novelWriter Documentation

Release 1.5-beta1

Veronica Berglyd Olsen

Sunday, 22 August 2021 at 11:38

INTRODUCTION

Last Updated: Sunday, 22 August 2021 at 11:38

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

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

Any operating system that can run Python 3 and has the Qt 5 libraries should be able to run novelWriter. It runs fine on Linux, Windows and macOS, and users have tested it on other platforms too. novelWriter can be run directly from the Python source, installed from the pip tool.

You can also download a minimal archive package of novelWriter tailored for your operating system. This package can be extracted anywhere on your computer, and a setup script can be run to create the necessary icons and file associations. See *Getting Started*, or one or the setup instructions for your operating system for further details.

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

Useful Links

- Website: https://novelwriter.io

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

- 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

INTRODUCTION

INTRODUCTION

novelWriter is a simple, multi-document plain text editor using a markup syntax inspired by markdown to apply simple formatting to the text. It is designed for writing novels, so the formatting features are limited.

The idea is to let the user focus on writing instead of spending time messing with the formatting of headers and text. Therefore you cannot change the look of the text in the editor window. Instead, you provide formatting tags where they're needed, like for instance which text is a header, where you want text bolded or italicised, and what alignment you want for paragraphs. The actual formatting is then added to the text when you run the *Build Novel Project* tool.

A document viewer to the right of the editor can also show a renderred version of any document if you want to inspect the result, or just want to keep a second document open for reference when you're writing.

You can split your novel project up into as many individual files as you want to. The files are glued together when you build the project, in the top-to-bottom order in which they appear in the project tree. Splitting the project up into chapter and scene files means you can easily reorder them using the drag and drop feature. More details about how projects are structured is covered in *Novel Structure*.

In addition to novel text documents, the project can contain notes on the various plot elements, characters, locations, etc, that make up the story. These notes are organised in a set of category-specific top-level folders referred to as *Root Folders*. Each note can be assigned one or more tags (one tag is allowed for each heading in the note), and these tags can be referenced from within the novel documents and other notes.

These tags make it possible to inter-link documents, and you can also generate an overview of the entire novel project and how the various documents and plot elements are interconnected. The tag and reference syntax is covered in *Novel Projects* and *Project Notes*.

These features are available through special meta keywords described in *Tag References*. Syntax highlighting is provided to make it easier to verify that the markdown tags are used correctly.

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

1.1 Design Philosophy

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

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

Tip: If you do need to align information in rows and columns in your notes, you can achieve this with tabs and line breaks. The tab stop width can be specified in *Preferences*.

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

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

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

The main window is split in two, or optionally three, panels. The left-most panel contains the project tree and all the documents in your project. The second panel is the document editor. An optional third panel is a document viewer which can view any document in your project independently of what is open in the document editor. It is not intended as a preview window, although you can use it for this. The main purpose of the viewer is for viewing your notes next to your editor while you're writing.

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

1.2 Project Layout

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

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

- **H1** is used for the book title, and for partitions.
- **H2** is used for chapter tiles.

- **H3** is used for scene titles – optionally replaced by separators.

- **H4** is for section titles within scenes, if such granularity is needed.

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

1.3 Project Export

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

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

You can also export the content of the project to a JSON file. This is useful if you want to write your own processing script in for instance Python as the entire novel can be read into a Python dictionary with a couple of lines of code. The JSON file can be populated either with HTML formatted text, or with the raw text as typed into the novel documents. See *Additional Export Options* for more details.

