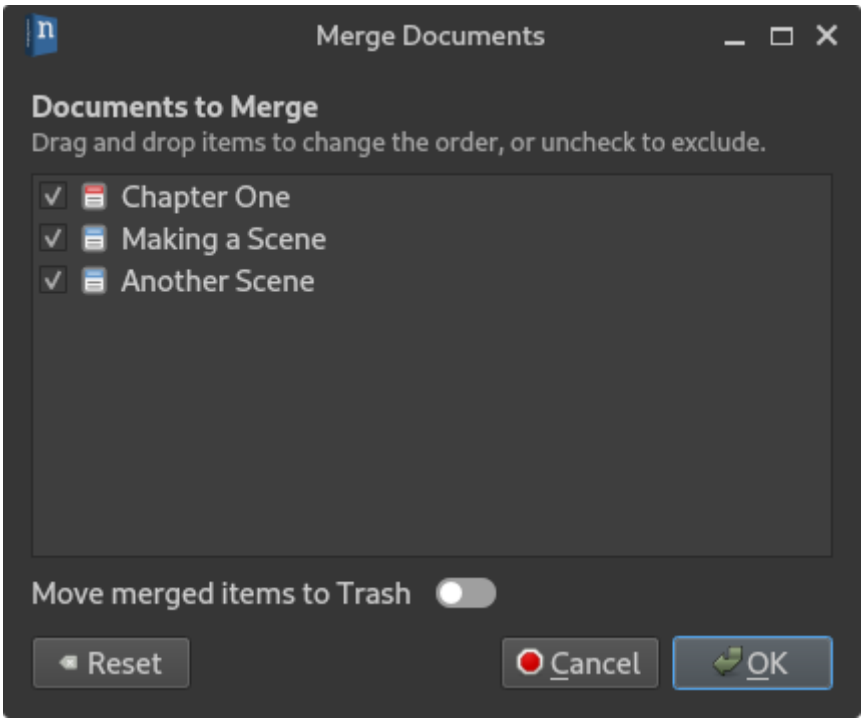


Fig. 2: The **Merge Documents** dialog.

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, only the latter option 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.

CHAPTER

FIFTEEN

BUILDING THE MANUSCRIPT

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

Note: The term "Build" in this context means to assemble or generate a single document from a selection of your project documents. You can select between multiple standard document formats.

New in version 2.1: This tool is new for version 2.1. A simpler tool was used for earlier versions.

15.1 The Manuscript Build Tool

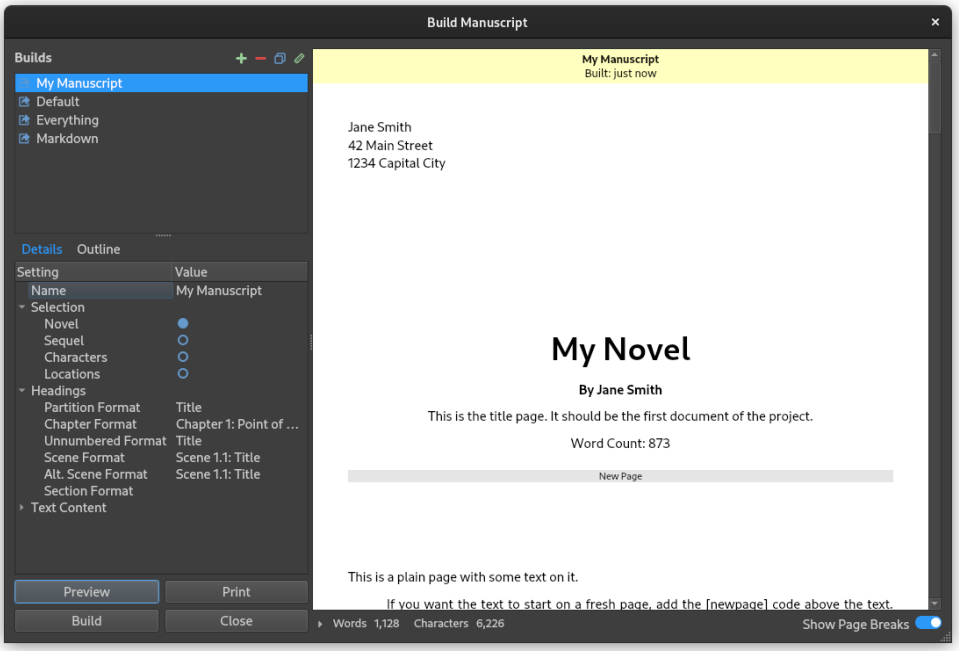


Fig. 1: The **Manuscript Build** tool main window.

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.

15.1.1 Outline and Word Counts

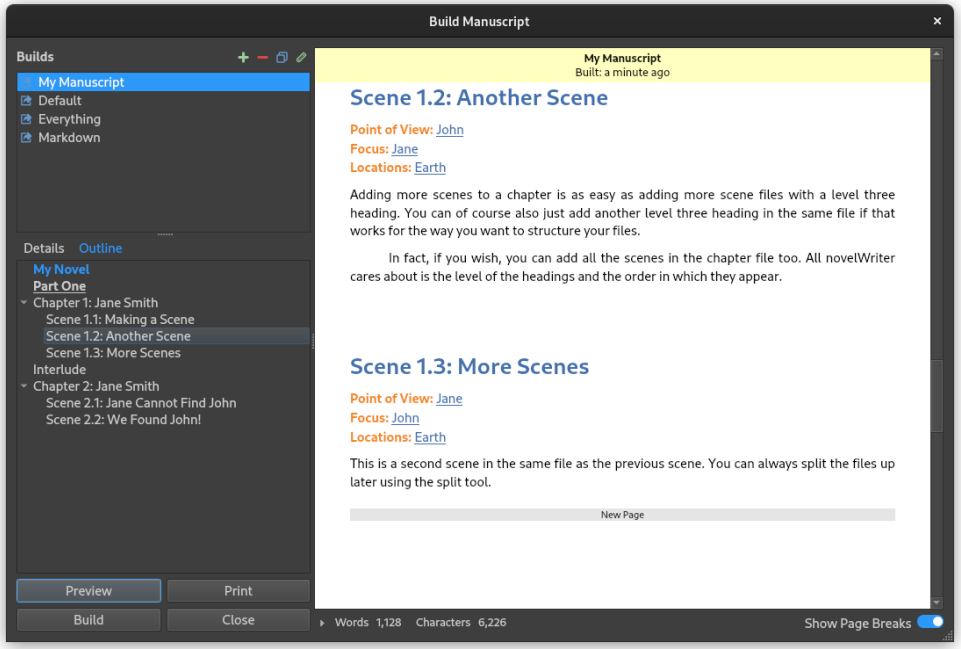


Fig. 2: The **Manuscript Build** tool main window with the **Outline** visible.

