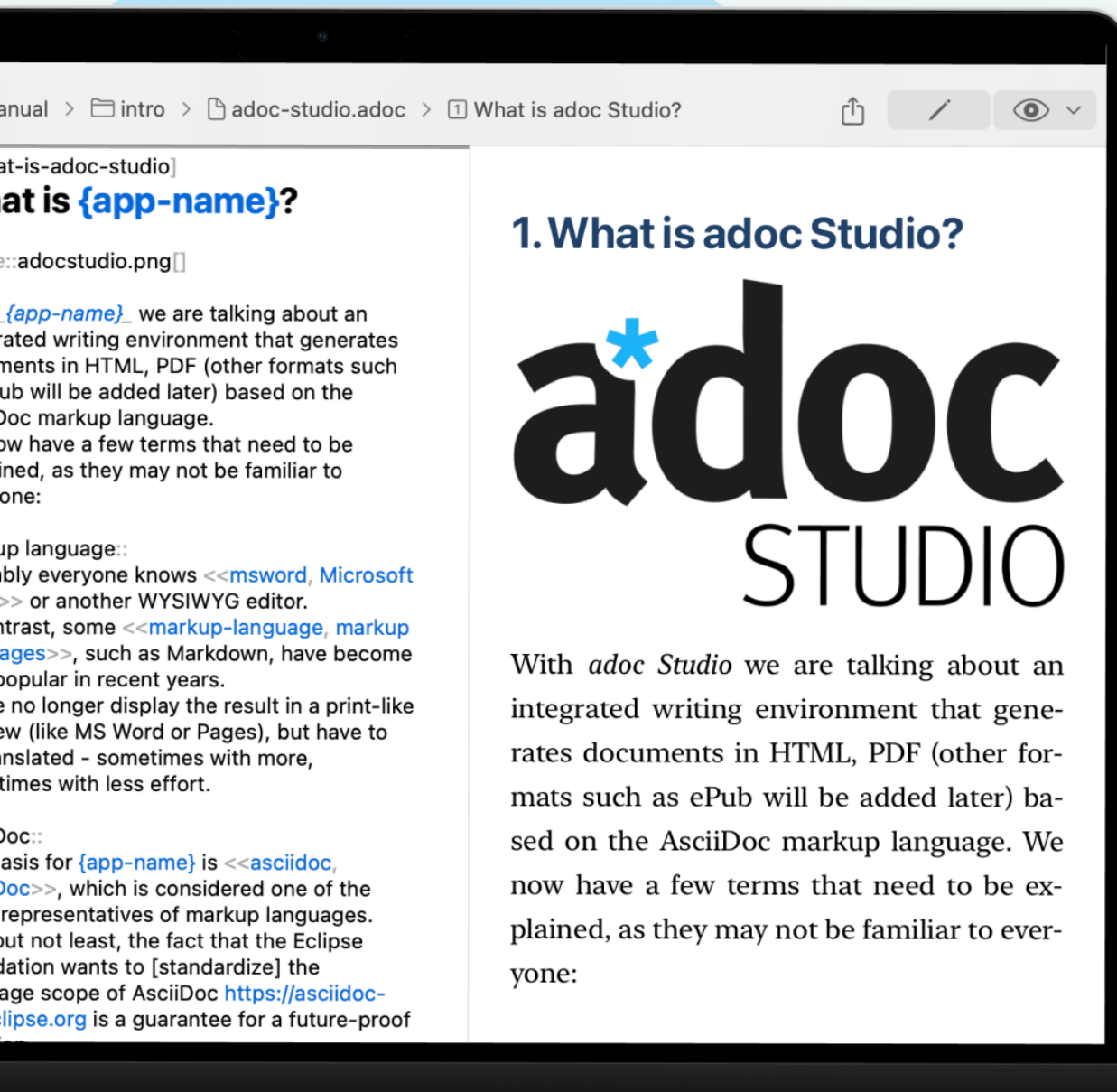


Working with adoc Studio

on the Mac



Working with adoc Studio

ProjectWizards GmbH, v1.0



This document guides you directly into working with adoc Studio. It explains the structure of the app, introduces the most important commands, and provides numerous tips for deeper exploration of the documentation with adoc Studio.


Additionally, the source code of this documentation gives you an example of how to set up a project in adoc Studio. The user manual can be accessed via the main menu:  [Help > User Manual](#).

Table of Contents

What is adoc Studio?	1
What is a Markup Language?	2
Considering Word, Pages, or Google Docs	2
Installation.....	3
Trial Phase & Licensing	4
Getting to Know adoc Studio	5
The User Interface	5
The Project Navigator	6
Saving a Project.....	12
Opening Additional Windows for a Project	12
External Projects.....	12
The Section Navigator	12
The Search Navigator	13
The Problem Navigator	13
Writing and Navigating within the Text	14
Input Aids.....	17
Application Settings.....	20
General.....	20
File Access.....	21
Editor Styles	22
Product Styles.....	23
The AsciiDoc Markup Language.....	25
What is AsciiDoc?	25
Writing Text.....	25
Formatting	26
Headings.....	27
Table of Contents.....	28
Links	29
Images	30
Lists	32
Block Elements.....	33
Admonitions.....	33
Tables.....	34
User Interface	35
Comments	36
Attributes	36
Conditional Statements.....	40
Includes.....	40
Exporting Documents.....	42

The Preview.....	42
Features for Editor and Preview	43
Exporting	44
Products	46
There's Still Something Missing.....	49
Appendix A: Support.....	50
System Requirements	50
First Aid.....	50
Help with Templates and Styles	51
Appendix B: Attributes.....	52
adoc Studio Attributes	52
Intrinsic Attributes	52
Compliance Attributes.....	54
Localization and numbering attributes	55
Styling and Layout.....	56
Document metadata attributes.....	58
Section title and table of contents attributes.....	60
General content and formatting attributes	62
Image and icon attributes	63
HTML styling attributes	64
Appendix C: Regular Expressions	65
Regular Expression Metacharacters.....	65
Regular Expression Operators	68
Template Matching Format	70
Flag Options.....	70
Appendix D: Glossary.....	72
Inhalt	72

Notation in this document

You will find some notes in the text that will give you special help.



Additional notes are displayed as a note.



Whenever there is an opportunity for a special hint, you will receive a tip.







The bell appears to alert you to special features.



Whenever you need to take special care before performing an action, this warning will appear.



Attention, here we try to draw your attention to existing dangers, damage or consequences.

You can recognize menu commands by the structure of the menu:  **File > Export > Products...**. If there is a keyboard shortcut, you can recognize it by this representation  +  + .

Legal information

Please note that operating instructions, manuals and software are protected by copyright. Copying, duplicating, translating or converting into any electronic medium or machine-readable form in whole or in part is not permitted without the prior written consent of ProjectWizards. All other rights to the software are set out in the supplied [license terms](#)⁷. The rights to other trademarks and product names mentioned in this manual belong to their owners and are hereby acknowledged. The naming of products that are not from ProjectWizards is for information purposes only and does not constitute advertising. ProjectWizards assumes no liability with regard to the selection, performance or usability of these products.

What is adoc Studio?

With adoc Studio, you can create technical documentation efficiently and effortlessly. The app supports you in creating extensive and structured documents. Once started, the cursor blinks in the editor, and every word you write instantly appears in the preview on the right. This intuitive workflow ensures high productivity. Whether on the Mac, iPad, or iPhone, you always have a consistent and user-friendly view.



adoc Studio uses an integrated writing environment based on the markup language [AsciiDoc](#) and produces documents in [HTML](#) and [PDF](#) formats.

Markup Language: Most people are familiar with Microsoft Word or another text editor. In contrast, markup languages like Markdown have become increasingly popular in recent years. These do not present the result like Microsoft Word or Apple Pages in a print-like preview. Instead, they need to be translated, which requires varying amounts of effort.

AsciiDoc: adoc Studio is based on [AsciiDoc](#), a well-established markup language. The decision by the Eclipse Foundation to standardize the scope of AsciiDoc ensures a future-proof choice. We chose AsciiDoc because it is easy for beginners to learn and offers an extensive set of commands for professionals.

Integrated Writing Environment: An efficient and user-friendly working environment requires a program that integrates all necessary functions. With adoc Studio, you can [write texts](#), organize them in [projects](#), use a [preview](#) for initial checks, and export the finished documents either individually or as entire [products](#).

+ adoc Studio offers extensive [customization options](#). You can customize the appearance of the editor and choose an output style. We provide some styles and regularly release more on our [website](#) ↗.

+ With adoc Studio, you can fully concentrate on the structure and content of your documents. You can **emphasize** important statements and make *highlighted* points. Additionally, the app supports the use of emojis 🤪

What is a Markup Language?

Historically, the term *markup* originates from typography. In the art of printing, it referred to highlighting parts of text through different font variations such as *italic* and **bold** typefaces, SMALL CAPS, uppercase letters (ALL CAPS), letter spacing, underlining, or **coloring** the text, as well as using different font sizes and styles. It also included manually marking corresponding places in the manuscript (from [Wikipedia](<https://de.wikipedia.org/wiki/Schriftauszeichnung>))[↗]. These techniques were adapted to the computer era and software, leading to the development of modern markup languages.

You may have heard of the language *Markdown* or the phrase “the text is written in Markdown.” You’re on the right track. On one hand, plain text is presented, meaning letters without formatting. On the other hand, there are **marks** that highlight the text specifically: indicating what is a heading, what is bold, italic, or a link, and so on.

There are numerous [markup languages](#)[↗] that are not to be confused with word processing programs like [Microsoft Word](#). The three best-known representatives of markup languages include:

- **HTML**, the language of the internet, is very widespread but often considered cumbersome.
- **TeX or LaTeX**, probably the most popular typesetting system for scientific articles, is rarely encountered in business and is considered difficult to learn.
- **Markdown** is very easy to use but too simple, as it lacks many features needed for technical documentation. A [detailed comparison](#)[↗] of Markdown and AsciiDoc offers further insights.

Due to the disadvantages of the above languages, the decision was made to use **AsciiDoc** as the markup language. At [ProjectWizards](#)[↗], all [manuals](#)[↗] and [public documents](#)[↗] have been created in this format since 2007.

AsciiDoc is a mature, lightweight, and semantic markup language mainly developed for creating technical documentation. It is ideally suited for all structured texts. The impressive number of over 10 million downloads (source: [Asciidoctor History](#)[↗]) demonstrates that help is always available. This recognition is owed to [Asciidoctor](#), the primary parser for AsciiDoc texts.



A parser is a program that analyzes text data, breaks it down into structured components, and checks it for syntactic correctness to enable further processing.

Considering Word, Pages, or Google Docs

When choosing between adoc Studio and WYSIWYG word processing programs like Microsoft Word, Apple Pages, or Google Docs, several reasons come into play:



The abbreviation "**WYSIWYG**" has been widely used since the 90s and stands for "*What you see is what you get*" in its full form. This describes the output of documents being exactly as they appear on the screen. In other words: *What you see is what you get*.

1. **Separation of Content and Presentation:** Text-based markup languages enable a clear separation of content and presentation. This ensures consistent formatting across various documents and makes it easier to uniformly adjust the appearance of documents. A document written in a markup language can be opened on any computer without special software.
2. **Portability:** Text-based markup languages are universal and cross-platform. They can be opened and edited on any device and operating system. Word files can already look very different on different computers.
3. **Version Control:** Markup languages are compatible with version control systems like Git, making it easier to collaborate and track changes in large documents. Although Word has introduced an XML format, many administrators are still reluctant to check these files into a Git repository.
4. **Reusability:** A document created in a markup language can be converted into various formats, such as HTML for a website, PDF for printing, EPUB for e-books, etc. The results in Word are often very different.
5. **Adaptability:** Text-based markup languages offer a very high level of adaptability. They allow for the customization of the appearance and behavior of documents in ways that are often not possible with standard word processing programs.
6. **Automation:** Automated scripts and tools can be used to generate, process, and manipulate documents in text-based markup languages. This can save time and reduce errors in large documents.
7. **Stability and Longevity:** Text-based markup languages are generally stable and durable, unlike proprietary file formats that can change over time.

Installation

You can download adoc Studio from the respective App Store for your Mac or your iPad or iPhone. If you are reading this text in adoc Studio now, it's already done. Congratulations!

System Requirements

The minimum system requirements for adoc Studio are:

- On the *Mac*: macOS 13 (Ventura)
- On the *iPad*: iPadOS 16
- On the *iPhone*: iOS 16

Trial Phase & Licensing

A subscription grants you access to both versions of adoc Studio, for the Mac and iOS, including iPad and iPhone. The trial starts upon the first launch of adoc Studio.

During the free 14-day trial, you have access to all features. After the trial period, you can choose between a monthly or annual subscription, with the annual subscription offering a price advantage. If you decide not to subscribe, you can still view adoc Studio projects, but editing within the app will no longer be possible.

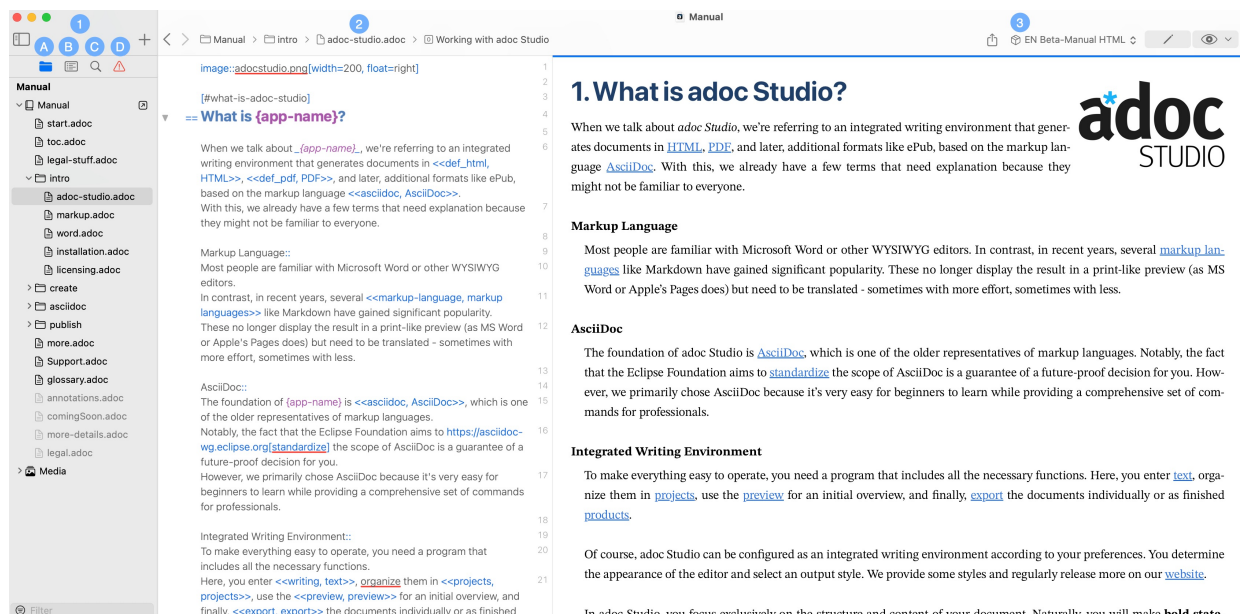
All text files are saved in plain text format (UTF-8 encoded). This ensures your data security during the trial phase. If necessary, the files can be further edited in any editor, such as [BBEdit](<https://www.barebones.com/products/bbedit/>)[↗], and processed with other processors, like [Asciidoctor](<https://asciidoctor.org>)[↗]. This is, however, intended only for the unlikely case that adoc Studio does not meet your expectations.

Getting to Know adoc Studio

In this chapter, you will get an overview of the app and the concepts that make it easier to use.













The User Interface

Upon starting adoc Studio, you will see a view like the one shown in the following image. It displays the user manual created with adoc Studio, which can be accessed through the Help menu.