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

1.4 Screenshots

novelWriter with default system theme:
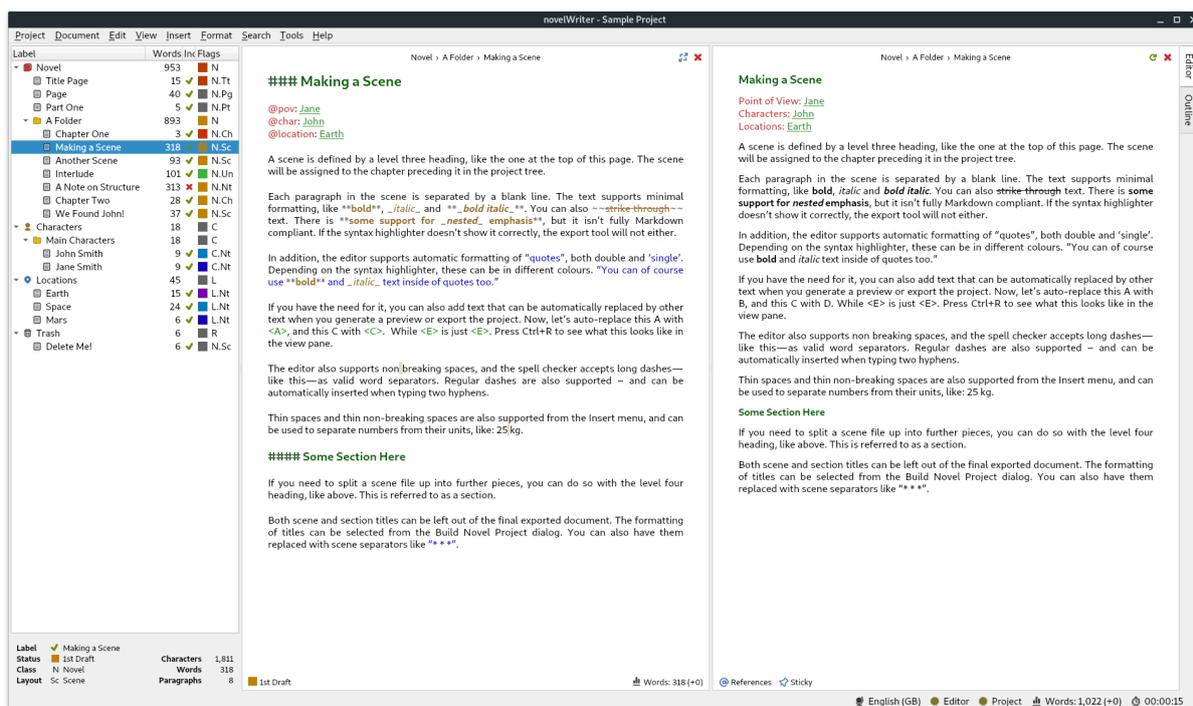
novelWriter with dark theme:
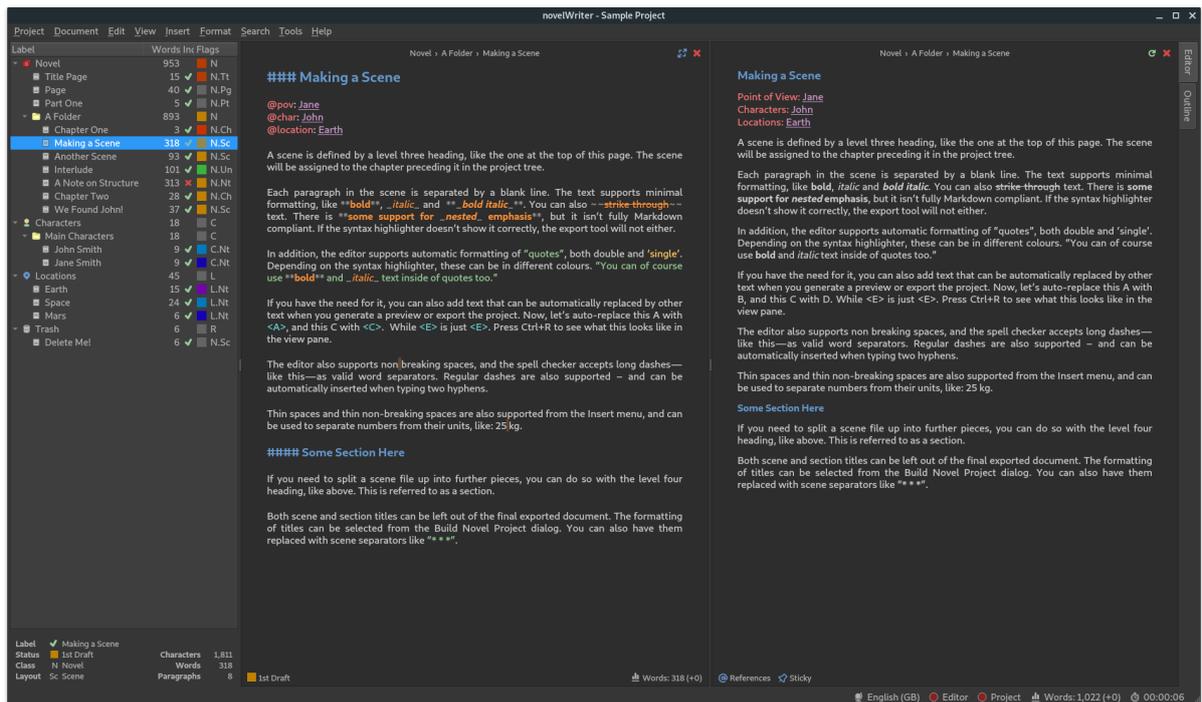
GETTING STARTED

This section contains brief guides to how you can get novelWriter running on your computer. These are the methods currently supported by the developer. Packages may also be available in other package managers, but those are not managed by the developer. No installers are provided at this time, but it is fairly straightforward to set up novelWriter with the provided install scripts.

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

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

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

2.1 Dependencies

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

The following Python packages are needed to run novelWriter:

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

PyQt/Qt should be at least 5.3, but ideally 5.10 or higher for nearly all features to work. For instance, searching using regular expressions with full Unicode support requires 5.13. There is no known minimum version requirement for package `lxml`, but the code was originally written with 4.2, which is therefore set as the minimum. It may work on lower versions. You have to test it.