The **Outline** tab on the left lets you navigate the headings in the preview document. It will show up to scene level headings for novel documents, and level 2 headings for notes.

A collapsible panel of word and character counts is also available below the preview pane. These are calculated from the text you have included in the document, and are more accurate counts than what's available in the project tree since they are counted *after formatting*.

For a detailed description on how they are counted, see *Word and Text Counts*.

15.2 Build Settings

You can edit a build definition 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 change by pressing the *Preview* button in the main **Manuscript Build** window. When you're happy with the result, you can close the settings.

15.2.1 Document Selection

Fig. 3: 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. The included documents are indicated by an 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 orange icon and the blue icon indicates which documents are included, and the red icon indicates that a document is explicitly excluded.

By default, inactive documents are excluded, but you can override this in the filter settings. See *Active and Inactive Documents* for more details.

15.2.2 Formatting Headings

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, or insert character names, like shown in the 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

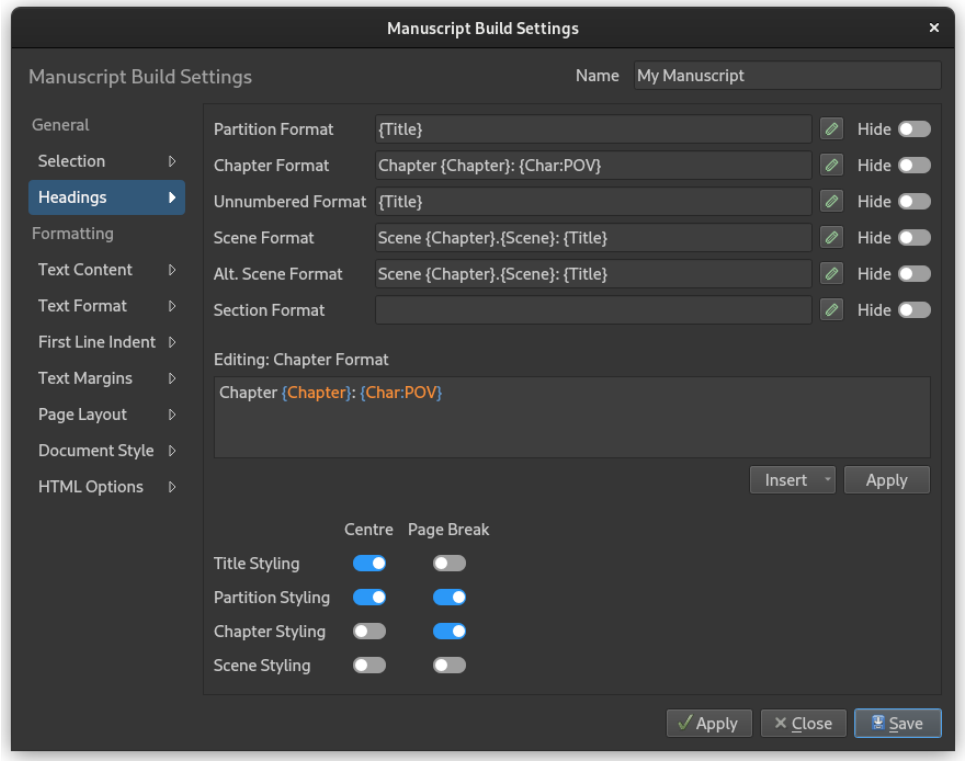


Fig. 4: The **Headings** page of the **Manuscript Build Settings** dialog.

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.

Table 1: Heading Formats

Code	Description
`{BR}`	Insert a line break.
`{Title}`	Insert the original title text.
`{Chapter}`	Insert a chapter number.
`{Chapter:Word}`	Insert a chapter number as a word.
`{Chapter:URoman}`	Insert a chapter number as an upper case Roman numeral.
`{Chapter:LRoman}`	Insert a chapter number as an lower case Roman numeral.
`{Scene}`	Insert a scene number within the current chapter.
`{Scene:Abs}`	Insert a scene number unique to the whole manuscript.
`{Char:POV}`	Insert the point-of-view character's *display name*.
`{Char:Focus}`	Insert the focus character's *display name*.

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

Note: The language used for generating chapter numbers is defined by the project language set in **Project Settings**. This feature relies on translation files existing for each language. If you want to help add new

languages to novelWriter, see: *Contributing*.

Automatic Numbering

The headings formatter allows you to automatically insert chapter and scene numbers into your headings. The automatic chapter number counter will skip all chapter headings marked as unnumbered using the heading format described in *Heading Levels*.

Scene numbers are mostly intended for use in a draft manuscript. You can either insert absolute scene numbers that counts every scene in the novel, or you can insert per-chapter scene numbers that reset to 1 for each new chapter.

Example

This will create a chapter title on the format "Chapter 1: Title Text":

```
Chapter {Chapter}: {Title}
```

This will create a scene title on the format "Scene 1.1: Title Text":

```
Scene {Chapter}.{Scene}: {Title}
```

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 enable 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. A static separator is only inserted between scenes, as opposed to a formatted heading which is also inserted before the first scene of a chapter.

Hard and Soft Scenes

If you wish to distinguish between so-called soft and hard scene breaks, you can use the alternative scene heading format in your text. You can then give these headings a different formatting in the **Headings** settings.

See *Heading Levels* for more info on how to format alternative scene headings in your text.

15.2.3 Output Settings

The **Formatting** sections of the **Manuscript Build Settings** dialog control a number of other settings for the output. This includes formatting, but also what content is included. You can for instance select to include comments, synopsis. tags and reference, and even exclude the body text itself.