The window layout of adoc Studio is divided into three sections:



1. Sidebar with:




- A. the  Project Navigator for all [files & directories](#) (keyboard shortcut:  + )
- B. the  Section Navigator for the [table of contents](#) ( + )
- C. the  Search Navigator for [searching across all files](#) ( + )
- D. the  Problem Navigator for all [warnings & error messages](#) ( + )

2. The [text editor](#)

3. The [preview](#)

At the top of the window is the toolbar with the following functions:

- Show/hide the sidebar (keyboard shortcut $\text{⌘} + 0$)
- Create a [new component](#) ($\text{⌘} + \text{⇧} + \text{N}$)
- Back and forward navigation buttons ($\text{⌘} + \text{⇧} + \text{<}$ and $\text{⌘} + \text{⇧} + \text{>}$)
- Directory tree as a popup list
- Content structure as a popup list
- Optionally: List of all [products](#)
- Show/hide editor (with  or keyboard $\text{⌘} + \text{⇧} + \text{E}$)
- Show/hide preview - (with  or keyboard $\text{⌘} + \text{⇧} + \text{E}$), including [settings](#)

Each section can be individually shown or hidden to optimally adjust to any screen size. The corresponding functions can be found in the menus  **View**,  **Editor**, and  **Preview**.


The Project Navigator

In adoc Studio, you work in a project-oriented manner. What does this mean exactly? A project includes your text and, if necessary, the associated images. However, as the text length increases, it becomes increasingly difficult to keep track. Therefore, long texts are often divided into smaller sections, which are stored in many individual files along with the images. This often leads to disorder and loss of overview.

To avoid this problem, adoc Studio offers the [composite document](#).

This chapter explains how to handle projects, files, and all associated elements.

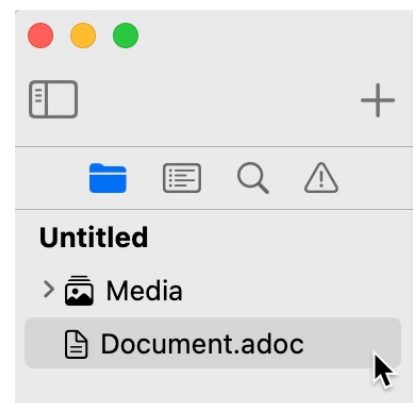
Creating a New Project

Create a new project in adoc Studio through the menu . In the [sidebar](#), you initially see two objects:

- A folder for media
- A file for the document

Whether the file extension `.adoc` is displayed as shown in the image can be set in the [settings](#) menu of adoc Studio (keyboard shortcut: $\text{⌘} + \text{,}$).

All objects (files and folders) in the sidebar can be moved. They can be nested within each other, restructured, and rearranged.



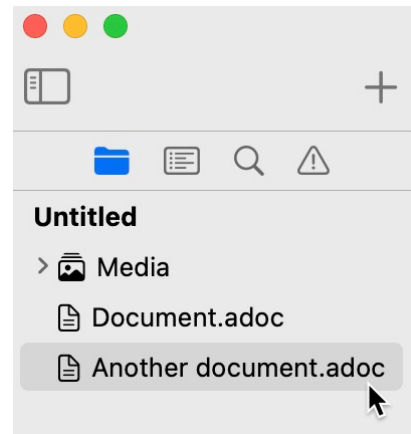


Media and images can be placed anywhere in the project. However, it is recommended to collect them in the [media folders](#). If the automatically created media folder is not enough, you can create as many additional folders as you need in your project.

Using Standalone Documents

In the next step, expand your project and add a standalone document. Use the menu **File > New**, the keyboard shortcut **⌘ + N**, or the + in the sidebar. The documents are independent of each other and exist on their own. The process is simple and logical:

1. Click on a document.
2. It is displayed in the editor for editing.
3. The preview shows the current status.



When you click on another document, it will be displayed in the editor.

This means that the contents of the second document **Another Document .adoc** will not be visible in **Document .adoc** and vice versa. There are techniques in AsciiDoc to display content from other documents in a document, such as `includes`. Read more about this later.

Using Composite Documents

Composite documents offer a structured merging of multiple documents, in contrast to [standalone documents](#). When you compile your documents into a large document, newly created files in the composite document are named as `Chapter.adoc` instead of `Document.adoc`.

In a composite document, you can use folders for structuring and nest them as deeply as you like. When you click on a folder, the preview shows all the chapters contained in it as a continuous document. The higher you go in the hierarchy, the more comprehensive the preview display will be.

When the focus in the sidebar is on a folder, the editor remains empty. To edit a chapter, select it specifically. The preview then immediately switches to the display of the text in the editor.

Using Folders for Structuring

With the step-by-step creation of new standalone files, the sidebar might eventually become cluttered. Although using the filter at the bottom of the sidebar can help you quickly find files, it resets after restarting adoc Studio.

For this reason, another solution for structuring presents itself. Similar to the operating system, you also have the option to create folders in adoc Studio. These can be named according to the rules of the operating system and offer space for numerous documents. Additionally, these folders can be created within other folders to allow even finer structuring.

The concept of individual documents remains intact. Even if you have created many folders and stored numerous files within them, they remain independent of each other. However, if you are writing a book with many chapters, you will need the [composite document](#).

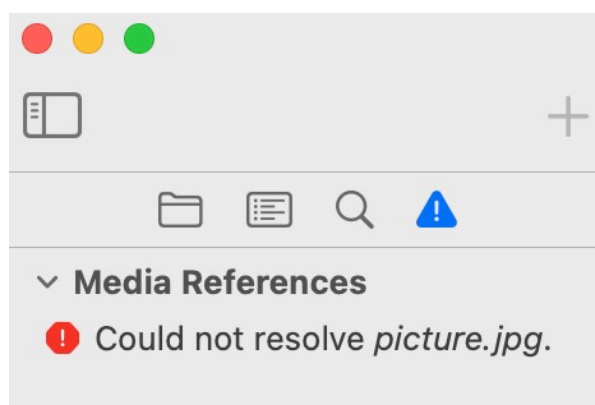
Using Media Folders

The media folder serves a special function. You can store all media, such as images, videos, etc., there by default. adoc Studio searches the project for these media in a specified order.

Search Order for Images

The line `images::image.jpg[]` prompts adoc Studio to search for the image in the following order:

1. adoc Studio first searches for the file "image.jpg" at the same level as the .adoc file.
2. If not found there, the app searches in the parent media folder.
3. If the image is not found there either, the app searches again in the next higher level media folder. This continues until the root directory of the project is reached.
4. If the image is still not found, the editor marks the file name in red, and an entry is added to the problem navigator in the [sidebar](#).



Images with Path Specifications

- The line `images::/image.jpg[]` searches for the image in the project directory where the `.adocproject` is located.
- The line `images::/Folder/image.jpg[]` searches for the image in the subdirectory `Folder`, starting from the project directory.
- The line `images::../image.jpg[]` searches for the image one level above the text file in which it is referenced.

In general, all paths can be specified as in the operating system. The root is always the project directory where your `.adocproject` file is located.



For AsciiDoc Pros:

In adoc Studio, there is also the concept of `imagesdir`. If you use this attribute in existing ([external](#)) projects, it should always precede the images and possible path specifications. Note, however, that it is not set automatically and must be adjusted manually.

Instead of:

```
image::Folder/image.jpg[]
```

use:

```
image::{imagesdir}/Folder/image.jpg[]
```

For more details and specific use cases, visit the [AsciiDoc website](#) ↗.

You can have many media folders. Here are some examples:

- A media folder with general images.
- A media folder with images of a specific language.
- A media folder with images of a specific platform.

In this project, for example, the images for the iPad are stored in a subdirectory of the media folder and are controlled by the attribute `Mac` in the [products](#). The values `:platform: iPad` or `:platform: Mac` serve as defaults. Platform-specific images are then included as follows:

```
image::Mac/new-component.jpeg
```

This way guides for Mac, iPad, and other platforms are created.



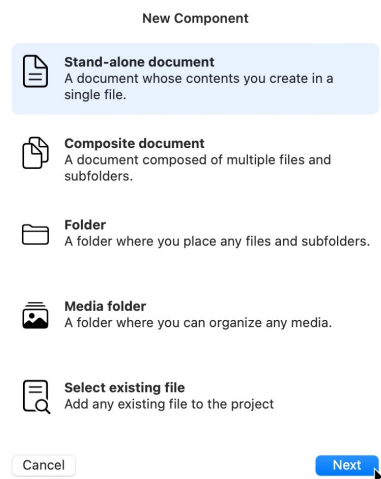
For [external projects](#), there are some additional peculiarities. More details can be found there.

Adding a New Component

To add a new component to your project, use the \oplus button in the toolbar or the keyboard shortcut $\text{⌘} + \text{⇧} + \text{N}$. The dialog adapts according to the selection in the file list as follows:

- If you have selected a standalone document, you can either create a new standalone document or a composite document.
- However, if you have selected a composite document or a chapter from it, you have the option to add new chapters.

In any case, you have the option to add new folders, media folders, or existing files.



You can drag and drop files directly from Finder into the file list to add them to your project.

Adding Existing Files

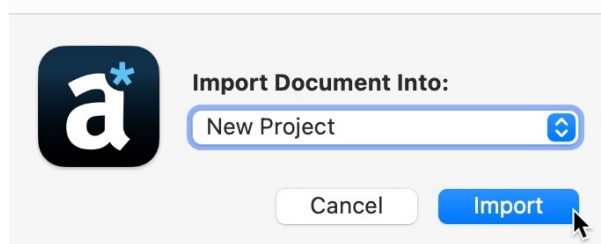
In adoc Studio, you can not only create composite documents with their chapters but also standalone documents.

A standalone document does not inherit attributes from other documents in the file list. All values must be explicitly set in these documents. Also, media folders located in composite documents are not found by a standalone document; only those outside of composite documents are detectable.

It is up to you how you want to integrate these files into your project. Below are some options:

Double-click in Finder

In Finder, you can open any .adoc file in adoc Studio by double-clicking, provided no other application has taken over the .adoc extension after installation. A dialog appears, allowing you to select the target project for the file or create a new project. The file is then copied into the selected project without being moved.







Drag & Drop from Finder

Another method to copy files into a project is by drag and drop. You can simply drag the desired file from Finder and drop it at the desired position in adoc Studio. If the target is a composite document, the file is automatically added as a part of that composite document.

Copy or Move Files in Finder

You can also directly copy or move files into the project directory in Finder. adoc Studio automatically recognizes them, and they appear in the file list of your project within a short time. There is no need to restart the app.

Menu Function


Finally, a menu function is available to add files to your projects. You can open the file selection either through the menu  **File > New > Add Files ...** or the keyboard shortcut  +  + . The selected file(s) are then added below the current selection in the file list. You can then move them to another location with the mouse.

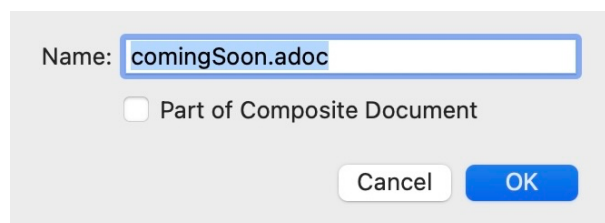


When you drag a file into a composite document, it automatically becomes a chapter of the corresponding composite document.

File Information


In case a file is part of a composite document but should not be output, adoc Studio offers a solution:

By right-clicking on the file in the file directory of the left sidebar, you open the context menu. There, you will find the entry  **Information ...**, which opens the dialog shown on the right.



Disable the "Part of the composite document" option to remove the file from the inclusion. If the file should still be used, you must include it via the **include** [command](#).

More Information

Most files and all folders in the sidebar can be modified via the menu  **Information...**. Besides renaming a file or folder, there are the following options:

For a file: **Part of a composite document** For a folder: Folder **Composite document** Media directory

These settings apply exclusively to working in adoc Studio. In macOS Finder ¹, all objects appear merely as folders and files.

1. and in all other operating systems

Saving a Project

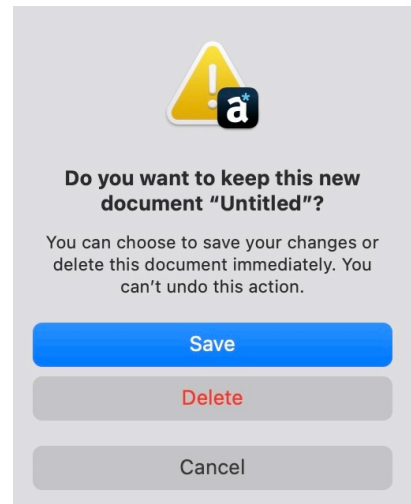
The old way

In the past, a new project was saved in the first step. You set the storage location where the project would be created. This is done via **File > Save ...** or the keyboard shortcut **⌘ + S**.

You are free to save the files of adoc Studio on your local computer, a server, or in a cloud storage. Using a version control system such as [Git](#) or [GitHub](#) is also possible.

The new way

An initial save is no longer required to avoid data loss. When you close the last project window, adoc Studio asks if the project should be saved. You then decide whether and where the project will be saved.



In adoc Studio, the app automatically saves the .adoc files at short intervals for you. If desired, you can still use the keyboard shortcut **⌘ + S**, but automatic saving is faster.

Opening Additional Windows for a Project

A project can be opened in multiple windows using the menu **Window > New Window**. This is especially helpful in a dual-monitor setup when you want to display the preview in a separate window.



On your Mac, click on the window title while holding the **⌘** key to open a menu with the path of the current project in Finder.

External Projects



The *external Projects* are not yet described. A detailed documentation will be available as soon as possible.

The Section Navigator

The second tab in the sidebar displays the structure of headings as an outline.

For composite documents, the entire table of contents is shown, regardless of which chapter you are currently working on.

For individual documents, only the respective table of contents is displayed.



To output a table of contents in the text, use the attribute `:toc:`, which is described [here](#).

The Search Navigator

To find texts in the entire project, use the third tab of the sidebar, the *Search Navigator*.

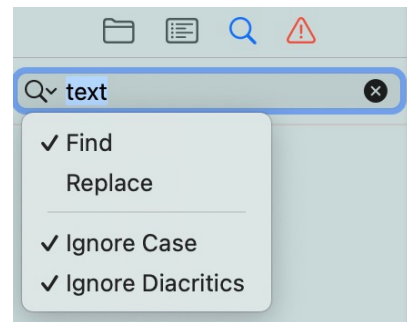
The tab with the magnifying glass is the *Search Navigator*. Use it to find text throughout the entire project.

In the context menu behind the magnifying glass, you can enable the replace function. Additionally, you can make the following settings here:

- Ignore Case,
- Ignore Diacritics


In this menu, you also activate the [regular expressions](#) for the search.

After the search, the results are grouped by the files in the project. Clicking on a file opens it in the editor and jumps to the desired position.



Special Features

If you also want to replace found text, adoc Studio offers some practical functions:

- Select individual results in the list (multiple selection with the  key is possible) to perform replacements only for the selected spots.
- Select a file in the results list to perform replacements only in that file.

The Problem Navigator

The concept for AsciiDoc and adoc Studio is to output text regardless of how many AsciiDoc errors are present in the text. The Problem Navigator is not about grammatical or spelling errors.

If there is an issue with an AsciiDoc command in the text, the error list icon appears in red:



Since not every error is of equal severity, minor issues are displayed as warnings in yellow.

The list itself is sorted by occurrence. Clicking on a message brings the editor to the corresponding position so that you can fix the error.

Writing and Navigating within the Text

Select a text file from the [file list](#). If the cursor is not already blinking in the editor window, place it in the middle area of the window with a mouse click or by pressing `^ + TAB`, and you can start writing. For information on AsciiDoc, see the [next chapter](#).

The entered text appears in the [preview](#). For longer texts, the position of both windows is synchronized, so you always see the text written on the left at the same height in the preview. The cursor position is synchronized with the position in the preview.



Note that switching to the PDF format preview can be time-consuming for longer texts. Therefore, we recommend staying with the HTML preview as it can be updated faster.

The editor can be customized according to your personal preferences (color perception, diopter strength, etc.). The [Editor Styles](#) tab in the [Application Settings](#) offers you almost endless design possibilities.

Additional Navigation

It starts with the navigation arrows to the left `<` and right `>`. The function is similar to navigation in your browser: it allows you to navigate forward and backward.

Next is a graphical block that shows you the current file and its structure.



Manual > create > writing.adoc > Writing Text and Navigating Within

Reading from left to right, you see where the file you are writing in is located. At the far right, you will find a small table of contents showing the structure, which also shows the overall context. You can also jump to any file with any element.

Line Numbers

A single line number appears at the right edge of the editor for orientation. In most cases, this is sufficient. If you prefer to display all line numbers, change this in the [settings](#).