Optionally, a package can be installed to interface with the Enchant spell checking libaries, but this isn't strictly required. If no external spell checking library is available, novelWriter falls back to using the internal `difflib` of Python to check spelling. This is a slower and less sophisticated spell checker

than the full spell checking libaries. The spell check library must be at least 3.0 to work with Windows. On Linux, 2.0 also works fine.

If you do use the internal spell checker, you also have to provide a text file with a list of all words correctly spelled. One word on each line. Dictionary files for English is included with novelWriter. For other languages, you will need to add more files yourself.

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

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

2.2 Installing from Source or PyPi

You can download the latest version of novelWriter from the source repository on GitHub. You can also install novelWriter from PyPi.

If you're running novelWriter from source, the main setup script has a number of options that may be useful to you. You can list them by running:

```
python setup.py --help
```

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

```
pip install novelwriter
```

To upgrade an existing installation, use:

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

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

```
novelWriter
```

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

2.3 Building the 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 `nw/assets/i18n` folder, and will have the `.qm` file extension.

You can build the `.qm` files with:

```
python3 setup.py qtlrelease
```

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

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

2.4 Building the Documentation

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

- `python3-sphinx`
- `python3-sphinx-rtd-theme`

Or from PyPi:

```
pip install sphinx sphinx-rtd-theme
```

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

```
make html
```

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

You can also build the PDF documentation locally using the setup script:

```
python setup.py pdfdocs
```

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

SETUP ON LINUX

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

3.1 Running from Source

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

3.1.1 Step 1: Installing Dependencies

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

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

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

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

3.1.2 Step 2: Install Package (Optional)

You can install novelWriter to the default location for Python packages using `setuptools`. This step is optional as you can also just put the novelWriter program folder wherever you like yourself. For instance in `/opt/novelWriter`, and then run Step 3 to set up icons and launcher.

To install novelWriter to the default location requires that you have `setuptools` installed on your system. If you don't have it installed, it can usually be installed from your distro's repository. For Debian and Ubuntu this is achieved with:

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

The package is also available from PyPi:

```
pip3 install --user setuptools
```

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

```
./setup.py install --user
```

This should install novelWriter as `~/.local/bin/novelWriter`. If you instead want to install for all users, i.e. as `/usr/local/bin/novelWriter`, run:

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

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

3.1.3 Step 3: Create Launcher Icons

Regardless of where you extract or install the source files, you can set up a standard icon and a launcher. To set up this desktop launcher, the needed icons, and the project file association, run the following from inside the novelWriter folder at the installed or final location:

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

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

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

3.2 Uninstalling Icons

The steps taken by the `xdg-install` step can be reversed by running:

```
./setup.py xdg-uninstall
```

This will remove the desktop launcher and icons from the system. As above, whether this is done on the current user, or system wide, depends on whether this command is called with `sudo` or not.

SETUP ON MACOS

This is a brief guide to how you can get novelWriter running on macOS. There are currently no install package of novelWriter for macOS, so it is recommended that you download either the full source or minimal package and extract it to a practical location on your system and run it.

4.1 Running from Source

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

4.1.1 Step 1: Installing Dependencies

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

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

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

```
brew install enchant
```

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

4.1.2 Step 2: Install Package (Optional)

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

```
./setup.py install --user
```

This requires that the package `setuptools` is installed on your system. If not, it can be installed with:

```
pip3 install --user setuptools
```

This is method of install is equivalent to what the `pip` installer does. It puts novelWriter in the location on your system where Python packages are usually kept. This is not really the best suited location for

a GUI application like novelWriter, so you may instead copy the entire source to a suiteable location yourself.

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

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

SETUP ON WINDOWS

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

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

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

5.1 Using the Installer

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

Please use the "Running from Source" option below instead. It has been improved and a script has been added that does nearly the same thing.

5.2 Running from Source

To run novelWriter from source, download the latest source zip file from the release page on GitHub, or from the main website. You can download the "Minimal Windows" zip file. It is a minimal version of the source that contains only the files needed for running on Windows. You can also download the full source if you wish.

In order to make novelWriter run on your system, you must first have Python installed (Step 1). Thereafter, a script will do the rest of the job (Step 2). Alternatively, you can run this step manually from the command prompt if you wish to.

Tip: If your system already has all the dependencies installed, you can run novelWriter directly from the extracted folder by double-clicking the `novelWriter.pyw` file. However, running the script also adds start icons.

