Tutorial : How to use Scribe's markup language

This document provides an overview of the syntax and elements of Scribe's markup language:

General Structure

The language has four main types of structures: **Paragraph**, **Blocks**, **Multiline blocks** and **Inline Markup**.

Paragraphs

Paragraphs are the most common elements in Scribe. A paragraph is a sequence of text that can be added anywhere in a document.

- Each paragraph can contain plain text and inline markup, such as formatting styles and links.
- You can break the text of a paragraph into several lines, as long as there is no blank space between them.
- Tabs, leading or trailing spaces in a paragraph will be ignored
- Sequences of spaces or tabs within a paragraph will be converted into a single space.
- You can use "///" to create a line break within a paragraph.

Formatting	Result

`Lorem ipsum {dolor}[b] sit amet, {consectetur}[i] adipiscing elit.`
 	
`Nullam imperdiet     mi a magna tincidunt, nec ornare sem.`
	Lorem ipsum **dolor** sit amet, *consectetur* adipiscing elit.
	
	Nullam imperdiet mi a magna tincidunt, nec ornare sem.

Blocks

Blocks are used to define distinct elements in the document. Each block can contain only one paragraph.

▾ Headers

There are 6 levels of headers:

```
[#] h1
[##] h2
[###] h3
[####] h4
[#####] h5
[######] h6
```

h1

h2

h3

h4

h5

h6

▾ Unordered lists

Unordered lists are created with the symbol (`*`):

```
[*] First item
[*] Second Item
[*] Third item
```

- First Item
- Second Item
- Third Item

▾ Ordered lists

Ordered lists are formed by specifying a number followed by a dot:

```
[1.] First item
[2.] Second Item
[3.] Third item
```

1. First item
2. Second Item
3. Third Item

▾ Quotes

Quotes are made with the word "quote":

```
[quote] Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc mollis diam neque, scelerisque
pulvinar enim pellentesque id.
```

Result:

> *Lorem ipsum dolor sit amet, consectetur adipiscing elit. nunc mollis diam neque, scelerisque pulvinar enim pellentesque id.*

You can also specify the author of a quotation in the following way:

```
[quote=Author] Lorem ipsum dolor sit amet.
```

Result:

> *Lorem ipsum dolor sit amet.*
>
> ***- Author***

▾ Code

Code blocks will not be formatted. All spaces, tabs, lines, and commands will be interpreted literally:

```
public int Factorial(int num)
{
    var result = 1;
    while (num > 1)
    {
        result *= num;
        num = num - 1;
    }
    return result;
}
```

Code blocks cannot contain other blocks, but they can have inline markup.

▾ Task list

A task can be pending:

```
[-] Pending task
```

☐ Pending task

Or complete:

```
[x] Complete Task
```

☑ ~~Complete Task~~

▾ Images

An image can be specified as follows:

```
[img=file_path]
```

You can also specify the size of an image as a percentage:

```
[img(75%)=file_path]
```

▾ Toggle list

In a toggle list you can click on the triangle icon to reveal or hide the list items:

```
[toggle] List
```

▸ List

An empty list is not very useful. See the Multiline Blocks section to learn how to add more than one item to a block.

In the case of a multiline list, the first element of the block will be considered as the header of the list:

```
[toggle]%
    List

    Item 1

    Item 2
%
```

Result:

▾ List

Item 1

Item 2

▾ Callout blocks

A callout block is used to highlight important information in a text. There are several types of callouts, all of which are prefixed by (`::`):

```
[::callout] Standard callout
[::favorite] Favorite
[::question] Question
[::success] Success
[::failure] Failure
[::warning] Danger
[::note] Note
```

Result:

✱ Standard callout

♥ Favorite

? Question

✓ Success

✗ Failure

! Danger

✎ Note

▾ Indentation Block

An indentation block simply indents its contents:

```
[>>] Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc mollis diam neque,
scelerisque pulvinar enim pellentesque id.
```

Result:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. nunc mollis diam neque, scelerisque pulvinar enim pellentesque id.

▾ Tables

Tables are a special type of block. They are the only blocks that are exclusively multiline.