Clicking on a line number or pressing  +  allows you to jump to a specific line.

Texts are also saved immediately

Changes in the texts are saved just as quickly as the [project files](#). So you don't have to worry about data loss!

Cursor

While writing, you don't want to constantly take your hand off the keyboard to position the cursor with the mouse. Therefore, the well-known shortcuts for text editing from macOS and iPadOS have been adopted and integrated into adoc Studio.

Table 1. Modifier Keys Used












	Command key
	Option key
	Control key
	Shift key
	Enter key
	Return key
	Delete key
	Forward delete key (only on extended keyboards)

Table 2. Moving in the Document

	Move one character to the left
	Move one character to the right
	Move one line up
















	Move one line down
 + 	Move one word to the left
 + 	Move one word to the right
 + 	Move to the beginning of the line
 + A	Move to the beginning of the line
 + 	Move to the end of the line
 + E	Move to the end of the line
 + 	Move to the beginning of the document
 + 	Move to the end of the document

Table 3. Selecting in the Document




























 + 	Extend selection to the left
 + 	Extend selection to the right
 + 	Extend selection upwards
 + 	Extend selection downwards
 +  + 	Extend selection to the word on the left
 +  + 	Extend selection to the word on the right
 +  + 	Extend selection to the beginning of the document
 +  + 	Extend selection to the end of the document

Table 4. Deleting in the Document

 + K	Delete the current line to the right of the cursor
 + 	Delete the current line to the left of the cursor
 + 	Delete the word to the left of the cursor
 + 	Delete the word to the right of the cursor

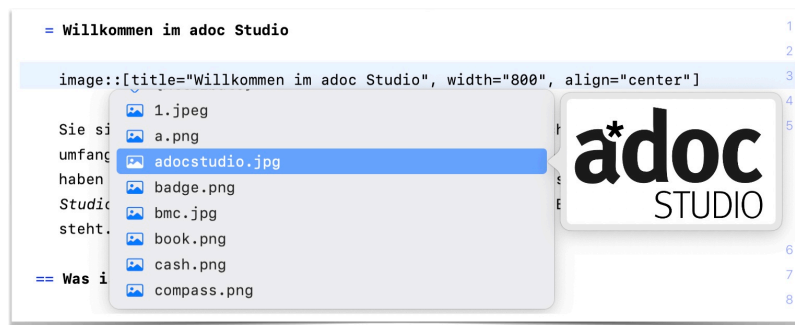
Another keyboard shortcut you might regularly use in adoc Studio is the `ESC` key. This enters you into the large area of input aids.

Input Aids

The functionality of AsciiDoc is very diverse. Even professionals may sometimes forget a command, miss a macro, or not remember a particular syntax. For this reason, adoc Studio has an intelligent input aid integrated.

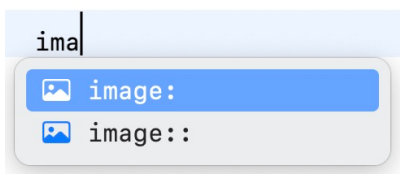
While you are writing, the app continuously searches for ways to assist you. Often, after the second or third character, it recognizes various options and offers these in a completion menu. The goal of adoc Studio is to interrupt your writing flow as little as possible. Therefore, you only receive completion suggestions when you pause your writing.

If needed, you can call up the input aid menu anytime by pressing the `ESC` key.

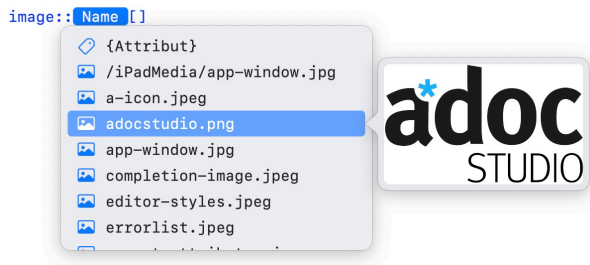


Embedding an Image in the Text

1. Start by typing `ima` at the beginning of a line.
2. A short menu appears:



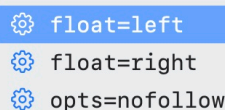
3. Navigate the list with the keys `↓` or `↑`.
4. Select the desired entry with the `↵` key. (here, whether it should be an [image](#) as a block or in the text line).
5. After selection, another menu appears:



The images are conveniently displayed as small previews.

6. After selecting the appropriate image with the `↵` key, the image appears at full width in the preview.
7. Place the cursor between the square brackets after the image name (`[]`) to enter the appropriate options for the image.
8. First, enter the width of the image by typing `wi` . Now the menu with all available options for images containing `wi` appears.
9. Navigate with the arrow keys to `width=` and press the `↵` key.
10. Now you see the following: `image::adocstudio.png[width=Size in pixels]`
11. Overwrite the placeholder text with a number (e.g., `200`) to see the scaling of the image immediately in the preview.
12. Place the cursor again between the square brackets after the image name (`[]`)—it doesn't matter whether before or after the `width` option. It is important that all options are separated by a comma.
13. Press the `ESC` key to bring up the completion menu with the other options for the image.
14. Suppose you want to place the image to the right of a text. You need the `float` option. Start typing `flo` , and the completion menu shows all options containing `flo` :

```
image::adocstudio.png[width=300, flo]
```



15. Select `float=right` .

Congratulations! You have inserted an image that is scaled and positioned to the right of the text. In the source text on the left side, it looks like this: _

```
image::adocstudio.png[width=200, float=right]
```

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor inci-
didunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud ex-
ercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.

```

The result on the right side in the preview looks like this:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.


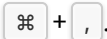


As shown in the example, the completion menu is available for every AsciiDoc command and all attributes. In case of doubt, a simple press of the `ESC` key helps to see all macros possible at the current position.

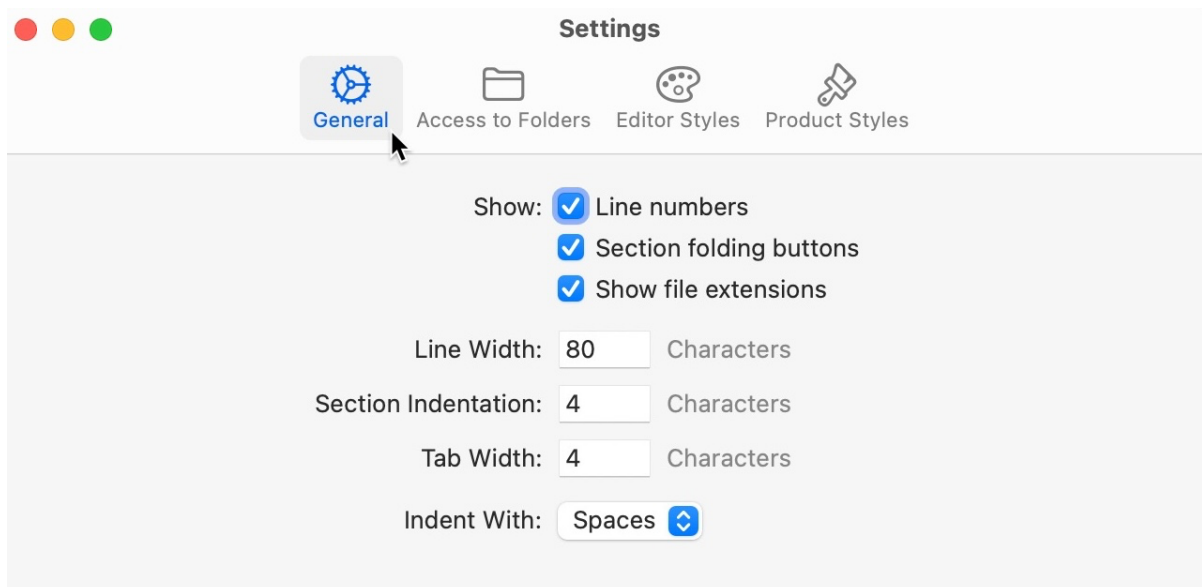
Application Settings

In adoc Studio, you can customize many areas to suit your preferences.

General

Open the settings window from the main menu  or use the keyboard shortcut .

Here you will find the most important settings for adoc Studio under the *General* tab:



Line Numbers:

Decide whether to display all line numbers or only the current line number.

Section Folding Buttons:

Enable the triangles to collapse and expand sections in the text.

Show File Extensions:

Choose whether to display all file extensions in the [project navigator](#) sidebar. If this option is disabled, only the file extension for the currently selected file is displayed.

Line Width:

Define the maximum number of characters per line in the editor before it wraps the text. If the displayed area is too small, the editor will still wrap the line.

Section Indentation:

Choose the amount of space to leave to the left of the equals signs in headings. Set a value between 0 and 6 to adjust the left margin. This aligns the text to the left, while the = signs extend further left.

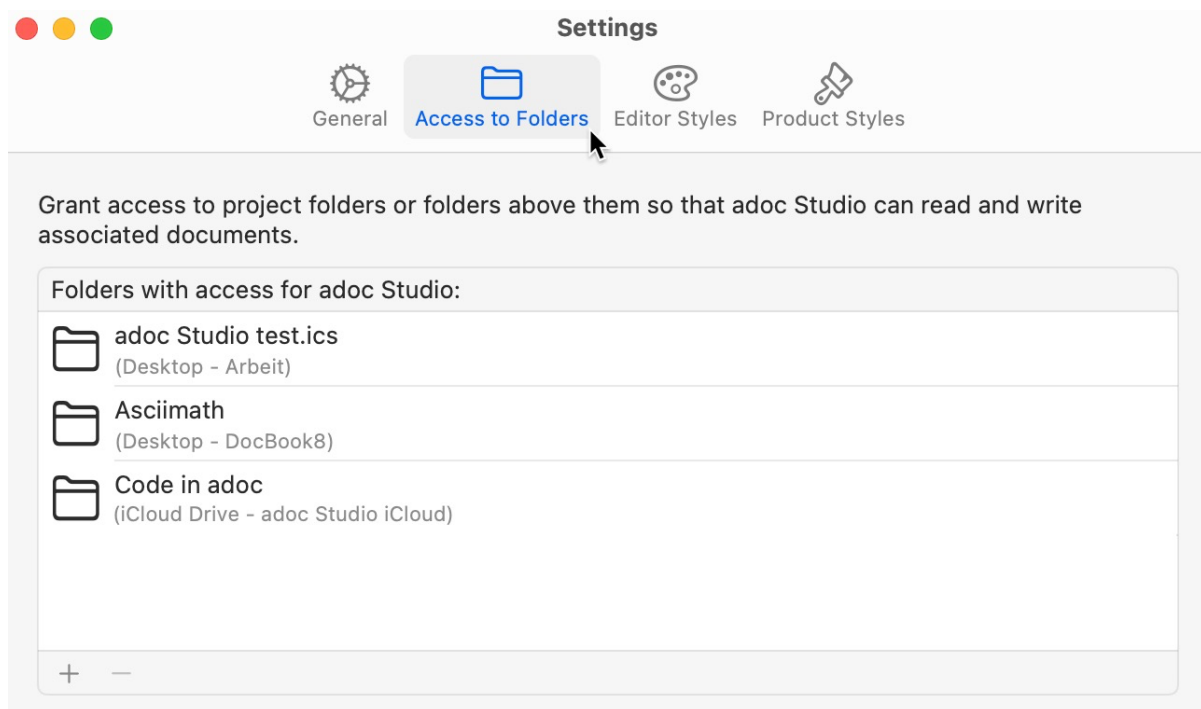
Tab Width:

Specify the width of a tab in the text. You can choose a value between 1 and 100 for the character width of a tab. Values between 3 and 5 have proven to be practical.

Indent With:

Choose whether to use tabs or spaces for indentation.



File Access



Security is a central theme in modern operating systems like *macOS* and *iOS*. To prevent malicious programs from accessing sensitive areas of the operating system, Apple introduced the concept of *sandboxes*.

Each app operates in its own sandbox and is only allowed to function within that environment. This means that while many apps run simultaneously on the computer, each is isolated in its own sandbox. If an app wants to operate outside its sandbox, it requires explicit permission from the user.

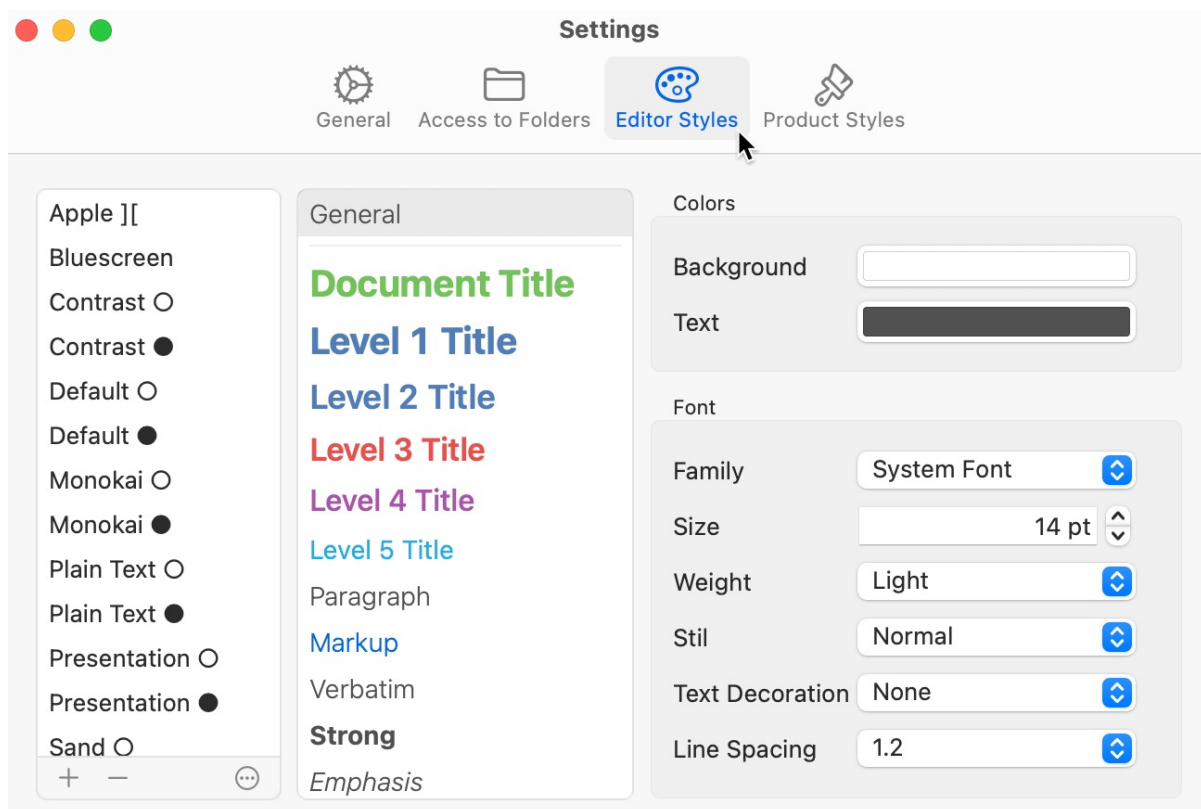
This security principle also applies to adoc Studio on macOS and iOS. Since adoc Studio needs to access all text and media files contained in projects for editing, it requires your explicit permission.

All approved file accesses are listed in this tab in the settings and can be managed using the  and  buttons. When you open a new project not listed here, the app will ask for permission and automatically register it.



To reduce the frequency of permission requests, you can register folders higher up in the directory structure.

Editor Styles




In this section, you can set the appearance of the editor. Browse the list of named styles on the left with your mouse. Most styles are available for both light mode (indicated by a ○ at the end) and dark mode (indicated by a ● at the end). Choose a style you like or that suits your taste.

The middle list displays all elements, attributes, and function names used in the text. A simple rule is to start with the "General" entry and set the font, size, and many other desired attributes there.

If your system is set to automatically switch between light and dark mode, adoc Studio remembers the last chosen style so that the app can follow the system's automatic switch.

Product Styles

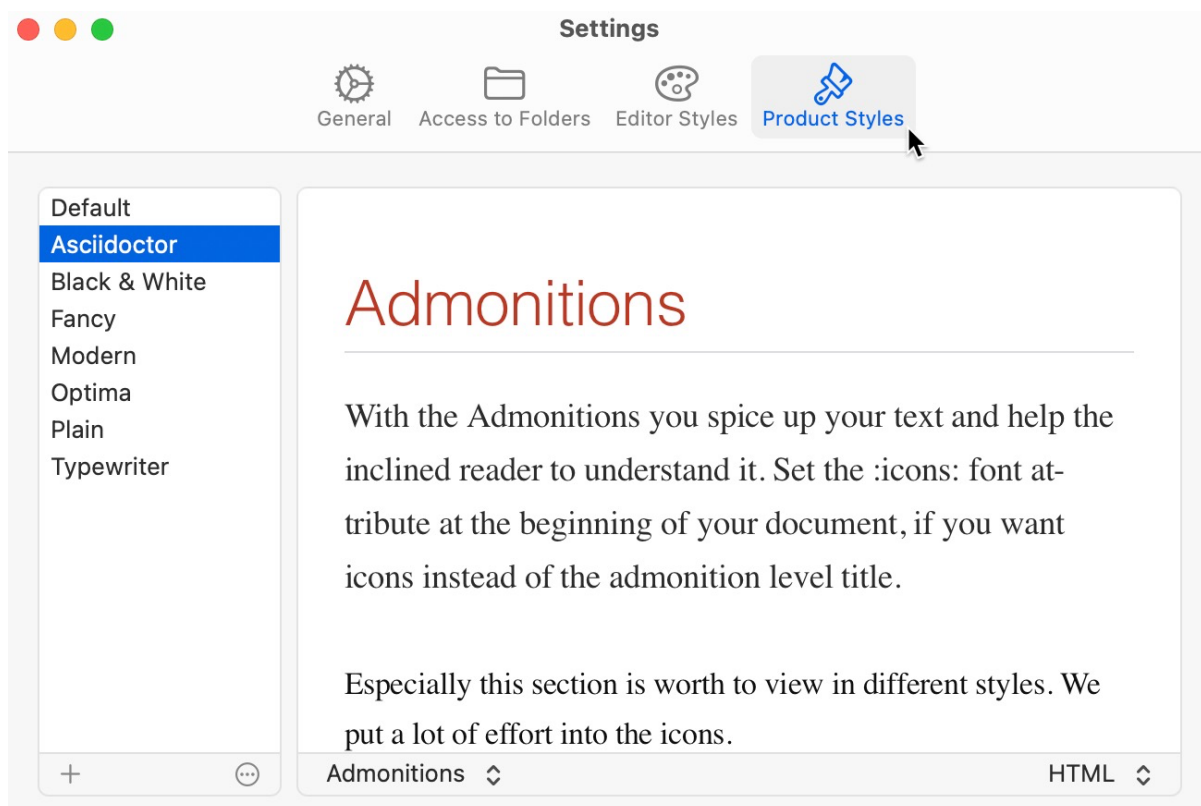
In the   + , under the fourth tab, you will find the product styles.

As previously described, text and design are separated in adoc Studio. Each design comes from a style file, the [product styles](#). Several product styles are already included, but you can also create your own. These product styles are based on [Cascading Style Sheets](#) (CSS), allowing you to design not only texts but also HTML or PDF files.




This means you can format all documents, including PDF files, with a CSS style.

In this tab, you can manage your product styles.



On the left, your own product styles are listed first, followed by the included styles. On the right, examples in the selected product style are displayed. Below the preview, you can choose different examples (left) and the output format (right).

Installing Custom Product Styles


[Product styles](#) are files with the extension `.adocstyle`. New product styles can be installed by double-clicking the file in Finder. Alternatively, you can drag and drop them into the list on the left side of the settings. You will also find a -button below the list of existing product styles to add new ones.

Removing Product Styles

To remove custom product styles, right-click on the desired product style and select *Delete* from the context menu.


Creating Custom Product Styles

To create or modify a custom product style, [CSS knowledge](#) is required. Here's how to easily create a product style based on an existing one:

- Select the desired product style to use as a base.
- Click on  and select *Duplicate* from the context menu. Alternatively, you can open the context menu with a secondary click on the product style name and select *Duplicate*.
- The duplicated style will now appear as a new entry at the top of the list.
- Double-click the name of the new style to edit it.

Editing a Style

To edit the product style, you need a text editor. A suitable tool is [BBEdit](#)[↗]. You can also use any other editor as long as it can save CSS files in UTF-8 format.

- Select the newly created product style.
- Click on the  menu or open the context menu with a secondary click and select *Edit in*.
- Choose your preferred editor from the list.
- The selected app will start, and the product style will open in the app.
- Now you can make all desired changes to the product style.



Ensure that adoc Studio is running in the background and the currently edited style is set for preview. This way, your changes in the external editor will be immediately updated and saved in adoc Studio.

The AsciiDoc Markup Language

adoc Studio uses the AsciiDoc markup language as its foundation. This chapter provides beginners with an initial overview of AsciiDoc's command set and includes further links to the official documentation.



Experienced AsciiDoc users can skip this chapter. However, there's a request: On the [Roadmap for adoc Studio](#)[↗], you'll find a list of the AsciiDoc functions not yet implemented. You can prioritize your personal needs there and discuss with other users in the [forum](#)[↗].others.

What is AsciiDoc?

AsciiDoc is a simplified markup language that makes it easier to create texts in various document formats. AsciiDoc files can be converted into HTML, PDF, ePub, and other formats.

Compared to XML-based document formats like DocBook, AsciiDoc offers the advantage of being *easy* to learn and remaining readable as plain text. More information can be found on [Wikipedia] (<https://en.wikipedia.org/wiki/AsciiDoc>)[↗].

Writing Text

Simply write your text as you are accustomed to. You can write everything continuously, like in other programs, which is often referred to as continuous text. Alternatively, you can write the text line by line, placing each sentence on a new line. The formatted text will always look the same.

Create a new paragraph by inserting a blank line between paragraphs.

The Preamble

The first paragraph of a document is called the *preamble* or sometimes the *abstract*. This paragraph automatically appears slightly larger than the regular text and is often used as an introduction to the document.

To make other paragraphs larger as well, you can prepend a `[.lead]`.

Example 1. Including an Abstract or Preamble in the Middle of the Text

`[.lead]`

This paragraph is displayed slightly larger than all other paragraphs. With the “Optima” product style, the paragraph even gets an initial, a large decorative first letter.

Result:

This paragraph is displayed slightly larger than all other paragraphs. With the “Optima” product style, the paragraph even gets an initial, a large decorative first letter.

Avoid this effect in the first paragraph by prepending a `[.nolead]`. This way, your text starts immediately in the regular font size.





You can find more information about paragraphs on the [AsciiDoc website](#) ↗.

























Formatting

As with any markup language, you can format individual words in **bold**, *italic*, or other styles. This makes transitioning easier. The editor menu in adoc Studio also provides all important formatting options for quick access with the mouse or keyboard.

The editor menu in adoc Studio provides all important formatting options for quick access with the mouse or keyboard.

Table 5. Direct Formatting in the Editor Menu

Command	Keyboard Shortcut	Markup in Text	Result in Preview
Bold	 + 	<code>*bold*</code>	bold

Command	Keyboard Shortcut	Markup in Text	Result in Preview
Italic	 + 	<code>_italic_</code>	<i>italic</i>
Marked	 + 	<code>#marked#</code>	marked
Fixed Width	 +  + 	<code>`fixed width`</code>	fixed width
Underlined	 + 	<code>[.underline]#underlined#</code>	<u>underlined</u>
Strikethrough	 +  + 	<code>[.line-through]#strikethrough#</code>	strikethrough
Overlined	 +  +  + 	<code>[.overline]#overlined#</code>	<u>overlined</u>
Superscript	 +  +  + 	<code>^superscript^</code>	^{superscript}
Subscript	 +  +  + 	<code>~subscript~</code>	_{subscript}

AsciiDoc can do much more. You can freely combine different formatting options.