5.2.1 Step 1: Installing Python

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

Make sure you select the "Add Python to PATH" option during installation, otherwise the `python` command will not work in the command line window.
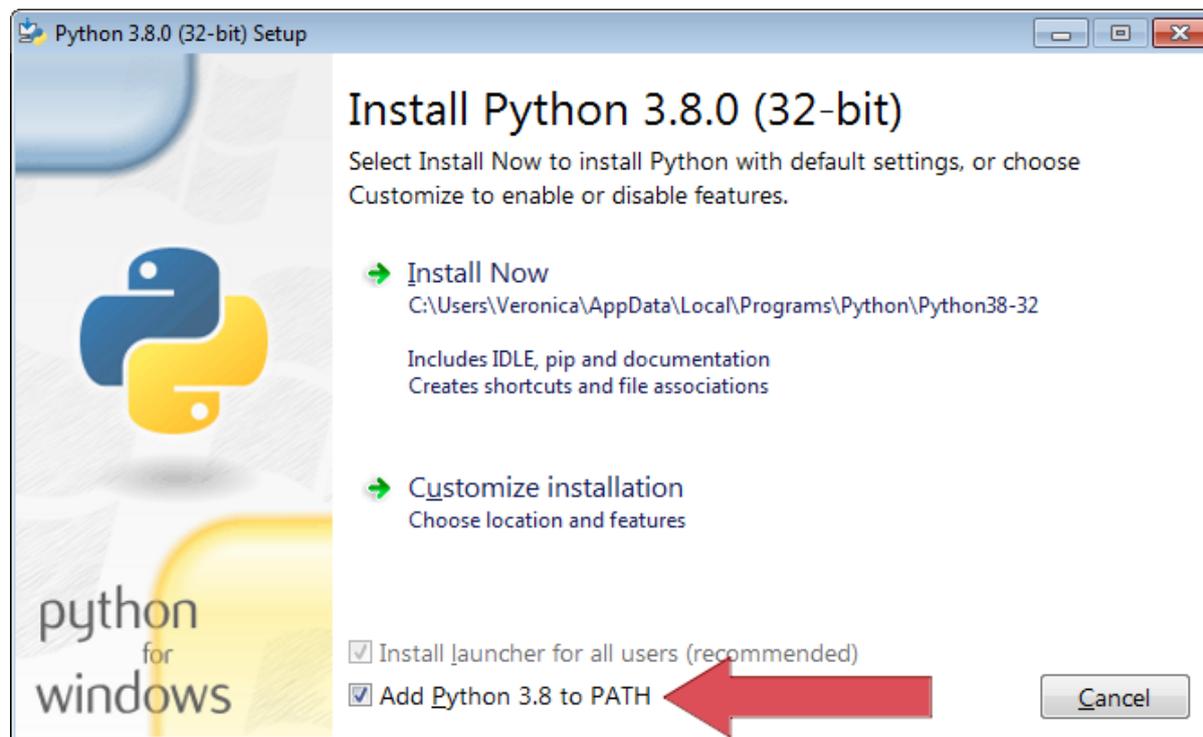
5.2.2 Step 2: Dependencies and Icons

Alternative A: By Script

Open the folder where you extracted novelWriter, and double-click the file named `windows_install.bat`. This should open a command line window and run the setup script to install dependencies, and add desktop and start menu icons.

Note: If you downloaded the full source package, the file may be in the `setup` subfolder.

The script will also check that it can find Python on your system and alert you if it cannot run it. If you are sure you have installed it, but the script cannot find it, you probably didn't install it with the "Add Python to PATH" option mentioned in Step 1.

Note: If you upgrade Python to a newer version and the path to `pythonw.exe` has therefore changed, you may need to run this script again. You can also run it to upgrade dependencies to the latest version.

Alternative B: Manual Installation

The above alternative can also be run manually.

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

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

```
pip install --user pywin32 -r requirements.txt
python setup.py win-install
```

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

5.3 Uninstalling

Alternative A: By Script

Open the folder where you keep the novelWriter files, and double-click the file named `windows_uninstall.bat`. This should open a command line window and run the setup script to remove the main dependency packages and remove desktop and start menu icons.

Note: If you downloaded the full source package, the file may be in the `setup` subfolder.

If you plan to also remove Python from your system, you must run the above script first as it needs Python in order to run.

Note: Due to limitations of the `pip` installer, dependencies of the dependencies will not be removed, only the ones the setup script directly installed.

Alternative B: Manual Uninstallation

Like for the install process, the script just runs two commands. You can of course run them yourself if you wish. They are:

```
python setup.py win-uninstall
pip uninstall -r requirements.txt
```

There may be other packages on your system installed by `pip`. To list all packages, run:

```
pip freeze --user
```

USER INTERFACE OVERVIEW

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

6.1 The Project Tree

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

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

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

Column 3 The third column indicates whether the document is included in the final project build or not. You may want to filter out documents that you no longer want to keep in the final manuscript, but want to keep in the project for reference.

Column 4 The fourth column shows the user-defined status or importance labels you've assigned to each project item. See *Document Importance and Status* for more details. By default, both the icon and the label text is shown, but you can turn off the text label from *Preferences*. If the text label is off, the text will instead appear in a tooltip when you hover your mouse over the icon.