- A table is defined with `table` block alongside with `cell` blocks.
- Each `cell` block represents a cell of the table. Each cell is inserted into a a different column.
- Use dividers to create a new row in the table.
- The first row of the table will be considered as the header of the table.
- Anything inside a `table` that is not a `cell` block or a divider will ignored.

Example:

```
[table]%
    [cell] Header 1
    [cell] Header 2

    ===

    [cell] Item 1.1
    [cell] Item 1.2

    ===

    [cell] Item 2.1
    [cell] Item 2.2
%
```

Result:

Header 1	Header 2
Item 1.1	Item 2.1
Item 1.2	Item 2.2

Multiline blocks

Multiline blocks are an extended version of normal blocks, and can be used to include other blocks and paragraphs within the same element.

To create a multiline block use %:

```
[*]%
    Paragraph 1

    Paragraph 2

    Paragraph 3
%
```

Result:

- Paragraph 1

  Paragraph 2

  Paragraph 3

All blocks can be turned into multiline blocks.

You can use any blocks inside a multiline block, including other multiline blocks:

```
[-]%
    Task 1:

    [-]%
        Subtask

        Text.
    %

    Text.
%

[-] Task 2
```

Result:

☐ Task 1:

  ☐ Subtask

    Text.

  Text.

☐ Task 2

Inline Markup

With inline markup you can format the text of a paragraph.

The markup follows the format:

```
{text}[modifier1, modifier2, ...]
```

Where the modifiers can be:

Modifier	Markup	Result
b	{text}[b]	**text**
i	{text}[i]	*text*
u	{text}[u]	<u>text</u>
s	{text}[s]	~~text~~
super	a{x}[super]	a^{x}
sub	a{x}[sub]	a_{x}
code	{text}[code]	`text`
spoiler	click to reveal {secret text}[spoiler]	click to reveal

You can combine different modifiers:

```
{text in bold, italic, and underlined}[b,i,u]
```

Result:

<u>text in bold, italic, and underlined</u>

There are also two types of special modifiers: **colors** and **links**.

▾ Colors

To specify colors for text you can use `foreg` (for *foreground*) and `backg` (for *background*):

```
{text in red}[foreg=#FF0000]
{text with red background}[backg=#FF0000]
```

text in red

text with red background

You can also specify a color with transparency:

```
{text with transparency}[foreg=#AAFF0000]
```

text with transparency

The colors must be in hexadecimal format, but there are some predefined names that you can use. These are:

- `black`, `white`, `gray`, `orange`, `yellow`, `green`, `blue`, `purple`, `pink` and `red`.

Example:

```
{text in blue}[foreg=#blue]
```

text in blue

▾ Links

To create links you can use:

```
{text with link}[link=https://example.com]
```

text with link

You can create links to both files and web pages. If a link cannot be created, the text will appear in gray:

```
{text with wrong link}[link=httpBlaBla://example.com]
```

text with wrong link

In addition, with a special markup you can create links to other documents in the same folder or to labels.

```
{link to document}[link=doc:New Document]
```

link to document

```
{link to label}[link=@label]
```

link to label

Special Elements

Apart from these basic elements, there are also some special elements. These are:

▾ Dividers

Dividers are horizontal lines that divide the content of a document.

To create a divider, simply use a sequence of one or more `=` characters on a line:

```
=====
```

To create partial dividers use 1 to 4 characters:

```
====
===
==
=
```

Result:

▾ Progress Bars

Create a progress bar using the following combination:

```
(ooo...)
```

■■■□□□ 50%

Where the `o` characters are the filled squares, and the `.` characters are the empty squares.

▾ Labels

Labels are invisible nodes that can be applied throughout a document.

```
@label=label-example
```

You can create a link to these labels, as explained in the inline markup section.

```
{link to label}[link=@label-example]
```

Other Commands

You can use the `$` character to ignore a markup, both for blocks and inlines.

For example:

```
{text}[b]
```

The above markup will be rendered as plain text, without any formatting:

{text}[b]

If the `$` character is used anywhere else, it will simply be ignored. If you really want to insert this character, repeat it twice:

```
$$
```

Result:

`$`