```
***Super#cali[underline]##fragilist__ic##[red]##expialidocious*** 😊
```

Result in preview:

Supercalifragilisticexpialidocious 😊



You can find more information about text formatting on the [AsciiDoc website](#) ↗.

Headings

You can define headings and their levels by using a number of `=` or `#` symbols. The latter makes it easier to transition from Markdown to AsciiDoc. However, it's advisable to get accustomed to the `=` symbol, as it offers more flexibility.

Here are all the headings in the source text:

```

= Document Title
== Heading 1
=== Heading 2
==== Heading 3
===== Heading 4
===== Heading 5

```

Additional Headings

If the six levels plus the document title are not enough, you can set a paragraph title by preceding a line with a `. :`

```
.Additional Headings
```



You can find more information about heading levels on the [AsciiDoc website](#)[↗].

Table of Contents

When setting headings, consider including a table of contents. To create a table of contents in adoc Studio, simply create the headings (referred to as *Section Title* in AsciiDoc).

If you insert `:toc:` at the beginning of the document, a table of contents will be automatically generated. By default, it includes two levels of headings and is placed above the document. The table of contents is updated automatically.

The `:toc:` attribute can also include parameters:

- **left:** Positions the table of contents to the left of the text.
- **right:** Positions the table of contents to the right of the text.
- **preamble:** Positions the table of contents after the preamble.
- **macro:** Inserts the table of contents at a desired location using the `macro toc::[]`.
- **auto:** Same as without parameters.

Additionally, the `:toclevels: 3` attribute is noteworthy. By specifying a number after the `to-clevels` attribute, you determine how many levels of headings should be displayed in the table of contents. This [attribute](#) must be set at the beginning of the document. By default, two levels are shown.



You can find more information about table of contents on the [AsciiDoc website](#)[↗].

Links

There are two types of links:

- Hyperlinks to the internet
- Cross-references within the document

Hyperlinks to the Internet

The simplest link consists of a written-out address. In the preview, it immediately becomes a clickable URL. Clicking it opens the corresponding page in the browser: <https://adoc-studio.app>[↗]. This works for the main URL schemes:

- http
- https
- ftp
- irc
- mailto

You can also output links as a `macro`: `https://www.adoc-studio.app[]`. You can optionally insert a text between the brackets to be displayed instead of the URL.

`https://www.adoc-studio.app[adoc Studio website]` will be converted to [adoc Studio website](https://www.adoc-studio.app)[↗].



You can find more information about Hyperlinks on the [AsciiDoc website](#)[↗].

Cross-references within the Document

To jump to other positions in the same document (standalone document and collection document), insert a cross-reference.

```
<<Reference>>
```

The reference corresponds to an automatically generated ID of a heading or an anchor.



Use automatic IDs sparingly. If you change a heading that you access via cross-reference, the cross-reference may become invalid.

It is always better to set an anchor before a heading.

```
[#my_anchor]
== My Heading
```

Set the cross-reference in the text with:

```
<<my_anchor>>
```

In the rendered document, the heading is displayed. Optionally, you can also specify a text to appear instead of the heading:

```
<<my_anchor, Jump here>>
```



You can find more information about Cross-references on the [AsciiDoc website](#)[↗].

Footnotes

The macro `footnote:[]` automatically generates footnotes.² The footnote text is inserted in square brackets after the macro:

```
footnote:[Here is the text of the footnote]
```



AsciiDoc currently only supports footnotes displayed as endnotes. Therefore, both `footnote` and `endnote` are implemented in adoc Studio. To ensure compatibility with existing documents, the attribute `:footnotes-position:` can be used.



You can find more information about Footnotes on the [AsciiDoc website](#)[↗].

Images

In the [introduction](#), we briefly discussed images and their paths. Now, we will delve into how to incorporate images into the AsciiDoc text.

2. Here is the footnote text

You can insert images in two ways: as block images or as inline images.

Block Images

The basic syntax is:

```
image : :image.png[]
```

The text should be on its own line. The brackets can contain many options, with adoc Studio's auto-complete menu being helpful, as shown in the example "[Embedding an Image in the Text](#)". Three important options are:

- **width**: The width of the image.
- **align**: The alignment of an image.
- **float**: The float property of the image.

Inline Images


The syntax for inline images is very similar to that of block images, with the difference that one colon `:` is omitted:

```
image: image.png[]
```

An image embedded in the flow of the text is inserted as follows:

```
Please press image:reload.svg[] to reload the page.
```

Results in:

Please press  to reload the page.

Placing images side by side can also be easily achieved with inline images:

hello hello hello



You can find more information about Images on the [AsciiDoc website](#) ↗.

Lists

To create an unordered list, use the following syntax:

- A list starts with a `*`.
- use `**` to go one level deeper.
 - And with each additional asterisk,
 - it goes one level deeper.
 - or
- up again.

If you are familiar with Markdown, you can also use the following syntax for list items:

- Markdown lists start with a `-`.
- this works
- in adoc Studio too

Ordered lists are just as easy to create:

1. one
2. two
3. three

You also have the option to specify numbers:

1. one
2. two
3. three
4. Incorrect numbering will be ignored. In this case, it won't show 7 as in the editor, but rather 4.



In adoc Studio, the numbering is always created automatically.

When lists need to be a bit more complex, follow the same concept as with the unordered list:

1. Step 1
 - a. Step 1a
 - b. Step 1b
2. Step 2

3. Step 3

Checklists are also possible:

- to do
- done
- also done

and can be mixed with other entries.



You can find more information about Lists on the [AsciiDoc website](#)[↗].

Block Elements

Block elements, often simply referred to as blocks, are extremely versatile in AsciiDoc. They can be used in a variety of ways and forms. Some common uses include:

- [Admonitions](#)
- Marginal notes
- Source code and listings
- Equations
- Pass-throughs
- Quotes

It's important to know that blocks—unless they're like, for example, the [Admonitions](#) that are represented on a single line—always have two delimiters: the block start and end. There are many different characters for this purpose, as seen in the following quote example:

"Learning is experience. Everything else is just information."

– Albert Einstein



You can find more information about Text and other blocks on the [AsciiDoc website](#)[↗].

Admonitions

In particular, this area is worth examining in different [styles](#). We put a lot of effort into the little icons.

Make sure that at the beginning of your document, the [attribute](#) `:icons: font` is set. Otherwise, instead of symbols, only text will appear.



NOTE provides additional information to the reader.



TIP suggests a trick or special tip to the reader.



WARNING alerts the reader to a danger.



CAUTION warns the reader of a pitfall.



IMPORTANT draws the reader's attention to an important point.

And here's a practical tip:



You can find more information about Admonitions on the [AsciiDoc website](#)[↗].

Tables

Tables in AsciiDoc are quite versatile, similar to block elements. You can start with simple tabular representations like this:

One	Two	Three
-----	-----	-------

Add a header row by inserting a blank line after the first row:

Title 1	Title 2	Title 3
One	Two	Three
Four	Five	Six

Here are some complex examples, as found on the [AsciiDoc website](#)[↗]:

Column 1	Column 2
<p>This content spans two columns (2*) and is aligned to the right edge of the cell (>).</p> <p>The text is rendered in a monospace font (m).</p>	
<p>This cell spans 3 rows (3+). The content is centered horizontally (^), aligned to the bottom edge of the cell (.>), and rendered in strong formatting (s).</p>	<p><i>This content is rendered in italics (e).</i></p>
	<p>This content is rendered in a monospace font (m).</p>
	<p>This content is bold (s), except for the word <i>content</i>.</p>



You can find more information about Tables on the [AsciiDoc website](#) ↗.

User Interface

In technical documentation, descriptions of the user interface (GUI) or simply the UI (User Interface) are commonplace. There are three particularly important areas:

Keyboard Shortcuts

With the `kbd:[]` macro, you can create keyboard shortcuts in your manual. For example, `⌘ + S` quickly saves the document.

A special feature of the `kbd:[]` macros is shown in the use of the [arrow keys](#) in this document.

Menus

The `menu:[]` macro helps in describing menu entries. For example, you start the product export via

 File > Export > Products ...

Buttons

The `btn:[]` macro is responsible for displaying all types of buttons. For example, you have a `OK` button available.



You can find more information about UI Macros on the [AsciiDoc website](#) ↗.

Comments

Especially when viewing this document as source code in adoc Studio, it's worth looking at the editor. After all, comments are only displayed here.

At the beginning of a line, you can mark the following text as a comment using `//`.