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

The label, status or importance setting, the layout, and the include flag can all be edited using the *Item Settings* dialog box. The dialog can be opened from the *Project* menu, or by pressing F2 with the item selected.

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

6.1.1 The Novel Tree

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

Note: You cannot reorganise the entries in the novel tree, or add any new ones, as that would imply restructuring the content of the document files.

6.1.2 Document Importance and Status

Each document or folder in your project can have either a "Status" or "Importance" flag set. These are flags that you control and define yourself. To modify the labels, go to their respective tabs in *Project Settings*.

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

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

6.1.3 Project Tree Drag & Drop

The project tree allows drag & drop to a certain extent. This feature is primarily intended for reordering your documents within each root folder. Moving a document in the project tree will also put it in a different place when you build the novel project.

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

Documents and their folders can be rearranged freely within their root folders. Novel documents cannot be moved out of the *Novel* folder, except to *Trash* and the *Outtakes* folders. Notes can be moved freely between all root folders, but keep in mind that if you move a note into a *Novel*, its "Importance" setting will be reset to the default "Status" setting. See *Document Importance and Status*.

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

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

6.2 Editing and Viewing Documents

To edit a document, double-click it in the project tree, or press the `Return` key while having it selected. This will open the document in the document editor. The editor uses a markdown-like syntax for some features, and a novelWriter-specific syntax for others. The syntax format is described in the *Formatting Your Text* section below. The editor has a maximise button (toggles the *Focus Mode*) and a close button in the top–right corner. On the top–left side you will find an edit button that opens the *Item Settings* dialog for the currently open document, and a search button to open the search dialog.

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

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

Any tag reference in the editor can be opened in the viewer by moving the cursor to the label and pressing `CtrlReturn`. You can also control-click them with your mouse. In the viewer, the references become clickable links. Clicking them will replace the content of the viewer with the content of the document the reference points to.

The document viewer keeps a history of viewed documents, which you can navigate through with the arrow buttons in the top–left corner of the viewer. If your mouse has back and forward navigation buttons, these can be used as well. They work just like the backward and forward features in a browser.

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

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

6.2.1 Search & Replace

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

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

There are a number of settings for the search tool available as toggle switches above the search box. They allows you to search for, in order: matched case only, whole word results only, search using regular

---

expressions, loop search when reaching the end of the document, and move to the next document when reaching the end. There is also a switch that will try to match the case of the word when the replacement is made. That is, it will try to keep the word upper, lower, or capitalised to match the word being replaced.

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

6.3 Auto-Replace as You Type

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

Tip: If you don't like auto-replacement, all symbols inserted by this feature are also available in the *Insert* menu, and via convenient *Insert Shortcuts*. You may also be using a Compose Key setup, which means you may not need the auto-replace feature.

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

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

6.4 Project Outline View

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

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

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

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

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

The information viewed in the outline is based on the project's main index. While novelWriter does its best to keep the index up to date when 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 `F10`. You can also enable automatic updating in the *Tools* menu, which will trigger an update whenever the index is updated and the *Outline* tab is active. You may want to disable this feature if your project is very large,

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

FORMATTING YOUR TEXT

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

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

7.1 Headings

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

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

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

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

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

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

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

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

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

7.2 Text Paragraphs

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

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

- A non-breaking space can be inserted with `CtrlK`, `Space`.

- Thin spaces are also supported, and can be inserted with `CtrlK`, `ShiftSpace`.

- Non-breaking thin space can be inserted with `CtrlK`, `CtrlSpace`.

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

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

Tip: Non-breaking spaces are the correct type of space to separate a number from its unit. Generally, it prevents the line wrapping algorithms from adding line breaks where it shouldn't.

7.3 Text Emphasis

A minimal set of text emphasis styles are supported.

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

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

~~text~~ Strikethrough text.

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

In addition, the following rules apply:

1. The emphasis and strikethrough formatting tags do not allow spaces between the words and the tag itself. That is, `**text**` is valid, `**text **` is not.

2. More generally, the delimiters must be on the outer edge of words. That is, `some **text in bold** here` is valid, `some** text in bold** here` is not.

3. If using both ** and _ to wrap the same text, the underscore must be the inner wrapper. This is due to the underscore also being a valid word character, so if they are on the outside, they violate rule 2.

4. Text emphasis does not span past line breaks. If you need to add emphasis to multiple lines or paragraphs, you must apply it to each of them in turn.

7.4 Comments and Synopsis

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

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

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

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

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

7.5 Tags and References

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

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

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

7.6 Paragraph Alignment and Indentation

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

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

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

Examples:

Table 1: Text Alignment and Indentation

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

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

7.7 Vertical Space and Page Breaks

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

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

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

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

KEYBOARD SHORTCUTS

Most features in novelWriter are available as keyboard shortcuts.