15.3 Building Manuscript Documents

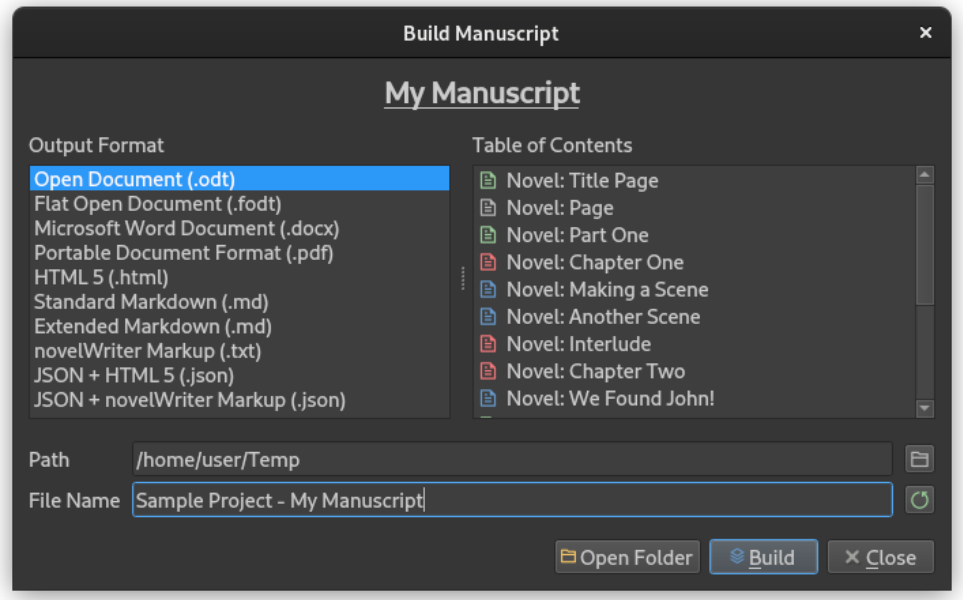


Fig. 5: The **Manuscript Build** dialog used for writing the actual manuscript documents.

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.

15.3.1 File Formats

The following document formats are supported:

Open Document
: 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.

Microsoft Word Document
: The Microsoft Word Document format writes a single `.docx` file. It uses a fairly basic format that should be compatible with most rich text editors.

Portable Document Format (PDF)
: The PDF is generated from a copy of the preview document, and should have the same formatting capabilities as the preview. It's identical to what is produced if you select the print option and print to PDF.

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.

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 strike through text, which is not supported by the Standard flavour.

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.

New in version 2.6: Microsoft Word and PDF output options were added.

15.3.2 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.

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.

15.4 Printing

The *Print* button allows you to print the content in the preview window. You can either print to one of your system's printers, or select PDF as your output format from the printer icon on the print dialog.

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

CHAPTER

SIXTEEN

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 itself, 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.

16.1 Idle Time

The log file 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. You can change the number of minutes in **Preferences**.

16.2 Session Timer

A session timer is by default visible in the status bar. The icon will show you a clock icon when you are active, and a pause icon when you are considered "idle" per the criteria mentioned above.

If you do not wish to see the timer, you can click on it once to hide it. The icon will still be visible. Click the icon once more to display the timer again.

New in version 2.6: As of version 2.6, clicking the timer text or icon in the status bar will toggle its visibility.

CHAPTER

SEVENTEEN

DIALOGUE HIGHLIGHTING

Dialogue recognition and colour highlighting is available both while you're writing and in generated manuscript documents.

The default language settings in novelWriter are for English. That includes the dialogue highlighting settings. But many dialogue styles are supported. You can tune a number of settings to fit your language and style preferences in the "Text Highlighting" section in **Preferences**. You can mix and match these settings.

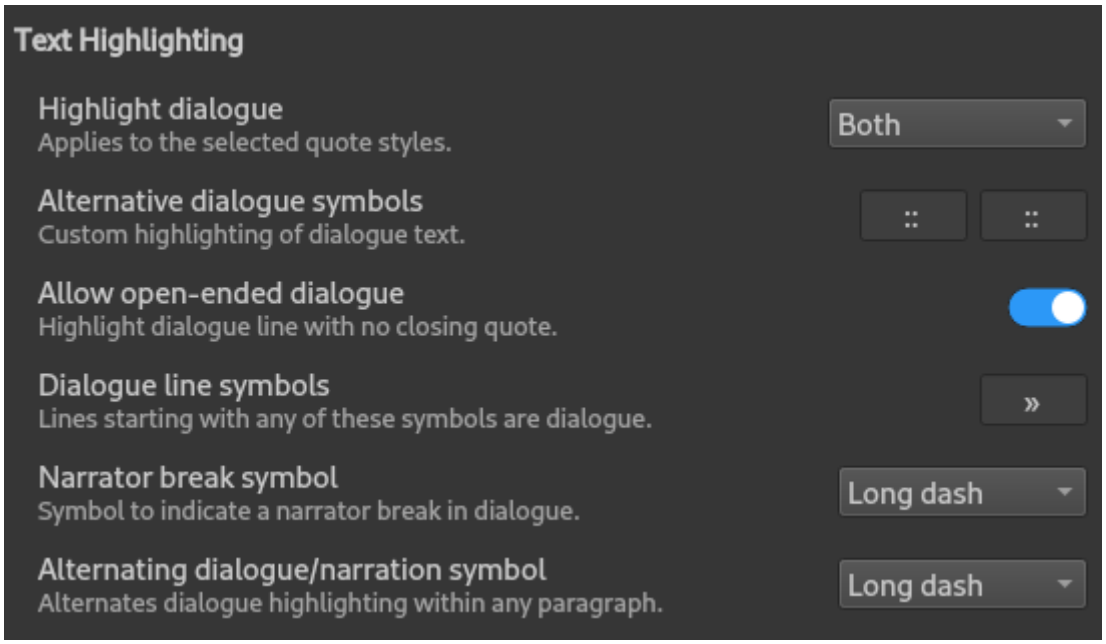


Fig. 1: The **Text Highlighting** section of **Preferences**.

17.1 Quoted Dialogue

By default, dialogue highlighting is enabled for the double quote symbols you have defined in the **Quotation Style** section of **Preferences**.

You can change which quote symbols are highlighted by selecting one of "None", "Single", "Double", or "Both" from the "Highlight dialogue" setting under **Text Highlighting**.

You can also enable or disable the "Allow open-ended dialogue" setting to allow for the style where multi-paragraph dialogue is not closed until the last paragraph.