```
// This text is a comment and will not be displayed in the document.
```

You can define entire sections as comments by using `////` at the beginning and end of the block:

Why should you insert comments that do not appear in the final document? Here are some reasons:

- `// TODO: :` Pending tasks such as optimizations, improvements, and corrections.
- Formatting instructions for the text.
- Justifications and explanations for specific content.
- Internal version notes about changes and revisions.
- Notes for editors.
- Additional contextual information that is helpful for understanding the source text in the editor but not needed in the final document.



You can find more information about Comments on the [AsciiDoc website](#) ↗.

Attributes

Using attributes can elevate your documentation to a whole new level.

The term "attribute" is used in adoc Studio in various ways. Essentially, they allow you to:

- Define values for later use in the text
- Customize settings in the document

Using Attributes as Variables

In many cases, certain words, URLs, and other information repeat throughout the text. Some terms change depending on the version of the document, while others, such as URLs, are so long that it's advisable to enter them only once to avoid errors.

To reuse attributes anywhere in the document, define them as follows:

*Example 2. Using an attribute as a variable**Definition*

```
:app-version: 1.2.3.4
```

Input

```
This document is based on version {app-version}.
```

Output

```
This document is based on version 1.2.3.4.
```

This already opens up extensive possibilities. Every time the name adoc Studio appears in this documentation, the attribute `{app-name}` is used. Here are some other attributes used in this document:

```
:app-name: adoc Studio
:lang: en
:url-website: https://www.adoc-studio.app
:url-roadmap: https://adoc-studio.canny.io
:url-styles: https://styles.adoc-studio.app
:url-forum: https://forum.adoc-studio.app
```

In adoc Studio, there are a number of standard attributes that are already prepared, as they are needed in (almost) every document. However, these must be entered at the beginning of your document in the [document header](#):

- Author information (`:author:`)
- Version attributes (`:revnumber:` , `:revdate:` , and `:revmark:`)
- Metadata (`:description:` , `:keywords:`)

An example for this document might look like this:

```
:author: Frank Blome
:revnumber: 1.0
:revdate: {docdate}
:description: This document introduces you to working with {app-name}.
:keyword: {app-name}, AsciiDoc, technical documentation, manuals, and more
```



You can abbreviate a standard title as follows:

```
= The Manual for {app-name}
Frank Blome <frank@projectwizards.net>
1.0, {docdate}
```

In this case, the information for `:author:`, `:email:`, `:revnumber:`, and `:revdate:` is automatically retrieved from the context after the document title (`=`) and can later be used as attributes with `{ }`.

Attributes for Document Settings

Another purpose of attributes is to control the settings of your document. Each document contains various name-value pairs referred to as document attributes.

These attributes allow you to configure the backend processor and specify document metadata. You can use pre-existing built-in attributes or define your own.



Always set document attributes at the beginning of a line!

Built-in Attributes

Built-in attributes actively support the configuration and control of the document. Their effect often unfolds only when they are placed in a line at the beginning of the document.

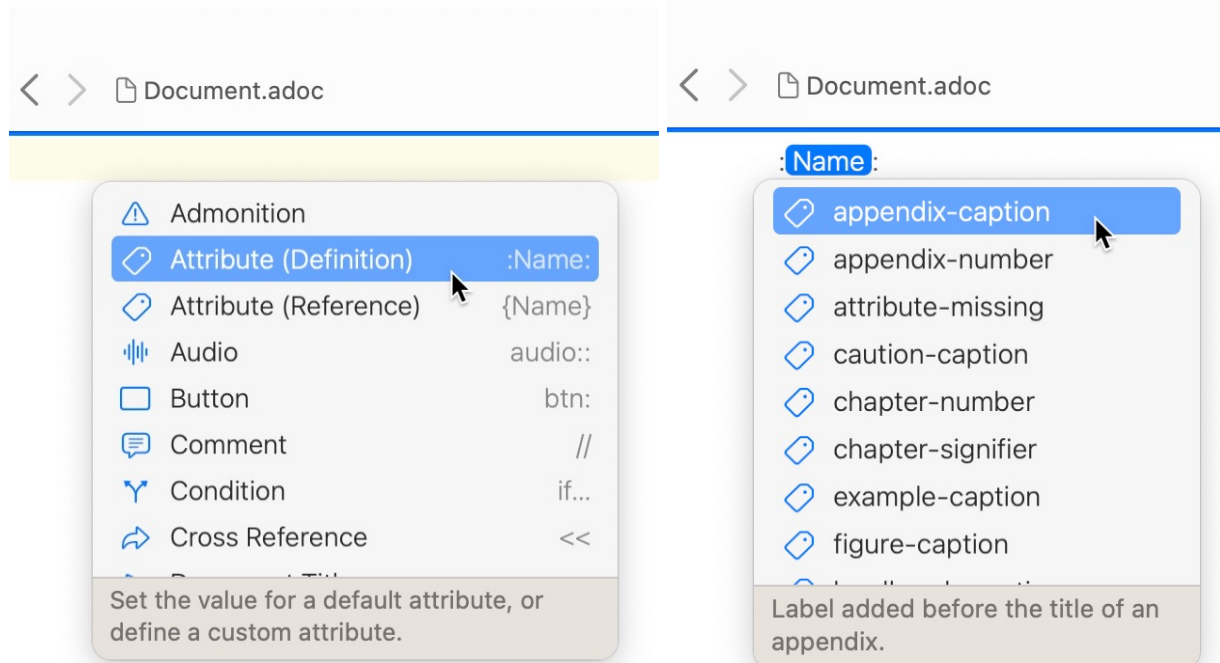
Attributes can:

- Provide access to document information.
- Define document metadata.
- Enable or disable built-in functions
- Configure built-in functions
- Specify the location of assets, such as images.
- Store content for reuse in a document.

Examples

- You set attributes either by just the plain entry: `:beta:` .
- You set attributes with additions: `:beta-version: 23` .
- You set attributes with an exclamation mark to disable or negate them: `:!beta:` or `:beta!` .

In adoc Studio, accessing all built-in attributes is extremely user-friendly. Simply press the `ESC` key at the beginning of a line and select **Attributes** from the text completion menu. You will then find all available built-in attributes listed.



You can find more information about built-in attributes on the [AsciiDoc website](#).

Custom Attributes

Custom attributes are those defined by the author and are not reserved by the AsciiDoc language or an extension. They are usually used to replace text. With these attributes, you can define named and reusable content.

Instead of repeating text throughout the document, such as a product name, this text can be set in an attribute and referenced by its name in the document. This method helps you make the document more efficient and apply the "Don't Repeat Yourself" (DRY) principle.

An example used regularly in the documentation for  [Merlin Project](#):

```
:mp-mpe-features: https://www.projectwizards.net/merlin-project/features
:not-in-mpe: CAUTION: This feature is not included in Merlin Project Express. +
    Please compare your requirements with the {mp-mpe-features}[feature
    comparison] table.
```

Becomes:



This feature is not included in Merlin Project Express. Please compare your requirements with the [feature comparison](#)[↗] table.

This method saves a lot of work because you don't have to copy and paste the text repeatedly. If you need to change the text, a single adjustment at one place in the document is enough.



You can find more information about document attributes on the [AsciiDoc website](#)[↗].

Conditional Statements

Conditional statements are control structures in programming where a section of code is executed only under certain conditions. Similarly, this can be applied to texts in adoc Studio. Here are the conditional statements you can use:

- **ifdef:** The content is included only if the specified attribute exists.
- **ifndef:** The content is included only if the specified attribute does *not* exist.
- **ifeval:** The content is included only if the expression within the square brackets of the ifeval directive evaluates to true.

When the processor encounters one of these conditions, it evaluates the specified condition. This is based on the presence or value of one or more document attributes. If the condition evaluates to true, the enclosed lines are included; otherwise, they are skipped.



You can find more information about conditional statements on the [AsciiDoc website](#)[↗].

Includes

One of the core features of AsciiDoc is managing split files instead of creating a single endlessly long file. You divide the text into many individual files and then assemble them according to certain rules.

In adoc Studio, a specific concept was already introduced in the section [The Project Navigator](#).

For instance, if you are collaborating with authors who cannot use adoc Studio, using the `include` command is advisable. This way, chapters available in the AsciiDoc format can be included sequentially.

```
Die Datei „chapter-1.adoc“ konnte nicht geöffnet werden, da sie nicht existiert.
Die Datei „chapter-2.adoc“ konnte nicht geöffnet werden, da sie nicht existiert.
Die Datei „chapter-3.adoc“ konnte nicht geöffnet werden, da sie nicht existiert.
```

Includes in adoc Studio

Within adoc Studio, using the `include` command can be beneficial. Suppose you face the challenge that not all features of your product can be documented at once. In such cases, you need a placeholder.

For this project, a placeholder named `comingSoon.adoc` is defined in the file list. To ensure this file is not included in the compiled document, the corresponding checkbox is unchecked in the file information. More details can be found [here](#).

Whenever a feature is not yet described, place the following text:

Die Datei „comingSoon.adoc“ konnte nicht geöffnet werden, da sie nicht existiert.

This will be output as follows:



The *important features* are not yet described. A detailed documentation will be available as soon as possible.

The trick lies in the content of the `comingSoon.adoc` file:

```
ifdef::feature[]
```

```
CAUTION: The _{feature}_ are not yet described. A detailed documentation will be available as soon as possible.
```