8.1 Main Shortcuts

The main shorcuts are as follows:

Table 1: Keyboard Shortcuts

Shortcut	Description
`Alt1`	Switch focus to the project tree. On Windows, use `CtrlAlt1`.
`Alt2`	Switch focus to document editor. On Windows, use `CtrlAlt2`.
`Alt3`	Switch focus to document viewer. On Windows, use `CtrlAlt3`.
`Alt4`	Switch focus to outline view. On Windows, use `CtrlAlt4`.
`AltLeft`	Move backward in the view history of the document viewer.
`AltRight`	Move forward in the view history of the document viewer.
`Ctrl.`	Open menu to correct word under cursor.
`Ctrl,`	Open the *Preferences* dialog.
`Ctrl/`	Toggle block format as comment.
`Ctrl0`	Remove block formatting for block under cursor.
`Ctrl1`	Change block format to header level 1.
`Ctrl2`	Change block format to header level 2.
`Ctrl3`	Change block format to header level 3.
`Ctrl4`	Change block format to header level 4.
`Ctrl5`	Change block alignment to left-aligned.
`Ctrl6`	Change block alignment to centred.
`Ctrl7`	Change block alignment to right-aligned.
`Ctrl8`	Add a left margin to the block.
`Ctrl9`	Add a right margin to the block.
`CtrlA`	Select all text in the document.
`CtrlB`	Format selected text, or word under cursor, with strong emphasis (bold).
`CtrlC`	Copy selected text to clipboard.
`CtrlD`	Strikethrough selected text, or word under cursor.
`CtrlE`	If in the project tree, edit a document or folder settings. (Same as `F2`.)
`CtrlF`	Open the search bar and search for the selected word, if any is selected.
`CtrlG`	Find next occurrence of search word in current document. (Same as `F3`.)

Table 1 – continued from previous page

Shortcut	Description
CtrlH	Open the search and replace bar and search for the selected word, if any is selected. (On Mac, this is `Cmd=`.)
CtrlI	Format selected text, or word under cursor, with emphasis (italic).
CtrlK	Activate the insert commands. The commands are listed in *Insert Shortcuts*.
CtrlN	Create new document.
CtrlO	Open selected document.
CtrlQ	Exit novelWriter.
CtrlR	If in the project tree, open a document for viewing. If the editor has focus, open current document for viewing.
CtrlS	Save the current document in the document editor.
CtrlV	Paste text from clipboard to cursor position.
CtrlW	Close the current document in the document editor.
CtrlX	Cut selected text to clipboard.
CtrlY	Redo latest undo.
CtrlZ	Undo latest changes.
CtrlF7	Toggle spell checking.
CtrlF10	Toggle automatic updating of project outline.
CtrlUp	Move item one step up in the project tree.
CtrlDown	Move item one step down in the project tree.
CtrlDel	Delete next word in editor.
CtrlBackspace	Delete previous word in editor.
Ctrl'	Wrap selected text, or word under cursor, in single quotes.
Ctrl"	Wrap selected text, or word under cursor, in double quotes.
CtrlRetrun	Open the tag or reference under the cursor in the Viewer.
CtrlShift,	Open the *Project Settings* dialog.
CtrlShift/	Remove block formatting for block under cursor.
CtrlShift1	Replace occurrence of search word in current document, and search for next occurrence.
CtrlShiftA	Select all text in current paragraph.
CtrlShiftG	Find previous occurrence of search word in current document. (Same as `ShiftF3`.)
CtrlShiftI	Import text to the current document from a text file.
CtrlShiftN	Create new folder.
CtrlShiftO	Open a project.
CtrlShiftR	Close the document viewer.
CtrlShiftS	Save the current project.
CtrlShiftW	Close the current project.
CtrlShiftZ	Undo move of project tree item.
CtrlShiftDel	If in the project tree, move a document to trash, or delete a folder.
F1	Open the online user manual.
F2	If in the project tree, edit a document or folder settings. (Same as `CtrlE`)
F3	Find next occurrence of search word in current document. (Same as `CtrlG`)
F5	Open the *Build Novel Project* dialog.
F6	Open the *Writing Statistics* dialog.

Table 1 – continued from previous page

Shortcut	Description
F7	Re-run spell checker.
F8	Activate *Focus Mode*, hiding the project tree and document viewer.
F9	Re-build the project index.
F10	Re-build the project outline.
F11	Activate full screen mode.
ShiftF1	Open the local user manual (PDF) if it is available.
ShiftF3	Find previous occurrence of search word in current document. (Same as CtrlShiftG.)
ShiftF6	Open the *Project Details* dialog.
Return	If in the project tree, open a document for editing.

Note: On macOS, replace Ctrl with Cmd.