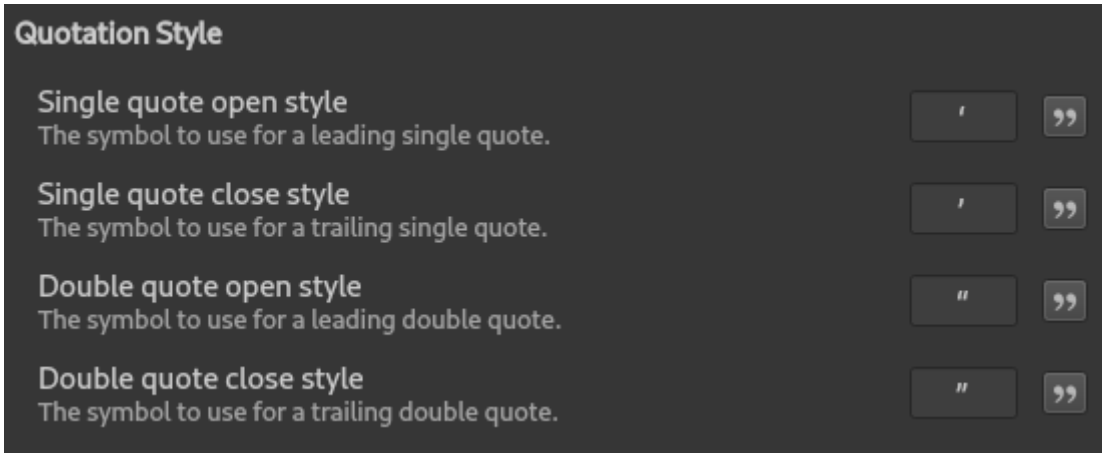


Fig. 2: The **Quotation Style** section of **Preferences**.

Limitations

Dialogue highlighting for single quotes is difficult to process when the same single quote symbol is also used for apostrophes. There isn't a good solution to this. Your best option in the cases where the highlighting is wrong is to insert an alternative apostrophe symbol instead. See *Modifier Letter Apostrophe* for more details.

17.2 Alternative Dialogue

You can use the "Alternative dialogue symbols" setting for custom dialogue wrapper symbols. These are highlighted in a different colour than regular dialogue.

The intended use case here is if you use an alternative style to distinguish a different style of communication. The feature idea came from a science fiction series where mind-to-mind communication used a different quotation style.

17.3 Dialogue Line Symbols

In some languages, a single symbol at the start of a paragraph can indicate that the whole paragraph is dialogue. For instance, this symbol can be a short dash (en dash).

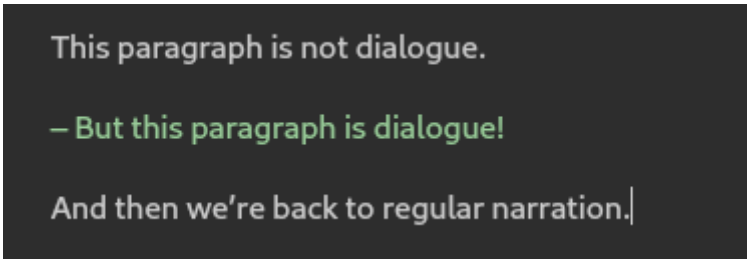


Fig. 3: An example of dialogue starting with a short dash.

You can enable this feature by adding the symbols to the "Dialogue line symbols" setting. Multiple symbols are allowed.

17.4 Dialogue with Narrator Break

The dialogue symbol setting will not detect if the dialogue ends in the paragraph. In some styles there is no way to actually indicate the switch from dialogue to narration; in others there are. These a narrator break symbols are usually dashes. You can select one of the supported dash symbols for narrator breaks. These can be used with any of the above dialogue recognition settings.

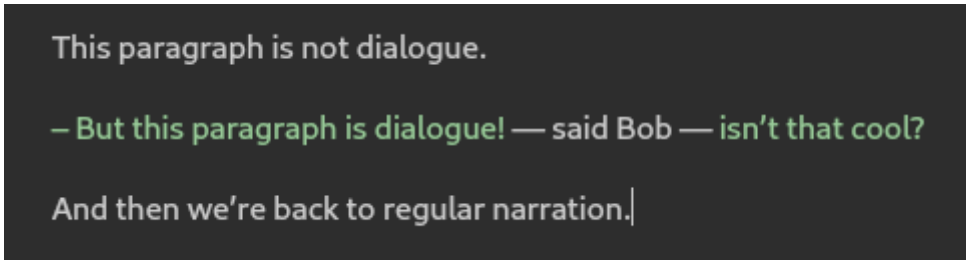


Fig. 4: An example of dialogue starting with a short dash and a long dash narrator break.

17.5 Alternating Dialogue and Narration

The alternating dialogue and narration style is supported with the "Alternating dialogue/narration symbol" setting. It can be set to one of the supported dashes. This style will switch into dialogue mode when it first encounters the selected dash in a paragraph, and switch back out when it sees the next one, and so forth.

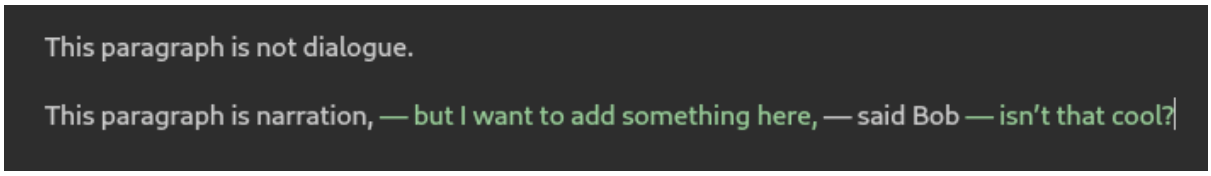


Fig. 5: An example of alternating dialogue and narration using a long dash.

CHAPTER

EIGHTEEN

STORY COMMENTS

A special set of comment styles allow for annotating your text with structure information. There are two styles of comments available. They are an extension to regular comments as described in *Comments and Notes*.

New in version 2.7.

18.1 Story Structure Comments

You can annotate story structure by using the `%Story` style of comment. To use the feature, make the first word of a comment `Story`, followed by a period, a structure term, a colon, a space and the text for that term.

18.1.1 Usage

The story term can be anything that you want to track in the manuscript. This construct is intended to make it easier to extract metadata from a work to perform a structural analysis of the story.