```
:!feature:
```

```
endif::[]
```

1. When you include the file using the `include` command, a text is first assigned to the `feature` attribute. In this case, "important features".
2. Then, the `comingSoon.adoc` file is included at the current position.
3. Next, it checks whether the attribute contains a text.
4. If it does, a warning is issued.
5. After that, the attribute is cleared.









You can find more information about Includes on the [AsciiDoc website](#) ↗.

Exporting Documents

When exporting documents, adoc Studio offers a new method that is particularly useful for large projects. You can choose between the regular export of a document and the output of products. But first, let's look at the preview.

The Preview

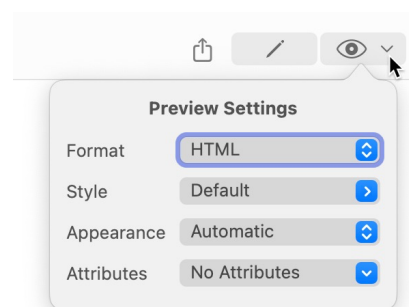
To the right of the editor, you will find the preview. If it is not displayed, click on the  icon or use the keyboard shortcut  +  +  to show it.

The preview adapts to the set output format. Sometimes it appears like a browser displaying an HTML page, sometimes like a PDF preview, or an RTF display. You select the output format in the dialog behind the icon  next to the eye icon . There you can also make further settings that vary depending on the output format.

HTML Preview and Settings

When you set the output format to HTML, the document in the editor is displayed as a long HTML document.

- **Style:** Choose the desired product style for your document's preview.
- **Appearance:** Decide whether the display should be light or dark. This is often referred to as *Darkmode* and *Brightmode*. In automatic mode, adoc Studio follows your operating system's control panel settings.
- **Attributes:** Manage the document attributes. In addition to the attributes defined in the document, you can set additional attributes through a dialog. More details can be found in the description of the products.




Text Preview and Settings


In addition to the options for style, appearance, and attributes, you can also choose:

- **Text format:** Choose whether the text should be formatted (as RTF) or plain text. When output as plain text, styles are not applied and images are not displayed.

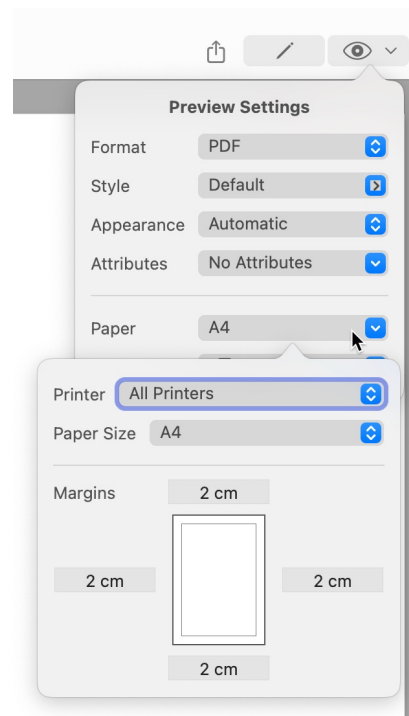
PDF Preview and Settings

The PDF preview shows the document as it will be written to a file upon export. The settings in the dropdown menu behind the  icon include not only the style, appearance, and attributes but also:

- **Paper:** You can set the printer, paper size, and margins. More details about these settings can be found in the chapter on styles.
- **Orientation:** Set whether the document should be displayed in portrait mode or landscape mode.

In the menu  **Preview**, you will find entries to optimize the display for your screen size when the PDF format is active.


- Single page
- Double page
- Single page scrolling
- Double page scrolling



Features for Editor and Preview

Linking Preview with the Editor

By default, the preview scrolls to the same line as the editor, so the corresponding text in the preview is always at the same height as the current line in the editor.

However, if you want to copy text from another preview and paste it into the editor, you can disable this function in the menu  **Preview > Link with Editor > Reading Position**.

Linking Fold State with Editor

If the option *Buttons to Collapse Sections* is activated in the settings, you can disable this linkage separately here.

The fold state affects the sections, i.e., the headings from 2x `==` to 6x `=====`, but not the document title.

When you collapse a section in the editor, it is also collapsed in the preview. Even if the editor is hidden, the collapse triangles are shown in the preview, allowing you to continue to expand or collapse the text.


Linking Content with Editor

Through the menu `Preview > Link with Editor > Content`, you can set whether a file is automatically displayed in both the editor and the preview. When this option is activated, everything you select in the sidebar is shown in both the editor and the preview.



If you hold the `⌘` key and click on a file in the sidebar, the selected file is unlinked from the editor and displayed only in the preview. This way, you can quickly view a file in the preview while editing another in the editor.

Exporting

To export a file, you can either use the menu `File > Export > Selection` or the keyboard shortcut `⌘ + ⌘ + E`. Alternatively, you can use the  icon in the toolbar on the right.

Export Selection

Product Like Preview

Format HTML

Style Default


Appearance Automatic

Attributes No Attributes

Resources Separate

Action Export

Cancel Export...

If you are satisfied with the display in the preview, simply check the *Like Preview* box and press .

Otherwise, the dialog offers the same output options as explained in the [Preview](#) section:

- **Format:** Choose the desired target format for the file (Text, HTML, or PDF).
- **Style:** Define the appearance of the file. More details can be found [here](#).
- **Appearance:** Decide whether the document should be displayed in light or dark mode, or let the system decide.
- **Attributes:** Set predefined or custom attributes for the output. An introduction can be found [here](#).
- **Specific output settings:** Adjust additional settings according to the output format.
- **Action:** Choose what should be done with the exported file. The default is "Export," which you can execute by clicking "Export." Alternatively, you can send the file directly by email, share it via Apple AirDrop, or share it with others in different ways.

When exporting in HTML format, you can also set the output of the HTML file's resources. HTML file resources include media and style files. Here are some possible options:

- **Separate:** Resources are saved as separate files in a folder next to the HTML file.
- **Inline:** Resources are embedded in the HTML file.
- **None:** Resources are not exported.

- **From Attributes:** Resources are not exported, but the attributes `:stylesheet:` and `:stylesdir:` are respected. If these attributes are not set, the resources will be exported.

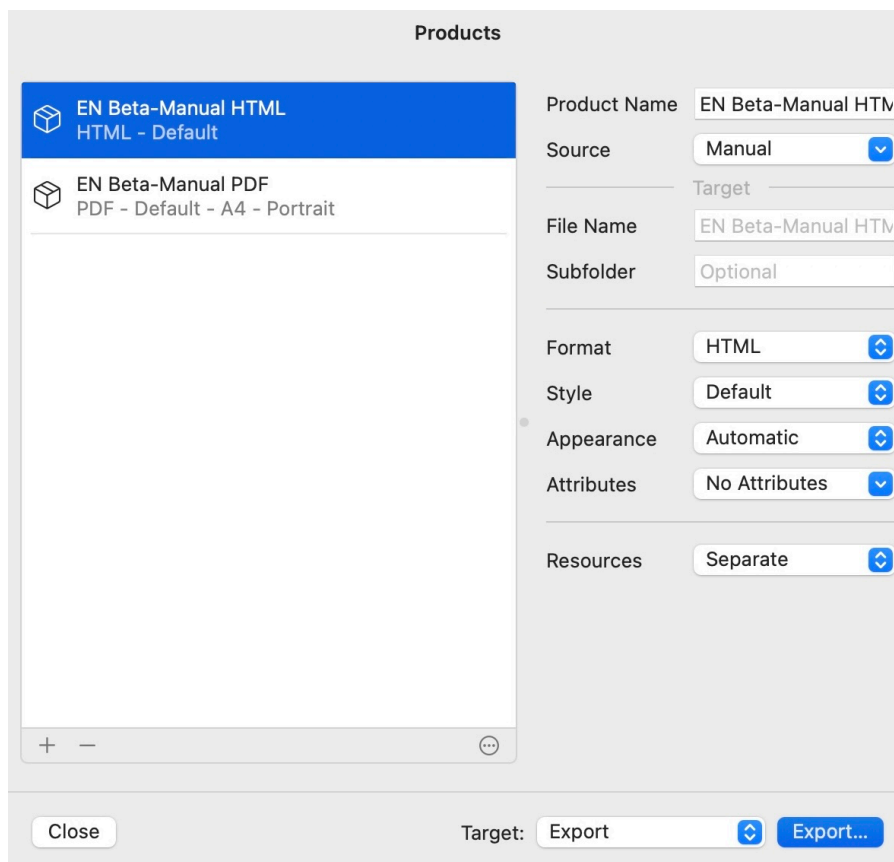
In the settings, you will find a list where you can set [specific attributes](#) for a document. Open the list to see a small dialog where you can create new attributes or select existing ones.



All entries in this list override the attributes entered in the text and therefore have the highest priority.

Products

For documents created with adoc Studio, long-term maintenance and output are common. Therefore, a tool is needed that efficiently manages complex export configurations as well as frequent switches between different settings and special attributes, making them quickly accessible. This task is handled by products.



You can open the dialog box for configuring products via the menu **File > Export > Products ...** or with the keyboard shortcut `⌘ + ^ + E`.

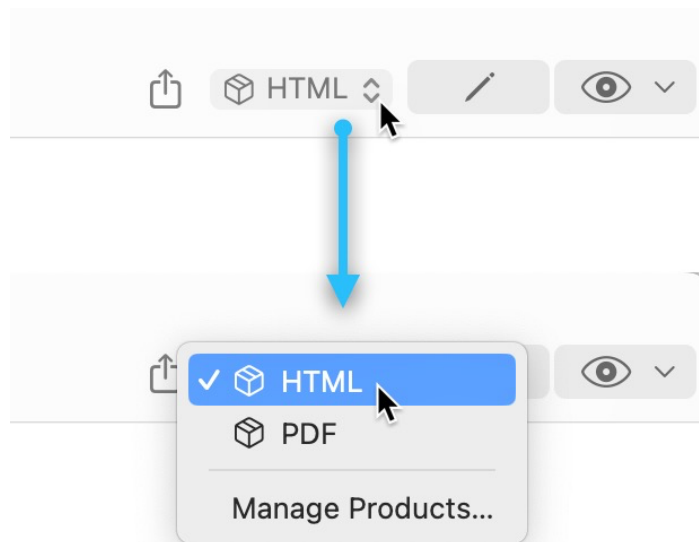
If you access the products for the first time, the list on the left side will still be empty. By clicking on the `+` symbol, you can access a list of all documents. Select the desired document or master document here.


On the right side, you will now find various options:

- **Product Name:** Here you can optionally specify a name for your product.
- **Source:** Here you select the source for your product. You can change this selection at any time.
- **Filename:** Enter the name for the exported file here. If this field remains empty, either the product name or, if not available, the file name of the source with the corresponding file extension will be used.
- **Subfolder:** You can optionally specify a subdirectory where the product should be exported.

You can now set the same export parameters as in the previous section, [Exporting](#).

Products in the Toolbar









Once you have created a first product, the product name appears in the toolbar to the left of the . All further products will be added to this list.



Set up a product for output in HTML format and another for PDF format. This way, you can easily switch between the quick HTML preview and the somewhat slower PDF preview in the toolbar.

Exporting Products

To export a product, select the desired target in the products dialog and then click either  or , depending on what you want to do.

Alternatively, you can also export multiple products at the same time. To do this, go to the context menu in the products dialog or click on  and select the  entry. This will display checkboxes next to each product. Check the products you want to export and then click  or .

If you perform multiple HTML exports, the products share the resources. This means that after the export, you will have a separate .html file for each product, but only one folder with shared images, CSS files, etc.

There's Still Something Missing...

The development of adoc Studio is still in its early stages. For the first version, the focus was placed on functions that 80% of users need. However, it's entirely possible that you might find that a specific function is missing.

Even if you are already familiar with AsciiDoc or adoc Studio and find that something is missing, you can actively support the development. Register for free on the *Roadmap Platform* and vote for the missing features. Features with the most votes are prioritized for implementation.

You also have the opportunity to express your own wishes or suggest new features. Your ideas might even gain support from other users. This ensures that the features you need most will be the next to appear in adoc Studio.

Look forward to the exciting developments that are yet to come!

Appendix A: Support

You've made it! Up to this point, you have focused on the description of concepts and features. Now it's your turn. Start writing and share your successes!

System Requirements

The minimum system requirements for using adoc Studio are:

- macOS 13 Ventura
- iOS 16

First Aid

Forum:

A [free forum](#)[↗] is available where the team from [ProjectWizards](#)[↗] is also on hand to offer advice and assistance.

adoc Studio Roadmap:

To not only see but also influence future development, you have access to [a public roadmap](#)[↗]. If you're missing a feature, please add it here, or vote for an existing request that you also find important.

The AsciiDoc Command Reference:

You can find the always up-to-date [AsciiDoc Command Reference](#)[↗] here.

Personal Experiences with AsciiDoc:

On the [ProjectWizards website blog](#)[↗], you'll find a variety of practical articles on AsciiDoc.

Additional Tips on AsciiDoc:

Visit [Awesome AsciiDoc\(tor\)](#)[↗] for an extensive collection of various tips and tricks for AsciiDoc.

Help with Templates and Styles

A partner network is currently being established to assist you with building your templates and designing product styles. You can find the current list of partners [online](#) [↗].

Over time, a directory of all styles will also be created there. If you're interested in actively participating, send a message to: office@projectwizards.net.

Appendix B: Attributes

We explained the concept and usage in the [Attributes](#) chapter. Below is a list of all the attributes used in adoc Studio.

adoc Studio Attributes

ads-compat-mode:

When set, compatibility with AsciiDoctor is maximized. For example, dates and times are not localized, author names are split differently.

ads-platform:

Contains *macOS* or *iOS* depending on the platform you are using adoc Studio.

ads-device:

Contains *Mac*, *iPhone* or *iPad* depending on the device you are using adoc Studio.

Intrinsic Attributes

backend:

Always set to *html5* for compatibility with AsciiDoctor.

backend-html5:

Always set for compatibility with AsciiDoctor.

basebackend:

Always set to *html* for compatibility with AsciiDoctor.

basebackend-html:

Always set for compatibility with AsciiDoctor.

docdate:

Last modification date of the source document without time. Localized according to the language and locale provided via the *lang* attribute.

docdatetime:

Last modification date and time of the source document. Localized according to the language and locale provided via the *lang* attribute.

doctime:

Last modification time of the source document. Localized according to the language and locale provided via the *lang* attribute.

docyear:

Year that the source document was last modified. Localized according to the language and locale provided via the *lang* attribute.

docdir:

Path to the directory that contains the source document. The path starts with a slash and the root directory of the project.

docfile:

Path to the source document. The path starts with a slash and the root directory of the project.

docfilesuffix:

File extension of the source document including the leading period.

docname:

Name of the source document without file extension.

embedded:

Can be set as an option for an HTML export. The generated HTML then does not contain *head* or *body* tags and can therefore be embedded directly in another HTML page.

filetype:

File extension of the main output file of the document, without leading period. The extension depends on the selected output format.

filetype-html:

A convenience attribute for checking whether the output *filetype* is *html*.

filetype-pdf:

A convenience attribute for checking whether the output *filetype* is *pdf*.

filetype-rtfd:

A convenience attribute for checking whether the output *filetype* is *rtfd* (rich text).

filetype-txt:

A convenience attribute for checking whether the output *filetype* is *txt* (plain text).

htmlsyntax:

Currently always *html* as adoc Studio does not support xhtml output yet.

outfilesuffix:

File extension of the actual output file of the document, with leading dot. The value depends on the selected output format.

localdate:

Date when the document was converted to the output format.

localdatetime:

Date and time when the document was converted to the output format.

localtime:

Time when the document was converted to the output format.

localyear:

Year when the document was converted to the output format.

Compliance Attributes

attribute-missing:

Controls how missing attribute references are handled.

attribute-missing → skip:

Leave the unresolved reference in place and create an error.

attribute-missing → drop:

Drop the reference.

Localization and numbering attributes

lang:

Language tag like *en* or *en-US*. Is added to the root element of the HTML output. Controls hyphenation and localization of dates.

nolang:

Prevents *lang* attribute from being added to root element of the HTML output.

appendix-caption:

Label added before the title of an appendix.

appendix-number:

Sets the seed value for the appendix number sequence.

caution-caption:

Text used to label *caution* admonitions when icons aren't enabled.

chapter-number:

Sets the seed value for the chapter (level 1 section titles) number sequence when using doctype *book*.

chapter-signifier:

Label added to level 1 section titles (chapters) when using doctype *book*.

example-caption:

Text used to label example blocks.

example-number:

Sets the start value for the example number sequence. Is only used if the *example-caption* attribute is set and the example has a title.

figure-caption:

Text used to label images and figures.

figure-number:

Sets the seed value for the figure number sequence. Is only used if the *figure-caption* attribute is set and the figure has a title.

important-caption:

Text used to label *important* admonitions when icons aren't enabled.

last-update-label:

Text displayed in the footer before the last modification date.

listing-caption:

Text used to label listing blocks.

listing-number:

Sets the seed value for the listing number sequence. Is only used if the *listing-caption* attribute is set and the listing has a title.

note-caption:

Text used to label *note* admonitions when icons aren't enabled.

table-caption:

Text of label prefixed to table titles.

table-number:

Sets the seed value for the table number sequence. Is only used if the *table-caption* attribute is set and the table has a title.

tip-caption:

Text used to label *tip* admonitions when icons aren't enabled.

toc-title:

Title for table of contents.

untitled-label:

Default document title if document doesn't have a document title.

version-label:

The label displayed before the revision number in the document title byline.

warning-caption:

Text used to label *warning* admonitions when icons aren't enabled.

Styling and Layout

hyphens:

Controls whether and how text is hyphenated in HTML and PDF.

hyphens → none:

Words are not broken at line breaks, even if characters inside the words suggest line break points. Lines will only wrap at whitespace.

hyphens → manual:

Words are broken for line-wrapping only where characters inside the word suggest line break opportunities.

hyphens → auto:

Words are broken automatically at appropriate hyphenation points according to the chosen language in the *lang* attribute. However, suggested line break opportunities will override automatic break point selection when present.

title-page:

Puts a dedicated title page at the start of a PDF document. The title page contains the *doctitle*, *author*, *date*, and *revision* info.

pagenums:

Enables showing of page numbers in PDF documents.

outline:

By default, adoc Studio generates a PDF outline for PDF documents. You can turn it off by unsetting this attribute.

outlinelevels:

Adjust the depth of section levels that are displayed in the PDF outline independent of the TOC level depth. By default, it is set to the same value as *toclevels*.

outline-title:

By default, the document title is displayed as the title of the PDF outline. You can customize this using this attribute.

page-background-image:

Set this attribute to add a background image to all PDF content pages. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image scale and position.

page-background-image-recto:

Set this attribute to add a background image separately for all recto PDF content pages. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image size and position.

page-background-image-verso:

Set this attribute to add a background image separately for all verso PDF content pages. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image size and position.

title-page-background-image:

Places a logo image in the content area of the title page. The title page needs to be enabled via the *title-page* attribute. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image size and position.

front-cover-image:

Creates a separate cover PDF page at the very beginning of the document containing only an image. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image size and position.

back-cover-image:

Creates a separate cover PDF page at the very end of the document containing only an image. For the value you can either specify a simple path to an image or an inline image macro. The latter allows to specify further attributes to control the image size and position.

pdf-page-size:

Setting this attribute overrides the PDF page size defined in the preview or product options.

pdf-page-layout:

Setting this attribute overrides the portrait or landscape PDF page orientation defined in the preview or product options.

pdf-page-margin:

Setting this attribute overrides the PDF page margins defined in the preview or product options.

pdf-page-margin-rotated:

Define separate PDF page margins for landscape orientation.

text-align:

Overrides the default text alignment of the chosen style.

Document metadata attributes

app-name:

Value for *application-name* meta element for mobile devices inside HTML document head.

author:

Contains the main author's name. Can be set automatically via the author info line or explicitly.

authors:

Contains a list of author names. Can be set automatically via the author info line or explicitly as a comma-separated value list.

authorinitials:

Derived from the author attribute by default. Can also be set explicitly.

email:

Extracted from author info line by default. Can be any inline macro, such as a URL. Can also set set explicitly.

firstname:

Derived from the author attribute by default. Can also be set explicitly.

middlename:

Derived from the author attribute by default. Can also be set explicitly.

lastname:

Derived from the author attribute by default. Can also be set explicitly.

copyright:

Value for *copyright* meta element inside HTML document head.

doctitle:

Set to the level 0 section title by default. Can be set explicitly.

description:

Value for *description* meta element inside HTML document head.

keywords:

Value for *keywords* meta element inside HTML document head.

publisher:

Is put into the *producer* metadata field of PDF documents.

subject:

Is put into the *subject* metadata field of PDF documents.

title:

Value of *title* element in HTML head. Used as a fallback when the document title is not specified via either a level 0 section or the *doctitle* attribute.

revnumber:

Extracted from revision info line.

revdate:

Extracted from revision info line.

revremark:

Extracted from revision info line.

Section title and table of contents attributes

idprefix:

By default, auto-generated section IDs begin with an underscore. You can change it by setting this attribute. The value must begin with a valid ID start character and can have any number of additional valid ID characters. If you want to remove the prefix, set the attribute to an empty value.

idseparator:

The default section ID word separator is an underscore. You can change it with this attribute. Unless empty, the value must be exactly one valid ID character.

leveloffset:

Changes the level depth of all headings by the specified number of steps. The value is relative and begins with + or -. This allows you to publish each chapter as a standalone document, complete with a document title.

partnums:

To autogenerate book part numbers, set this attribute in the book header.

sectanchors:

When enabled, an anchor (empty link) is added before the section titles. The default stylesheet renders it as a § symbol that floats to the left of the section title.

sectids:

Section and discrete headings without an explicitly set identifier receive an auto-generated ID. You can disable this by unsetting this attribute. Elements without an ID cannot be cross-referenced.

sectlinks:

When set, section titles are turned into links. The default stylesheet shows them with the same style as unlinked section titles. Linked section titles allow for easier bookmarking in a Webbrowser.

sectnums:

Sections aren't numbered by default. However, you can enable this feature by setting this attribute. You can toggle numbering on and off throughout a document.

sectnums → all:

Sections that are assigned a built-in special style aren't numbered by default. To number regular sections as well as special sections, set `sectnums` and assign it a value of *all*.

sectnumlevels:

When *sectnums* is set, level 1-3 section titles are numbered by default. You can adjust the level limit to value of 0 through 5.

toc:

Turns on table of contents and specifies its location. By default, it is inserted directly below the document title, author, and revision lines.

toc → auto:

The table of contents is inserted directly below the document title, author, and revision lines.

toc → left:

The table of contents is positioned to the left of the main content column. It is both fixed and scrollable.

toc → right:

The table of contents is positioned to the right of the main content column. It is both fixed and scrollable.

toc → preamble:

The table of contents is positioned immediately below the preamble, the content between the end of the document header and the first section title.

toc → macro:

Place the table of contents in a specific location. Assign the *macro* value, then enter the table of contents block macro (`toc::[]`) on the line in your document where you want the table of contents to appear.

toclevels:

By default, the table of contents displays level 1 and 2 section titles. Use this attribute to adjust the depth to a value from 1 through 5.

toc-class:

CSS class on the table of contents container HTML element.

media:

Enables behavior specific to a PDF media type.

media → print:

Shows the URLs for links unless the link text matches the URL.

media → prepress:

In addition to print mode also uses double-sided (mirror) page margins and automatic facing pages.

General content and formatting attributes

data-uri:

Embeds images and other media in HTML files as data-uri elements. This attribute only comes into effect if you select the value *From Attributes* in the export settings for resources.

doctype:

Document type. Currently supported are *article* and *book*.

doctype → article:

The default *doctype*. Unless you are making a book you don't need to worry about setting the *doctype*.

doctype → book:

Builds on the article doctype with the additional ability to use a top-level title as part titles. There's also the concept of a multi-part book, but the distinction from a regular book is determined by the content. A book only has chapters and special sections, whereas a multi-part book is divided by parts that each contain one or more chapters or special sections.

doctype-article:

A convenience attribute for checking whether the *doctype* is *article*.

doctype-book:

A convenience attribute for checking whether the *doctype* is *book*.