8.2 Insert Shortcuts

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

Table 2: Keyboard Shortcuts

Shortcut	Description
CtrlK, –	Insert a short dash (en dash).
CtrlK, _	Insert a long dash (em dash).
CtrlK, Ctrl_	Insert a horizontal bar (quotation dash).
CtrlK, ~	Insert a figure dash (same width as a number).
CtrlK, 1	Insert a left single quote.
CtrlK, 2	Insert a right single quote.
CtrlK, 3	Insert a left double quote.
CtrlK, 4	Insert a right double quote.
CtrlK, '	Insert a modifier apostrophe.
CtrlK, .	Insert an ellipsis.
CtrlK, Ctrl'	Insert a prime.
CtrlK, Ctrl"	Insert a double prime.
CtrlK, Space	Insert a non-breaking space.
CtrlK, ShiftSpace	Insert a thin space.
CtrlK, CtrlSpace	Insert a thin non-breaking space.
CtrlK, *	Insert a list bullet.
CtrlK, Ctrl–	Insert a hyphen bullet (alternative bullet).
CtrlK, Ctrl*	Insert a flower mark (alternative bullet).
CtrlK, %	Insert a per mille symbol.
CtrlK, CtrlO	Insert a degree symbol.
CtrlK, CtrlX	Insert a times sign.
CtrlK, CtrlD	Insert a division sign.

Table 2 – continued from previous page

Shortcut	Description
CtrlK, G	Insert a @tag keyword.
CtrlK, V	Insert a @pov keyword.
CtrlK, F	Insert a @focus keyword.
CtrlK, C	Insert a @char keyword.
CtrlK, P	Insert a @plot keyword.
CtrlK, T	Insert a @time keyword.
CtrlK, L	Insert a @location keyword.
CtrlK, O	Insert an @object keyword.
CtrlK, E	Insert an @entity keyword.
CtrlK, X	Insert a @custom keyword.

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.

9.1 Special Notes on Symbols

Some additional notes on the available symbols.

9.1.1 Dashes and Ellipsis

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

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

9.1.2 Single and Double Quotes

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

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

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

9.1.3 Single and Double Prime

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

9.1.4 Modifier Letter Apostrophe

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

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

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

9.1.5 Special Space Symbols

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

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

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

The thin and non-breaking spaces are converted to their corresponding HTML codes on export to HTML format.

PROJECT FORMAT CHANGES

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

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

10.1 Format 1.3 Changes

This project format vas introduces in novelWriter version 1.5.

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

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

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

Title Pages

- The formatting of the level one title on the title page must be changed from `# Title Text` to `#! Title Text` in order to retain the previous functionality. See *Headings*.

- Any text that was previously centred on the page must be manually centred using the new text alignment feature. See *Paragraph Alignment and Indentation*.

Unnumbered Chapters

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

Plain Pages

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

10.2 Format 1.2 Changes

This project format was introduces in novelWriter version 0.10.

With this format, the way auto-replace entries were stored in the main project XML file changed. Opening an old project automatically converts the storage format up to and including version 1.1.1.

Format 1.2 projects can be opened without loss of information up until version 1.1.1, and if the auto-replace is not being used, can still be opened in novelWriter as of version 1.5-beta1.

10.3 Format 1.1 Changes

This project format was introduces 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.

Format 1.1 projects can be opened without loss of information up until version 1.1.1, and if the auto-replace is not being used, can still be opened in novelWriter as of version 1.5-beta1.

10.4 Format 1.0 Changes

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

Format 1.0 projects can be opened without loss of information up until version 1.1.1, and if the auto-replace is not being used, can still be opened in novelWriter as of version 1.5-beta1.

NOVEL PROJECTS

New projects can be created from the *Project* menu by selecting *New Project*. This will open the *New Project Wizard* that will assist you in creating a barebone project suited to your needs. A novelWriter project requires a dedicated folder for storing its files on the local file system. See *How Data is Stored* for further details on how files are organised.

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

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

11.1 Project Roots

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

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

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

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

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

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

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

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

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

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

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

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

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

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

11.1.1 Deleted Documents

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

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

11.1.2 Archived Documents (Outtakes)

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

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

11.1.3 Recovered Documents

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

The discovered files are scanned for meta information that give clues as to where the document may previously have been located in the project. The project loading routines will try to put them back as close as possible to this location, if it still exists. Generally, it will be appended to the end of the folder where it previously was located. If that folder doesn't exist, it will try to add it to the correct root folder. If it cannot figure out which root folder is correct, the document will be added to the *Novel* root folder. Only if the *Novel* folder is missing will it give up.

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

11.1.4 Project Lockfile

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

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

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

11.1.5 Using Folders in the Project Tree

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

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

Tip: You can use folders to sort your scene documents into chapters. You will still need to add a chapter document as the first item of your chapter folder, and the scene documents as the following items. Other ways to use folders is to make a folder for each act or part.

11.2 Project Documents

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

The layout of the document is also defined here. The two options available are *Novel Document* and *Project Note*. These behave differently when the project is built. A project note is never treated as part of the novel, no matter where in the project it is located. See *Document Layout* for more details.