There are probably as many ways to examine story structure as there are authors and editors combined. For this reason the story tag is flexible. You can use any terms you want and track any aspect of the story that serves your purposes.

An example method has been advanced by Shawn Coyne in *The Story Grid*. This method asserts that a story is composed of "beats", and that each beat has an inciting incident, a complication, a crisis, and a resolution. One might capture these elements of a beat where a character overcomes their fear of giving a speech as:

Example

```
### Scene

%Synopsis: Carol overcomes her fear of giving a speech.

%Story.incite: Carol is pleased to be invited to a conference to see her boss␣
↪deliver a keynote.
%Story.complication: Carol's boss calls in sick and asks her to deliver a big␣
↪speech.
%Story.crisis: Carol has a fear of appearing on stage.
%Story.resolution: Carol engages the help of a coach who helps her overcome␣
↪her fears and delivers a great speech.
```

Other analytical models propose tracking a scene's pace, how it affects the mood of the story, or which element(s) of the story's genre are being satisfied. An author can use this mechanism to track any element of a scene. Some examples include time of day, how much time passes in the scene, or even the physical form of a shape-shifting character. If a story involves magic, one could track which wand a main character has in hand. It's up to tha author.

When the story and other scene metadata is extracted into a tabular form, it is possible to get a comprehensive overview of the story and to identify possible issues. For example, so many fast-paced scenes without a break that readers might become fatigued or over-stimulated.

18.1.2 Output

The story structure comments can be included in the manuscript, and are formatted similarly to the synopsis comments:

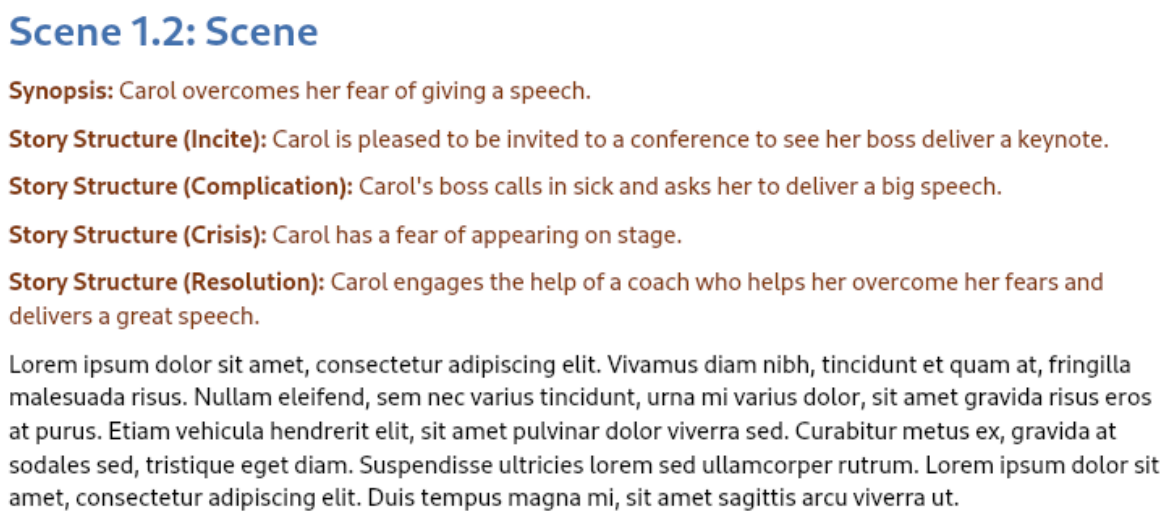


Fig. 1: A set of story structure comments as shown in the Manuscript tool.

When you export your project data from the Outline View, all story structure terms are added as columns to the exported file, which can then be opened in the spread sheet software of your choice.

18.2 Story Notes

Story notes are similar to story structure comments, but have no predefined meaning. Essentially they are a generalisation of the story structure comment, and the only point of having this additional format is to allow you do filter them in and out of your manuscript independently.

You can annotate story notes by using the `%Note` style of comment. To use the feature, make the first word of a comment `Note`, followed by a period, a term, a colon, a space and the text for the note.

18.2.1 Usage

These notes are free form, but one intended use case is to add consistency annotations to your text to remind yourself where you have described something that must be checked against other parts of your text later on.

Example

```
### Scene

%Synopsis: Carol overcomes her fear of giving a speech.

%Note.consistency: This is the first time in the story Carol gives a speech.
```

18.2.2 Output

Story notes are included in the manuscript in exactly the same way story structure comments are, but has a separate inclusion setting in the build settings. They are also included in CSV exports from the **Outline View**.

CHAPTER

NINETEEN

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`.

19.1 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's index
`F11`	Toggle full screen mode
`Ctrl+,`	Open the **Preferences** dialog
`Ctrl+E`	Switch or toggle focus for the editor or viewer
`Ctrl+T`	Switch or toggle focus for the project tree or novel view
`Ctrl+Q`	Exit novelWriter
`Ctrl+Shift+,`	Open the **Project Settings** dialog
`Ctrl+Shift+O`	Open the Welcome dialog to open or create 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

19.2 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 the selected document in the editor
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

19.3 Document Editor Shortcuts

19.3.1 Text Search Shortcuts

Shortcut	Description
F3	Find the next occurrence of the search word
Ctrl+F	Open search and look for the selected word
Ctrl+G	Find the next occurrence of the search word
Ctrl+H	Open replace and look for the selected word (Mac Cmd+=)
Ctrl+Shift+1	Replace selected occurrence, and move to the next
Ctrl+Shift+G	Find the previous occurrence of the search word
Ctrl+Shift+F	Open project search and look for the selected word
Shift+F3	Find the previous occurrence of the search word

19.3.2 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 comment format for block or selected text
`Ctrl+0`	Remove format for block or selected text
`Ctrl+1`	Change block format to heading level 1
`Ctrl+2`	Change block format to heading level 2
`Ctrl+3`	Change block format to heading level 3
`Ctrl+4`	Change block format to heading 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 bold
`Ctrl+D`	Format selected text, or word under cursor, with strike through
`Ctrl+I`	Format selected text, or word under cursor, with italic
`Ctrl+M`	Highlight selected text, or word under cursor
`Ctrl+Shift+/`	Remove format for block or selected text
`Ctrl+Shift+D`	Toggle ignored text format for block or selected text

19.3.3 Other Editor Shortcuts

Shortcut	Description
`F7`	Re-run the spell checker on the document
`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

19.3.4 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, M`	Insert a `@mention` 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

19.4 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

TWENTY

VIM MODE

Vim is a keyboard-centric text editor. "Vim mode" refers to another editor implementing the ability to perform a subset of vim motions. This will allow you to write, select, edit, copy, paste, navigate, etc, efficiently by using keyboard commands.