hardbreaks-option:

Preserve line breaks in all paragraphs throughout the entire document.

notitle:

Hides the document title in a document. Mutually exclusive with the *showtitle* attribute.

showtitle:

Shows the document title in an embedded document. Mutually exclusive with the *notitle* attribute.

noheader:

Turns off document header

nofooter:

Turns off document footer

table-frame:

Controls default value for frame attribute on tables.

table-grid:

Controls default value for grid attribute on tables.

table-stripes:

Controls default value for stripes attribute on tables.

Image and icon attributes

icons:

Chooses images or font icons instead of text for admonitions. Any other value is assumed to be an `icontype` and sets the value to empty (image-based icons).

icons → image:

Icons resolve to image files in the directory specified by the `iconsdir` attribute.

icons → font:

Icons are loaded from an icon font like *Font Awesome*.

iconsdir:

Location of non-font-based image icons. Defaults to the `icons` folder under `imagesdir` if `imagesdir` is specified and `iconsdir` is not specified.

icontype:

File type for image icons. Only relevant when using image-based icons.

icon-set:

For use with `icon` macros: Choose a set from which the font-based icons are taken. This attribute only has an effect if you have activated font-based icons via the `icons` attribute.

icon-set → fa:

Icon macros show symbols from the deprecated Font Awesome 4 set. In adoc Studio, this is equal to the *Regular* set. This is the default setting.

icon-set → fas:

Icon macros show symbols from the Font Awesome set *Solid*.

icon-set → fab:

Icon macros show symbols from the Font Awesome set *Brands*.

icon-set → far:

Icon macros show symbols from the Font Awesome set *Regular*.

imagesdir:

By default, media files for images or movies are picked based on the path specified in the macro's target. The text set in *imagesdir* is automatically added to the beginning of this path.

HTML styling attributes

linkcss:

Inserts a reference to the stylesheet in HTML files instead of embedding it completely. This attribute is only used if you select the value *From Attributes* in the export settings for resources.

stylesheet:

Normally, you select one of the installed styles to design your documents. If you set this attribute to the name of a CSS file in your project, this stylesheet will be used for the preview and export instead. If the style file is not in the same folder as the document, you can also specify the path to its folder in the *stylesdir* attribute.

stylesdir:

The file path optionally specified here is prefixed to the file name contained in the *stylesheet* attribute in order to localize the CSS stylesheet file in the project.

Appendix C: Regular Expressions

As the scope of a writing project increases, the search for content also becomes more complex. Instead of searching word by word in each file and possibly replacing them, you can also use a search for a pattern in the text to find relevant text passages. These are then the *regular expressions*. And because the name is somewhat unwieldy, many people shorten it to *RegEx*.

A short example clearly shows the power of this grammar:

Search: `h*.us` replaces the `*.` with one or more arbitrary characters Finds: House, Haus and even Huus. But not Hans, only words that begin with an *h* and end with *us*.

And there are many of these patterns. They can consist of one or more characters, operators or constructs. We will give you an initial overview below.

Regular Expression Metacharacters

The following tables³ describe the character expressions used by the regular expression to match patterns within a text. The pattern operators specify how many times a pattern is matched and additional matching restrictions. The last table specifies flags that can be included in the regular expression pattern that specify search behavior over multiple lines.

The table [Regular Expression Metacharacters](#) describe the character sequences used to match characters within a string.

Table 6. Regular Expression Metacharacters

Character Expression	Description
<code>\a</code>	Match a BELL, <code>\u0007</code>

3. Taken from this website: <https://developer.apple.com/documentation/foundation/nsregularexpression> ⁷

Character Expression	Description
\A	Match at the beginning of the input. Differs from ^ in that \A will not match after a new line within the input.
\b, outside of a [Set]	Match if the current position is a word boundary. Boundaries occur at the transitions between word (\w) and non-word (\W) characters, with combining marks ignored.
\b, within a [Set]	Match a BACKSPACE, \u0008.
\B	Match if the current position is not a word boundary.
\cX	Match a control-X character
\d	Match any character with the Unicode General Category of Nd (Number, Decimal Digit.)
\D	Match any character that is not a decimal digit.
\e	Match an ESCAPE, \u001B.
\E	Terminates a \Q ... \E quoted sequence.
\f	Match a FORM FEED, \u000C.
\G	Match if the current position is at the end of the previous match.
\n	Match a LINE FEED, \u000A.
\N{UNICODE CHARACTER NAME}	Match the named character.
\p{UNICODE PROPERTY NAME}	Match any character with the specified Unicode Property.
\P{UNICODE PROPERTY NAME}	Match any character not having the specified Unicode Property.
\Q	Quotes all following characters until \E.

Character Expression	Description
<code>\r</code>	Match a CARRIAGE RETURN, \u000D.
<code>\s</code>	Match a white space character. White space is defined as <code>[\t\n\f\r\p{Z}]</code> .
<code>\S</code>	Match a non-white space character.
<code>\t</code>	Match a HORIZONTAL TABULATION, \u0009.
<code>\uhhhh</code>	Match the character with the hex value hhhh.
<code>\Uhhhhhhhh</code>	Match the character with the hex value hhhhhhhh. Exactly eight hex digits must be provided, even though the largest Unicode code point is \U0010ffff.
<code>\w</code>	Match a word character. Word characters are <code>[\p{Ll}\p{Lu}\p{Lo}\p{Nd}]</code> .
<code>\W</code>	Match a non-word character.
<code>\x{hhhh}</code>	Match the character with hex value hhhh. From one to six hex digits may be supplied.
<code>\xhh</code>	Match the character with two digit hex value hh.
<code>\X</code>	Match a Grapheme Cluster.
<code>\Z</code>	Match if the current position is at the end of input, but before the final line terminator, if one exists.
<code>\z</code>	Match if the current position is at the end of input.
<code>\n</code>	Back Reference. Match whatever the nth capturing group matched. n must be a number ≥ 1 and \leq total number of capture groups in the pattern.
<code>\0ooo</code>	Match an Octal character. ooo is from one to three octal digits. 0377 is the largest allowed Octal character. The leading zero is required; it distinguishes Octal constants from back references.
<code>[pattern]</code>	Match any one character from the pattern.
<code>.</code>	Match any character. See <code>dotMatchesLineSeparators</code> and the <code>s</code> character expression in Table 4.
<code>^</code>	Match at the beginning of a line. See <code>anchorsMatchLines</code> and the <code>\m</code> character expression in Table 4.

Character Expression	Description
\$	Match at the end of a line. See anchorsMatchLines and the m character expression in Table 4.
\	Quotes the following character. Characters that must be quoted to be treated as literals are *? + [() { } ^ \$ \ . /

Regular Expression Operators

The table [Regular Expression Operators](#) defines the regular expression operators.

Table 7. Regular Expression Operators

Operator	Description
	Alternation. A B matches either A or B.
*	Match 0 or more times. Match as many times as possible.
+	Match 1 or more times. Match as many times as possible.
?	Match zero or one times. Prefer one.
{n}	Match exactly n times.
{n,}	Match at least n times. Match as many times as possible.
{n,m}	Match between n and m times. Match as many times as possible, but not more than m.
*?	Match 0 or more times. Match as few times as possible.
+?	Match 1 or more times. Match as few times as possible.
??	Match zero or one times. Prefer zero.
{n}?	Match exactly n times.
{n,}?	Match at least n times, but no more than required for an overall pattern match.
{n,m}?	Match between n and m times. Match as few times as possible, but not less than n.

Operator	Description
*+	Match 0 or more times. Match as many times as possible when first encountered. Do not retry with fewer, even if overall match fails (possessive match).
++	Match 1 or more times (possessive match).
?+	Match zero or one times (possessive match).
{n}+	Match exactly n times.
{n,}+	Match at least n times (possessive match).
{n,m}+	Match between n and m times (possessive match).
(...)	Capturing parentheses. Range of input that matched the parenthesized sub-expression is available after the match.
(?:...)	Non-capturing parentheses. Groups the included pattern, but does not provide capturing of matching text. Somewhat more efficient than capturing parentheses.
(?>...)	Atomic-match parentheses. First match of the parenthesized subexpression is the only one tried; if it does not lead to an overall pattern match, back up the search for a match to a position before the "(?>"
(?# ...)	Free-format comment (?# comment).
(?= ...)	Look-ahead assertion. True if the parenthesized pattern matches at the current input position, but does not advance the input position.
(?! ...)	Negative look-ahead assertion. True if the parenthesized pattern does not match at the current input position. Does not advance the input position.
(?< ...)	Look-behind assertion. True if the parenthesized pattern matches text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators.)
(?<! ...)	Negative Look-behind assertion. True if the parenthesized pattern does not match text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators.)

Operator	Description
(?ismwx-ismwx: ...)	Flag settings. Evaluate the parenthesized expression with the specified flags enabled or -disabled. The flags are defined in Flag Options.
(?ismwx-ismwx)	Flag settings. Change the flag settings. Changes apply to the portion of the pattern following the setting. For example, (?i) changes to a case insensitive match. The flags are defined in Flag Options.

Template Matching Format

The regular expression provides find-and-replace methods for both immutable and mutable strings using the technique of template matching. Table [Template Matching Format](#) describes the syntax.

Table 8. Template Matching Format

Character	Description
\$n	The text of capture group n will be substituted for \$n. n must be >= 0 and not greater than the number of capture groups. A \$ not followed by a digit has no special meaning, and will appear in the substitution text as itself, a \$.
\	Treat the following character as a literal, suppressing any special meaning. Backslash escaping in substitution text is only required for '\$' and '\', but may be used on any other character without bad effects.

Flag Options

The following flags control various aspects of regular expression matching. These flag values may be specified within the pattern using the (?ismx-ismx) pattern options. Equivalent behaviors can be specified for the entire pattern when a regular expression is initialized.

Table 9. Flag Options

Flag (Pattern)	Description
i	If set, matching will take place in a case-insensitive manner.
x	If set, allow use of white space and #comments within patterns

Flag (Pattern)	Description
s	If set, a "." in a pattern will match a line terminator in the input text. By default, it will not. Note that a carriage-return / line-feed pair in text behave as a single line terminator, and will match a single "." in a regular expression pattern
m	Control the behavior of "^" and "\$" in a pattern. By default these will only match at the start and end, respectively, of the input text. If this flag is set, "^" and "\$" will also match at the start and end of each line within the input text.
w	Controls the behavior of \b in a pattern. If set, word boundaries are found according to the definitions of word found in Unicode UAX 29, Text Boundaries. By default, word boundaries are identified by means of a simple classification of characters as either "word" or "non-word", which approximates traditional regular expression behavior. The results obtained with the two options can be quite different in runs of spaces and other non-word characters.

Appendix D: Glossary

The glossary provides translations of some terms from adoc Studio, originally derived from the vocabulary of AsciiDoc and therefore in English. These translations result in a mixture of English and German terms. The glossary helps you navigate through this language blend.

Inhalt

[[A](#)|[B](#)|[C](#)|[D](#)|[E](#)|[F](#)|[G](#)|[H](#)|[I](#)|[J](#)|[K](#)|[L](#)|[M](#)|[N](#)|[O](#)|[P](#)|[Q](#)|[R](#)|[S](#)|[T](#)|[U](#)|[V](#)|[W](#)|[X](#)|[Y](#)|[Z](#)]

A

Abstract:

An abstract is a brief overview or summary of a document. Sometimes it is used synonymously with a [preamble](#).

Admonition:

Admonitions are used to draw attention to specific statements by taking the block or paragraph out of the content flow. They are defined with one of five types (NOTE, TIP, IMPORTANT, CAUTION, WARNING) that determine their appearance. An admonition can be defined as a [paragraph](#) or a [block](#).

adoc:

The abbreviation for this app and the file extension for [AsciiDoc](#) text files.

adoc Studio:

The name of this app.

Anchor:

A jump target defined by the [ID attribute](#).

Appendix:

The appendix section style can be used in [books](#) and [articles](#) and can have additional subsections. Although the AsciiDoc structure allows for appendices to be placed anywhere, it is customary to include them at the end of a document.

Article:

The standard [doctype](#) in adoc Studio. This includes optional sections such as [Appendix](#), [Abstract](#), [Bibliography](#), [Glossary](#), and [Index](#).

ASCII:

An acronym for [American Standard Code for Information Interchange](#)⁷. This standard defines 128 characters, consisting of 33 non-printable and 95 printable characters, including spaces. In AsciiDoc, however, ASCII stands for "everything is text - even emojis." AsciiDoc uses the full Unicode character set.

AsciiDoc:

A mature, lightweight, and semantic markup language mainly designed for structured document creation. You can convert AsciiDoc files to HTML or PDF using programs like adoc Studio or [Asciidoctor](#). The AsciiDoc language was created by Stuart Rackham in 2002. A reimplementaion is [Asciidoctor](#), which has been developed by Dan Allen since 2013. Since 2020, the Eclipse Foundation, along with the AsciiDoc Working Group, has been working on standardizing AsciiDoc.

Asciidoctor:

A [command-line](#) tool for parsing [AsciiDoc](#) files and converting them to output formats such as [HTML](#) and others. To create PDF documents, [Asciidoctor PDF](#) is required. Essentially, it is a similar program to adoc Studio, but without the [project integration](#), [editor](#), [text completion](#), and [preview](#) features. Asciidoctor is open source and has been developed and maintained by Dan Allen since 2013.

Asciidoctor PDF:

Is a [command line](#) tool for [parse](#) of [AsciiDoc](#) files and their conversion into the output format [PDF](#). Asciidoctor PDF is OpenSource and is developed by Dan Allen.

Attribute:

Used in AsciiDoc to configure documents and store variables. A document attribute is available from the point it is defined in the document. However, some attributes must appear at the beginning of a document. Attribute entries are also frequently used to toggle features. An attribute entry consists of two parts: an attribute name and an attribute value.

Attribute List (or attrlist):

The definition of attributes used for an element is called an attrlist. An attribute list is always enclosed in a pair of square brackets. This applies to both [element attributes](#) and attributes in a block or inline macro.

Author:

The writer of a document. In [AsciiDoc](#) documents, the author can be specified below the title and is automatically stored in the `:author:` attribute. Additional [attributes](#) related to the author are also possible.

B

Bibliography:

An independent listing of references or the act or process of creating such a listing. In adoc Studio, it can be used in [books](#) and [articles](#).

Block:

In an AsciiDoc document, blocks define the structure of the document. Some blocks can contain other blocks, making the document structure inherently hierarchical (i.e., a tree structure). You can view this block structure in the preview, for example, by enabling the automatic table of contents. Examples of blocks include paragraphs, sections, lists, delimited blocks, tables, and admonitions.



Block Name:

Refers to custom blocks that can associate any name with one or more contexts. It serves a similar function to the style for a built-in block, such as an [admonition block](#).

Block Style:

Additional information that specializes the context of a [block](#). For example, a source block can be additionally annotated with the language of the source code used.

Bold:

In bold text, the weight of the text is increased by using a thicker and/or darker font compared to regular text. The formatting character for **bold** is an asterisk (*) before and after the word (keyboard:  + ).

Boolean Attribute:

A built-in attribute that functions as a toggle. It can only specify two states: *true* or *false* – *on* or *off*. The purpose is to activate a function or behavior.

Book:

Builds on the [doctype](#) of [article](#) and additionally provides the option to use a top-level title as a sub-title. It includes the [appendix](#), [dedication](#), [preface](#), [bibliography](#), [glossary](#), [index](#), and [colophon](#). A book can contain either only [chapters](#) or, additionally, various [parts](#), which in turn can contain one or more [chapters](#).

Bounded Blocks:

A bounded block is a section of content enclosed by a pair of congruent line-level boundary markers on both sides. A bounded block is used either to encapsulate other blocks (e.g., multiple paragraphs) or to define the content model of the content (e.g., a literal block).

Built-in Attribute:

Is a document attribute used for processing the [AsciiDoc](#) document. It can control integration, styling, localization, and more.

C**Chapter:**

Is a content-separating subdivision in texts. In adoc Studio we speak of chapters for the files in [collective documents](#).

CLI:

Is the abbreviation for command-line interface. It is often also referred to as the command line, console or terminal. It refers to an input area for controlling software. This typically takes place in text mode.

Composite Document:

In adoc Studio, a composite document is a distinct document type and contrasts with [standalone documents](#). It collects many [chapters](#) into a single large document. Within a composite document, many [folders](#) and [media folders](#) can be created for further structuring. All folders can be selected to combine the underlying chapters in the [preview](#). [Attributes](#) defined anywhere in the composite document can be used in any chapter within the composite document.

Colophon (also subscription):

Is an element of a [book](#) and is usually at the end. It contains information about the content, author, place, time, producer, publisher and production details of the publication.

Content model:

The content model of a block determines what type of content the block can have and how this content is processed (e.g., simple, compound, literal, raw, etc.).

Converter:

Is a software component that an AsciiDoc processor calls to convert a [AsciiDoc](#) document to a specific output format.

Cross-reference (crosslink):

Is a [link](#) from a specific position to another position in the document, which is identified by a [anchor](#).

CSS (Cascading Style Sheets):

Is a set of instructions or a template with settings for designing a document. A document or website with the same content can be displayed completely differently using different CSS. In adoc Studio, both [HTML](#) and [PDF](#) are designed with CSS. In addition, custom .css files can be transferred to a [project](#) and integrated using `:stylesheet:`.

Custom Attribute:

A document attribute defined by the author used for storing reusable content and controlling conditional inclusion.

D**Dedication:**

Used to express gratitude to third parties. In [AsciiDoc](#) the dedication is a separate [block type](#), which is preferably used in [books](#).

DocBook:

Is a document format defined in a Document Type Definition (DTD) for SGML and XML. It is suitable for creating books, articles, and documentation in technical fields (hardware or software). DocBook is an open standard maintained by the [Organization for the Advancement of Structured Information Standards](#)[↗] (OASIS).

Document (or standalone document):

A file in adoc Studio is referred to as a document. It is in plain text format. A document can consist of a single sentence (or even a single character, to be academic).

Document Attribute:

Used to configure behavior in the processor or convey information about the document and its environment. An addition to document attributes is [element attributes](#). There are [built-in](#) and [custom](#) attributes.

Docs as Code:

Refers to the philosophy that documentation should be written using the same methods as software code. This means following the same workflows as development teams and integrating into the product team. It enables a culture where both authors and developers take responsibility for documentation and work together to make it as good as possible.

Document Header:

An AsciiDoc document begins with a document header. The document header can contain the document title, author and revision information, document-wide attributes, and other document metadata. Furthermore, all [document attributes](#) must be defined in the document header.

Document Type (Doctype):

Specifies the expected structure of an AsciiDoc document. adoc Studio offers [articles](#) (as the default) and [books](#).

E**Editor:**

Is a program or component for creating and modifying text. In adoc Studio, the editor is located in the middle window area. It provides special features such as intelligent [text completion](#) to assist while writing. The editor's behavior can be customized in adoc Studio through a program setting.

Element:

Is an identifiable part of the content in a document. An [AsciiDoc](#) document is always a composition of all elements contained within it.

Element Attribute:

Define integrated and user-defined settings and metadata. They can be applied to an individual block element or inline element in a document (including [macros](#)).

Encoding:

Most AsciiDoc processors, including adoc Studio, assume that the text in the file uses UTF-8 encoding. UTF-16 encodings are only supported if the file begins with a BOM.

Environment attribute:

An environment attribute is a dynamic document attribute that refers to or provides information about the runtime environment.

Export:

The output of an adoc file is in a format such as [HTML](#) or [PDF](#). An export can have its own [styles](#) and [attributes](#), which can be assigned either in the [AsciiDoc](#) document or in the export dialog. The output can be saved to a file and optionally shared directly with other programs.

F**Fixed line breaks:**

Since adjacent text lines in AsciiDoc are merged into a paragraph during conversion, you can place each sentence or phrase on its own line. Line breaks are not displayed in the output. To separate paragraphs from each other, simply use an additional empty line. This method is often used in the [Docs as Code](#) workflow. However, if you want to preserve line breaks within a paragraph, you can use a space followed by a plus sign (+) or set the "Hard Breaks" option for the paragraph.

Folder:

Is a directory within a data structure. Any number of folders can be created in adoc Studio - even nested. A folder can be selected in [collective documents](#) to display the [chapter](#) contained therein in the [preview](#) combined.

Formatting mark and pairs:

A formatting mark is a symbolic character such as *,_, or ~ that indicates the inline style the AsciiDoc converter should apply to the text. Formatting characters always come in pairs. A formatting pair consists of identical opening and closing characters that enclose the text to be formatted. The formatted text (i.e., the text enclosed by a formatting pair) can span multiple consecutive lines. The opening mark indicates where the formatting should begin, and the closing mark indicates where the formatting should end.

G**Git:**

Git is free software for distributed [version management](#) of files, initiated by Linus Torvalds in 2005. All [texts](#) and [media](#) from adoc Studio can be versioned in a Git repository.

Glossary:

A glossary is a list of words with accompanying explanations or translations. When included as an appendix in a work, it is also referred to as a word list, and as a standalone glossary, it is known as a dictionary. By the way, you are currently reading the glossary of adoc Studio.

H

Header Attribute:

A header attribute is a document attribute defined in the [document header](#) of a document. It is visible to all nodes in the document and is often used for global attributes, e.g., for the icon mode:

```
:icons: font .
```

HTML:

Hypertext Markup Language (HTML) is a text-based markup language for structuring electronic documents, including text with [hyperlinks](#), images, and other content. HTML documents form the foundation of the World Wide Web and are rendered by web browsers.

I

ID Attribute or Anchor:

With an anchor, you can assign a unique name (i.e., identifier) to a block or inline element. It is used to identify the element for linking purposes or styling.

Index:

You can explicitly mark index terms in AsciiDoc content. Index terms form a controlled vocabulary that allows you to navigate the document based on an index.



Inline:

The inline [doctype](#) allows the AsciiDoc processor to cover the entire range of applications, from unstructured (inline) text to complete standalone documents.

Inline Element:

An inline element is a phrase (i.e., a span of content) within a block element or one of its attributes in an AsciiDoc document. They are used for text formatting and performing various types of text substitutions. Inline elements and the syntax for inline elements are defined in configuration files. Conversion of inline documents is not yet implemented in adoc Studio.

Italic:

Text is often italicized to emphasize a word or phrase, quote a speaker, or introduce a concept. Italic type is slanted slightly to the right and, depending on the font, may have italic swashes and flourishes. The formatting character for *italic* is an underscore ( + ).

Integrated Writing Environment (IWE for short):

Is software that provides comprehensive writing and knowledge management functions for authors and information workers. So does adoc Studio. IWEs allow authors and information workers to perform a variety of tasks related to the document in the IWE in a single environment. In the best case, principles such as [Docs as Code](#) are also supported. This ensures a distraction-free workspace and an optimized writing environment.

J

K

Keywords:

The `:keywords:` attribute contains a list of comma-separated values that are assigned to the HTML `<meta>` element.

L

Line:

[AsciiDoc](#) is a line-oriented language, so the line is an important construct. A line is defined as text that is separated on both sides either by a line break or by the boundary of the document. Many aspects of the syntax must take up an entire line. When working according to [Docs as Code](#), the single line is particularly important as it is easier to recognize changes in a [Git](#).

Link:

tbd

- links
- hyperlinks
- crosslink

List block:

Is defined by a group of sibling list elements, each characterized by the same marker.

List continuation:

A list continuation is a plus sign (`+`) in a line that connects neighboring lines of text to form a list element.

M

Macro:

Is a syntax for representing non-text elements or a syntax that transforms into text using the provided metadata. See [Macro](#)[↗] to learn more about the significance of this term.

Macro Attribute:

Is an attribute associated with a block or inline macro. These attributes can influence the processing of the macro but cannot be referenced via an attribute reference.

Manual page:

A man page, short for "manual page," is a text-oriented form of software documentation commonly accompanying software on UNIX and Unix-like operating systems. Its formalized structure allows the `man` command to display a formatted document in a [terminal](#). Writing a man page in AsciiDoc has the advantage of being convertible into various formats, including HTML and PDF, making the source of the man page reusable. Conversion to "man pages" is not yet implemented in adoc Studio.

Markdown:

A simplified [markup language](#) released in December 2004 by John Gruber and Aaron Swartz.

Markup Language:

A markup language is a machine-readable language for structuring and formatting text and other data. The most well-known example is Hypertext Markup Language ([HTML](#)), the core language of the World Wide Web.

Marking:

Another way to draw attention to text is to highlight it with a marker. This semantic style is used for referencing or notational purposes or to emphasize the significance of an important topic or point. The formatting character for **Marking** is a hash # (`#`).



Media Folder:

In adoc Studio, the media folder has some special functions. When selected, all contained media items are displayed as previews. You can create as many media folders as needed.

Metadata:

Are structured data containing overarching information about a resource, such as books, web documents, videos, or images. Regarding a document, it can include a description of the document, keywords, and custom information that can be assigned as [attributes](#) in the [document header](#). When converted to [HTML](#), the values of these attributes correspond to the elements found in the `<head>` section of an HTML document.

Monospace:

In technical content, text often needs to be styled to indicate a command or source code. Such text is typically highlighted using a fixed-width font (i.e., Monospace). The formatting character for **Monospace** is a backtick ` ( + ).

N**O****Outline:**

An outline is the division of a whole into several structural parts or sections that are largely self-contained but cannot be removed from the whole without making it incomplete. adoc Studio contains multiple outlines: the file navigator and section navigator in the sidebar, the heading structure above the editor, and more.

P**Paragraph:**

In most documents, the paragraph is the most important block type. Therefore, you don't need to use special markings or attributes to create paragraphs. You can simply start typing sentences. Whether you use a separate [line](#) for each sentence or write them one after the other, the content will always form a paragraph. A blank line separates paragraphs.

Parser:

A parser is a computer program in computer science responsible for parsing and converting input into a more suitable format for further processing. For the technically inclined: The parser in adoc Studio is written in Swift, inspired by [AsciiDoc](#). It is a migration, not a direct copy or port.

Part:

In the context of a multipart book, a part is an additional subdivision. A part can contain its own [chapters](#) or [appendices](#).

PDF (Portable Document Format):

Is a platform-independent file format developed and published by Adobe Inc. in 1992. Electronic documents in this file format can be faithfully reproduced independent of the original application, operating system, or hardware platform.

Plaintext:

In IT, plaintext refers to data that is human-readable without any formatting and can be directly converted into text using characters. The [AsciiDoc](#) documents in adoc Studio use plaintext.

Preamble:

Is the content between the end of the document header and the first section title in the document body. A preamble is optional in [AsciiDoc](#).

Predefined attribute:

A predefined attribute is a document attribute that has been defined for reasons of convenience. It is often used to insert special characters for the content.

Preview:

In the preview, adoc Studio displays all content from the [editor](#) in various formats. [HTML](#), [PDF](#) and [RTF text](#) are available. Each output format can be formatted with a style.

Preface:

A preface is a special section that precedes the first chapter of a book or part of a book. In [AsciiDoc](#), the preface is a separate [block type](#), which is preferably used in [books](#).

Preprocessor Directive:

Is a function that controls the lines input into the parser. A conditional preprocessor directive can configure lines to be included or excluded based on the presence of an attribute (`ifdef`, `ifndef`) or any other condition (`ifeval`). An include directive can add additional lines from another document to the document.

Products:

Are a logical grouping of many [exports](#). Within a product, many exports can be created, each of which can have its own [styles](#) and [attributes](#) and output formats.

Product Styles:

In adoc Studio, text is separated from styling. Therefore, a technique is needed to combine both. This task is accomplished using *product styles*. A product style is a [file bundle](#)⁷ that includes at least an information file and the style file in [CSS format](#). Optionally, additional resources such as media or fonts can be included.

Project:

In adoc Studio, a project is a summary of all [AsciiDoc](#) files and [folders](#), including [media folders](#). Additionally, projects can also contain other files such as [CSS](#) files and others. It is organized and managed in adoc Studio through the project navigator. Additionally, continuous synchronization with the operating system's directory takes place, allowing the project to be modified in the Finder.

Q

Quote, quoted text:

Text surrounded by special punctuation to *quote* other authors, or to give the text a special meaning.

R

Role:

[blocks](#) and most other [inline elements](#) can be assigned one or more roles with the [attribute role](#). A role adds additional semantics to an element, for example to assign an additional style (via a [CSS](#) selector).

RTF (Rich Text Format):

Is a proprietary file format for texts that was introduced by Microsoft in 1987. It can be used as an exchange format between word processing programs from different manufacturers on different operating systems. In adoc Studio, the [preview](#) can be set to text, which corresponds to the Rich Text Format. This format can also be selected for the [export](#) and within the [products](#).

S

Section:

Divides an [AsciiDoc](#) document into a hierarchy of content. A section is defined by a [section title](#) prefixed with one or more equals signs. The section encompasses all content that follows the section title until the next sibling or parent section title or until the end of the document.

Section Title:

Marks the beginning of a section and simultaneously serves as a heading for that section. The section level is determined by the prefixed section marker (equals signs). In adoc Studio, the equals signs can be displayed in the left margin of the [editor](#).

Source code:

Is a human-readable text of a computer program written in a programming language. In the [Docs as Code](#) working principle, the [AsciiDoc](#) text is also often referred to as source code.

Stylesheet:

Most closely resembles a template. The fundamental idea is to separate text from presentation. The program that evaluates the stylesheet interprets the assigned data (text, tables, graphics, etc.) and formats them according to predefined rules. [Cascading Stylesheets](#) are perhaps the most well-

known example of a stylesheet. In adoc Studio, custom stylesheets can be placed in the project as .css files and used with the `attribute :stylesheet:`.

Subscript (English):

Subscript text is often used in mathematical expressions and chemical formulas. The formatting character for `subscript` is a tilde ~ (`~` + `n`).

Superscript:

Superscript text is often used in mathematical expressions and chemical formulas. The [formatting character](#) for `superscript` is a caret (`^`).

Syntax:

In general, it refers to a rule system for combining different characters into natural or artificial systems. The rules of syntax contrast with the interpretation rules of semantics. The command set of a language, such as [AsciiDoc](#), is often referred to as its syntax.

T

Text:

Enclosed by markers, delimiters, and metadata lines, text is the core of a document and the reason why [AsciiDoc](#) syntax gives it so much breathing room. Text is most commonly found within the lines of a [block](#) (e.g., a paragraph), in the block title (e.g., section title), and in list items, among other places. Text is subject to substitutions. Substitutions interpret markup as text formatting, replace macros with text or non-text elements, expand attribute references, and perform other types of text replacements. By default, normal text is subject to all substitutions unless otherwise specified. Literal text is subject to a minimal set of substitutions to ensure it is displayed in the output as it appears in the source. It is also possible to disable all substitutions to pass the text through unchanged (i.e., as unprocessed) to the output. Parsing the text involves a mixture of inline elements and other forms of transformations.

Text Completion (intelligent or automatic):

A technique that, after entering two to three letters in the [editor](#), recognizes that contextually relevant [attributes](#), [macros](#), or other possibilities are available at the current typing position. The key is that it doesn't block the writer's flow but rather provides support. It is often used to save time.

Text Formatting:

[AsciiDoc](#) provides a set of formatting characters that allow you to visually emphasize your document and use typographic punctuation. You can build on these basic formatting characters using built-in and custom roles. Examples include: [bold](#), [italic](#), [highlighting](#), etc.

U

UTF-8:

see [Encoding](#)

V

Version management:

Version management (like [Git](#)) is used to manage multiple versions of a document in a space-saving manner. With most version management systems, only the changes that are necessary to create a specific version are saved alongside the original document. In addition, version management makes it possible to work together on the same document.

W

X

Y

Z