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

11.2.1 Word Counts

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

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

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

11.3 Project Settings

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

11.3.1 Settings Tab

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

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

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

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

11.3.2 Status and Importance Tabs

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

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

See also *Document Importance and Status*.

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

11.3.3 Auto-Replace Tab

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

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

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

11.4 Backup

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

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

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

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

11.5 Writing Statistics

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

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

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

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

NOVEL STRUCTURE

This section covers the structure of a novel project.

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

12.1 Importance of Headings

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

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

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

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

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

Header Level 2 This header level signifies a chapter level partition. Each time you want to start a new chapter, you must add such a heading. If you choose to split your manuscript up into one document per scene, you need a single chapter document with just the heading. You can of course also add a synopsis and reference keywords to the chapter document. If you want to open the chapter with a quote or other introductory text that isn't part of a scene, this is also where you'd put that text.

Header Level 3 This header level signifies a scene level partition. You must provide a title text, but the title text can be replaced with a scene separator or just skipped entirely when you export your manuscript.

Header Level 4 This header level signifies a sub-scene level partition, usually called a "section" in the documentation and the user interface. These can be useful if you want to change tag references mid-scene, like if you change the point-of-view character. You are free to use sections as you wish, and can filter them out of the final manuscript just like with scene titles.

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

12.1.1 Novel Title and Front Matter

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

```
#! My Novel
```

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

If you want an additional page inserted 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 when the project is exported.

12.1.2 Unnumbered Chapter Headings

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

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

There is a separate formatting feature for such chapters in the *Build Novel Project* tool as well. See the *Exporting Projects* page for more details.

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

12.2 Tag References

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

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

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

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

@focus The character that has the focus for the current section. This can be used in cases where the focus is not a point-of-view character. The target must be a note tag in the *Character* type root folder.

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

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

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

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

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

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

@custom Custom references in the current section. The target must be a note tag in a *Custom* type root folder. You can add more than one Custom folder, but they all use the same reference keyword.

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

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

12.3 Document Layout

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

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

Depending on which icon theme you're using, the project tree can distinguish between the different layouts and header levels of the documents to help indicate which are project notes and which are novel documents containing a partition, chapter, or scene. If the icon theme you've selected doesn't show a difference, you can still see the layout description in the details panel below the project tree.

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

PROJECT NOTES

novelWriter doesn't have a database and complicated forms for filling in details about plot elements, characters, and all sorts of additional information that isn't a part of the novel text itself. Instead, all such information is saved in notes that are written and maintained just like all other text in your project.

The relation between all these additional elements is extracted from the documents and notes by the project indexer, based on the tags and references you set within them.

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

13.1 Tags in Notes

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

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

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

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

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

EXPORTING PROJECTS

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

14.1 Header Formatting

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

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

`%ch%` This will be replaced by a chapter number. The number is incremented by one each time the build tool sees a new heading of level two in a document, unless the heading formatting code has the added `!`. In the latter case, the counter is *not* incremented. This is useful for for instance Prologue and Epilogue chapters.

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

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

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

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

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

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

Note: Header formatting only applies to novel documents. Headings in notes will be left as-is.

Example

- The format `%title%` just reproduces the title you set in the document.

- The format `Chapter %ch%:  %title%` produces something like "Chapter 1: My Chapter Title".

- The format `Scene %ch%.%sc%` produces something like "Scene 1.2" for scene 2 in chapter 1.

14.2 Scene Separators

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

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

14.3 File Selection

Which documents and notes are selected for export can be controlled from the options on the left side of the dialog window. The switch for *Include novel files* will enable or disable inclusion of novel documents, and the switch for *Include note files* will do the same for project notes. This allows for exporting just the novel, just your notes, or both, as you wish.

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

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

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

14.4 Printing

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

14.5 Export Formats

Currently, six formats are supported for exporting.

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

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

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

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

14.6 Additional Export Options

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

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

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

HOW DATA IS STORED

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

15.1 Project Structure

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

15.1.1 Main Project File

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

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

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

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

15.2 Project Documents

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

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

The reason for this cryptic file naming is to avoid issues with file naming conventions and restrictions on different operating systems, and also to have a file name that does not depend on what the user names the document within the project, or changes it to.

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

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

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

15.2.1 The File Saving Process

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

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

15.3 Project Meta Data

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

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

15.3.1 The Project Index

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

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

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

15.3.2 Cached GUI Options

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

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

15.3.3 Session Stats

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

15.4 Project Cache

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

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

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.

16.1 Dependencies

The dependencies for running the tests can be installed with:

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

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

16.2 Simple Test Run

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

```
pytest
```

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

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

```
xvfb-run pytest
```

16.3 Advanced Options

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

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

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

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

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

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

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

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

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

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

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

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

INDICES AND TABLES

- genindex
- modindex
- search