Vim is modal. Three such modes have been implemented in novelWriter:

- **Normal** mode is the default mode used to navigate text.
- **Insert** mode is where you can write text like in a 'normal' text editor.
- **Visual** mode is for selecting text, with:
  - **Visual** mode for per character selection.
  - **V-Line** mode for per-line selection.

The vim mode setting is found in the **Features** section of the **Preferences**.

Note: Vim mode is an advanced feature. When you enable it, the text editor will behave very differently than a standard text editor. A label in the footer of the editor will show which of the vim modes it is in.

20.1 Mode Switching

To switch between the various vim modes, these keystrokes implemented:

From Mode	To Mode	Keystrokes
Normal	Insert	`i`, `I`, `a`, `A`, `o`, `O`
Insert	Normal	`Esc`
Normal	Visual	`v`, `V`

You can exit visual mode back to normal mode by pressing `Esc`, but all visual mode commands that logically "terminate" the visual mode usage will return you to normal mode.

For instance, press `V` to enter V-Line mode, select next word with `w`. press `y` to yank (copy), you are now automatically put back in normal mode as you have completed selecting your text.

20.2 Implemented Keystrokes (Vim Motions)

The table below shows the vim motions currently implemented in novelWriter.

Motion	Mode(s)	Description
`i`	Normal → Insert	Enter insert mode
`I`	Normal → Insert	Jump to first non-blank of line and enter insert
`v`	Normal → Visual	Enter character-wise visual mode
`V`	Normal → V-Line	Enter line-wise visual mode
`dd`	Normal	Delete (cut) current line
`x`	Normal	Delete character under cursor
`w`	Normal / Visual	Move to next word start
`b`	Normal / Visual	Move to previous word start
`e`	Normal / Visual	Move to next word end
`dw`	Normal	Delete from cursor to next word start
`de`	Normal	Delete from cursor to next word end
`db`	Normal	Delete from cursor to previous word start
`d$`	Normal	Delete from cursor to end of line
`yw`	Normal	Yank (copy) from cursor to next word start
`gg`	Normal / Visual	Jump to first line
`G`	Normal / Visual	Jump to last line
`yy`	Normal	Yank current line
`p`	Normal	Paste after current line
`P`	Normal	Paste before current line
`o`	Normal → Insert	Open new line below and enter insert
`O`	Normal → Insert	Open new line above and enter insert
`$`	Normal / Visual	Move to end of line
`a`	Normal → Insert	Enter insert after cursor
`A`	Normal → Insert	Enter insert at end of line
`u`	Normal	Undo last change
`zz`	Normal	Centre cursor vertically
`h`	Normal / Visual	Move left
`j`	Normal / Visual	Move down
`k`	Normal / Visual	Move up
`l`	Normal / Visual	Move right
`d`	Visual / V-Line	Delete selection
`x`	Visual / V-Line	Delete selection
`y`	Visual / V-Line	Yank selection
`0`	Visual	Move to start of line (extend selection)

20.3 Known Issues

- Currently, `dd` on an empty line will not delete, but using `x` will.
- `vypyy` will have `ypyy` in command memory and thus not register `yy`. Expected behavior would be visual mode, yank, paste, yank line.
- Differing behavior from vim: the `e` command behaves a bit differently with regards to the last character of a word. The behavior is inconsistent with vim but functional and still logical to use. The cursor is placed at the end of the word after the last character rather than on the last character.

```
test
   ^ Cursor placed here in vim
test
    ^ Cursor placed here in novelWriter vim mode
```

  You will only really ever notice this behavior if you try to combine the `e` command with another. For instance, `de` will not delete the last character but delete forward as it starts one character after the word boundary.

CHAPTER

TWENTYONE

WORD AND TEXT COUNTS

This is an overview of how words and other counts of your text are performed. The counting rules should be relatively standard, and are comparable to Libre Office Writer rules.

The counts provided in the app on the raw text are meant to be approximate. For more accurate counts, you need to build your manuscript in the **Manuscript Tool** and check the counts on the generated preview.

21.1 Text Word Counts and Stats

These are the rules for the main counts available for for each document in a project.

For all counts, the following rules apply.

1. Short (–) and long (—) dashes are considered word separators.
2. Any line starting with `%` or `@` is ignored.
3. Trailing white spaces are ignored, including line breaks.
4. Leading > and trailing < are ignored with any spaces next to them.
5. Valid shortcodes and other commands wrapped in brackets `[]` are ignored.
6. In-line Markdown syntax in text paragraphs is treated as part of the text.

After the above preparation of the text, the following counts are available.

Character Count
: The character count is the sum of characters per line, including leading and in-text white space characters, but excluding trailing white space characters. Shortcodes in the text are not included, but Markdown codes are. Only headings and text are counted.

Word Count
: The words count is the sum of blocks of continuous character per line separated by any number of white space characters or dashes. Only headings and text are counted.

Paragraph Count
: The paragraph count is the number of text blocks separated by one or more empty line. A line consisting only of white spaces is considered empty.

21.2 Manuscript Counts

These are the rules for the counts available for a manuscript in the **Manuscript Tool**. The rules have been tuned to agree with LibreOffice Writer, but will vary slightly depending on the content of your text. LibreOffice Writer also counts the text in the page header, which the **Manuscript Tool** does not.

The content of each line is counted after all formatting has been processed, so the result will be more accurate than the counts for text documents elsewhere in the app. The following rules apply:

1. Short (–) and long (—) dashes are considered word separators.
2. Leading and trailing white spaces are generally included, but paragraph breaks are not.
3. Hard line breaks within paragraph are considered white space characters.
4. All formatting codes are ignored, including shortcodes, commands and Markdown.
5. Scene and section separators are counted.
6. Comments and meta data lines are counted after they are formatted.
7. Headers are counted after they are formatted with custom formats.

The following counts are available:

Headings
: The number of headings in the manuscript.

Paragraphs
: The number of body text paragraphs in the manuscript.

Words
: The number of words in the manuscript, including any comments and meta data text.

Words in Text
: The number of words in body text paragraphs, excluding all other text.

Words in Headings
: The number of words in headings, including inserted formatting like chapter numbers, etc.

Characters
: The number of characters in all lines, including any comments and meta data text. Paragraph breaks are not counted, but in-paragraph hard line breaks are.

Character in Text
: The number of characters in body text paragraphs. Paragraph breaks are not counted, but in-paragraph hard line breaks are.

Characters in Headings
: The number of characters in headings.

Character in Text, No Spaces
: The number of characters in body text paragraphs considered part of a word or punctuation. That is, white space characters are not counted.

Character in Headings, No Spaces
: The number of characters in headings considered part of a word or punctuation. That is, white space characters are not counted.

CHAPTER

TWENTYTWO

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.

22.1 Dashes and Ellipsis

With the auto-replace feature enabled (see *Auto-Replace as You Type*), two and three hyphens are converted automatically to short and long dashes, four hyphens to a horizontal bar, and three dots to ellipsis.

Tip: 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.

22.2 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**.

If your text contains straight single and double quotes, there are two convenience functions in the **Format** menu that can be used to re-format a selected section of text with the correct quote symbols.

You can enable dialogue recognition and colour highlighting for novel documents. See *Dialogue Highlighting* for more details.

22.3 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.

22.4 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 in some places, and not others.

22.5 White 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.

You can insert these spaces in your text using the following keyboard combinations:

- 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. The keyboard shortcuts for them are also listed in *Keyboard Shortcuts*.

CHAPTER

TWENTYTHREE

SPELL CHECK DICTIONARIES

Spell checking is provided by the Enchant library. Depending on your operating system, it may or may not load all installed spell check dictionaries automatically.

23.1 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.

23.2 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 and should thus work for any application that uses Enchant for spell checking.

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.

CHAPTER

TWENTYFOUR

CUSTOM THEMES

There are a few ways you can customise novelWriter yourself. Currently, you can relatively easily add new GUI themes. You can also add new icon themes, although this is not as straightforward.

24.1 Colour Themes

Adding your own colour 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 colour themes are stored in novelwriter/assets/themes.

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

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

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

Note: The theme file formats change regularly in new releases. It is up to you to keep custom theme files up to date.

24.1.1 The Theme File Format

A colour theme `.conf` file consists of the following settings:

Listing 1: The theme file for the "Default Light Theme"

```
[Main]
name   = Default Light Theme
mode   = light
author = Veronica Berglyd Olsen
credit = Veronica Berglyd Olsen
```

(continues on next page)

(continued from previous page)

```
url    = https://github.com/vkbo/novelWriter

[Base]
base    = #fcfcfc
default = #303030
faded   = #6c6c6c
red     = #a62a2d
orange  = #b36829
yellow  = #a39c34
green   = #296629
cyan    = #269999
blue    = #3a70a6
purple  = #b35ab3

[Project]
root     = blue
folder   = yellow
file     = default
title    = green
chapter  = red
scene    = blue
note     = yellow
active   = green
inactive = red
disabled = faded

[Icon]
tool      = default
sidebar   = default
accept    = green
reject    = red
action    = blue
altaction = orange
apply     = green
create    = yellow
destroy   = faded
reset     = green
add       = green
change    = green
remove    = red
shortcode = default
markdown  = orange
systemio  = yellow
info      = blue
warning   = orange
error     = red

[Palette]
window          = base:D105
```

(continues on next page)

(continued from previous page)

```
windowtext      = default
base            = base
alternatebase   = #e0e0e0
text            = default
tooltipbase     = #ffffc0
tooltiptext     = #15150d
button          = #efefef
buttontext      = default
brighttext      = base
highlight       = #3087c6
highlightedtext = base
link            = blue
linkvisited     = blue
accent          = #3087c6

[GUI]
helptext  = #5c5c5c
fadedtext = #6c6c6c
errortext = red

[Syntax]
background     = base
text           = default
line           = default:32
link           = blue
headertext     = green
headertag      = green:L135
emphasis       = orange
whitespace     = orange:64
dialog         = blue
altdialog      = red
note           = yellow:D125
hidden         = faded
shortcode      = blue
keyword        = red
tag            = green
value          = green
optional       = blue
spellcheckline = red
errorline      = green
replacetag     = green
modifier       = blue
texthighlight  = yellow:72
```

24.1.2 Theme Sections

The theme file is made up of different sections depending on what part of novelWriter the theme affects.

Table 1: Theme Sections Overview

Section	Description
`[Main]`	Meta data about the theme, You must at least set `name`, `mode` and `author`, and `mode` must be either `light` or `dark`.
`[Base]`	The base colours of the theme. These are also selectable colours in various places inside the app, like for icon colours in **Preferences**.
`[Projec`	The colours used for icons and markers for the different project item types.
`[Icon]`	The colours used for icons and buttons on the user interface. The names correspond to button and icon roles.
`[Palett`	The colours used for styling the user interface. The values correspond to the ColorRole values in the Qt library.
`[GUI]`	The colours used for styling additional elements of the user interface.
`[Syntax`	The colours used for syntax highlighting in documents.

24.1.3 Colour Value Formats

There are several ways to enter colour values:

Table 2: Colour Formats

Syntax	Description
`#RRGGBB`	A CSS style hexadecimal values, like `#ff0000` for red.
`#RRGGBBAA`	A CSS style hexadecimal values with transparency, like `#ff00007f` for half-transparent red.
`name`	A name referring to one of the colours already specified under the `[Base]` section, like `red`. Note that you should not use named colours in the `[Base]` section itself as that may have unintended results.
`name:255`	A name referring to one of the colours already specified under the `[Base]` section, with a transparency value added. The value must be in the range `0` to `255`, like `red:127` for half-transparent red.
`name:L100`	A name referring to one of the colours already specified under the `[Base]` section, where the L-number is a percentage value that makes it lighter. The value must be greater than `0`. `L100` means no change.
`name:D100`	A name referring to one of the colours already specified under the `[Base]` section, where the D-number is a percentage value that makes it darker. The value must be greater than `0`. `D100` means no change.
`r, g, b`	A set of red, green and blue numbers in the range `0` to `255`, like `255, 0, 0` for red.
`r, g, b, a`	A set of red, green, blue and alpha numbers in the range `0` to `255`, like `255, 0, 0, 127` for half-transparent red.

New in version 2.5: The `fadedtext` and `errortext` theme colour entries were added.

New in version 2.7: The `icontheme` setting was dropped as the icon theme is now its own setting. The `[Icons]` and `[Project]` sections were added, and the `status*` settings removed.

New in version 2.8: The `[Syntax]` section was moved into the main theme file. Previously, these settings were in their own file. The `[Icons]` section was renamed to `[Base]`, and a new `[Icon]` section added for button and icon roles. Added the `line` and `whitespace` settings. Dropped the `license`, `licenseurl`, and `description` settings. The `author` field is now required if the theme is included in the app, but not for user themes.

24.2 Icon Themes

Icon themes are *not* straightforward to add, but if you want to make the effort, this section describes how to do it.

The existing icon themes are stored in novelwriter/assets/icons.

As with colour themes, remember to 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 `icons` for icon themes. These folders are created the first time you start novelWriter.

24.2.1 The Icons File Format

Icon themes are kept in files with the `.icons` file extension. The file format is a custom format with entries on the form `section:key = value`.

Listing 2: The icons file for "Material Symbols - Rounded Medium" (truncated)

```
# Meta
meta:name    = Material Symbols - Rounded Medium
meta:author  = Google Inc
meta:license = Apache 2.0

# Icons
icon:alert_error     = <svg ...>
icon:alert_info      = <svg ...>
icon:alert_question  = <svg ...>
icon:alert_warn      = <svg ...>
icon:cls_archive     = <svg ...>
icon:cls_character   = <svg ...>
icon:cls_custom      = <svg ...>
icon:cls_entity      = <svg ...>
icon:cls_none        = <svg ...>
icon:cls_novel       = <svg ...>
icon:cls_object      = <svg ...>
icon:cls_plot        = <svg ...>
icon:cls_template    = <svg ...>
icon:cls_timeline    = <svg ...>
icon:cls_trash       = <svg ...>
icon:cls_world       = <svg ...>
```

The icon keys are associated with icon placement locations inside novelWriter, and the template for them is defined in the script that generates the default icon themes.

The script can be found under utils/icon_themes.py in the source code.

This file includes all the code needed to generate the themes that are included in novelWriter. The icon keys are mapped to icon keys from the specific themes in JSON files in the `icon_themes` folder next to the script. This is the recommended way to generate these themes. Doing it manually is not advisable.

24.2.2 Icon Value Format

As can be seen from the example, an icon is defined in the `icon` section with a key and an in-line SVG XML block. The XML must fit on one line and obey the following rules:

1. It must be single colour, that is, the fill colour attribute must be able to colourise the entire icon.
2. The fill colour attribute *must* be defined and must be set to: `fill="#000000"`. This value is replaced by the relevant theme colour when the icon is processed in novelWriter.

New in version 2.7: The icon theme files were added. Previously, icons were stored as individual SVG files with a config file mapping the file names to the internal icon keys.

CHAPTER

TWENTYFIVE

HANDLING ERRORS

In case something goes wrong, novelWriter has a few built-in features to reduce the chance your work is lost. In case of a crash, it will also try to save whatever changes you have made before exiting, if this is at all possible.

The storage solution is designed to save each text document independently, so only the document you're working on is actually at a risk of losing data in the event of a crash.

25.1 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 storage folder that aren't tracked in the core project file. These files, when discovered, are recovered and added back into the project when a project is opened.

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 a **Novel** does not exist, 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.

25.2 Project Lockfile

To prevent data loss caused by file conflicts when novelWriter projects are synchronised via file synchronisation tools, a project lockfile is written to the project storage folder when a project is open. If you try to open a project that already 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 or your computer 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.

CHAPTER

TWENTYSIX

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 under “More Documents”.

> **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.

26.1 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.

26.2 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.

26.3 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 *Heading Levels*.
- Any text that was previously centred on the page must be manually centred using the text alignment feature. See *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 *Heading Levels*.

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*.

26.4 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. Conversion from this format is done automatically.

26.5 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. Conversion from this format is done automatically.

26.6 Format 1.0 Changes

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

CHAPTER

TWENTYSEVEN

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.

27.1 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.

27.2 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 **Welcome** 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.

The Application Data location also holds several folders:

`cache`
: This folder is used to save the preview data for the **Manuscript Build** tool.

`icons`, `syntax` and `themes`
: These folders are empty by default, but this is where the user can store custom theme files. See *Custom Themes* for more details.

CHAPTER

TWENTYEIGHT

HOW DATA IS STORED

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

28.1 Overview

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 are stored in other folders below the project folder.

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. All the JSON documents have line breaks and indents as well, 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.

28.2 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.

28.2.1 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 under "More Documents".

28.3 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.

28.3.1 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.

28.4 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.

28.4.1 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.

28.4.2 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.

28.4.3 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.

28.4.4 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.

28.4.5 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

TWENTYNINE

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
```

29.1 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:

- `PyQt6` – needed for connecting with the Qt6 libraries.
- `PyEnchant` – needed for spell checking (optional).

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 novelWriter from PyPi, these dependencies should be installed automatically.

You can run novelWriter directly from source with uv:

```
uv run novelwriter
```

If you prefer to install dependencies using `pip`, you must first generate the `requirements.txt` file:

```
python pkgutils.py gen-req
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 rendering the SVG icons may be left out and needs to be installed manually. This is the case on for instance Arch Linux.

29.2 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
```

29.3 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.

29.4 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
```

29.5 Building the Documentation

A local copy of this documentation can be generated as HTML.

If you're using `pip`, you must first generate the `requirements.txt` file:

```
python pkgutils.py gen-req docs
pip install -r requirements.txt
```

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

```
make -C docs html
```

Or you can run directly with uv:

```
uv run 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 docs-pdf en
```

This will build the English 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. Replace `en` with `all` to build for all languages. The Sphinx build system has a few extra dependencies when building the PDF. Please check the Sphinx Docs for more details.

CHAPTER

THIRTY

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.

30.1 Simple Test Run

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

```
uv run pytest
```

This uses uv. See below for manually installing dependencies using `pip`.

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 rendering of the GUI by setting the flag `QT_QPA_PLATFORM=offscreen`:

```
export QT_QPA_PLATFORM=offscreen pytest
```

30.1.1 Dependencies

To generate the requirements file and install dependencies using `pip`, run:

```
python pkgutils.py gen-req app test
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`.

30.2 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.