Scripst Documentation

Article Style Set

AnZrew AnZreww AnZrewww

2025-03-11

Abstract: Scripst is a simple and easy-to-use Typst language template, suitable for various scenarios such as daily documents, assignments, notes, papers, etc.

Keywords: Scripst, Typst, template

Contents

Typst is a simple document generation language with syntax similar to lightweight Markdown markup. Using appropriate `set` and `show` commands, you can highly customise the style of your documents.

Scripst is a simple and easy-to-use Typst language template, suitable for various scenarios such as daily documents, assignments, notes, papers, etc.

I Typesetting Typst Documents with Scripst

1.1 Using Typst

Typst is a lighter language to use compared to LaTeX. Once the template is written, you can complete the document writing with lightweight markup similar to Markdown.

Compared to LaTeX, Typst has the following advantages:

- Extremely fast compilation speed
- Simple and lightweight syntax
- Strong code extensibility
- Easier mathematical formula input
- ...

Therefore, Typst is very suitable for writing lightweight daily documents. You can get even better typesetting results than LaTeX with the time cost of writing Markdown.

You can install Typst in the following ways:

```
sudo apt install typst # Debian/Ubuntu
sudo pacman -S typst # Arch Linux
winget install --id Typst.Typst # Windows
brew install typst # macOS
```

You can also find more information in the Typst GitHub repository.

1.2 Using Scripst

Based on Typst, Scripst provides some simple templates for convenient daily document generation.

1.2.1 Online Usage

Scripst Package has already been submitted to the community. If network available, you can directly use

```
#import "@preview/scripst:1.1.1": *
```

to import the Scripst templates in your document.

You can also use `typst init` to create a new project with the template:

```
typst init @preview/scripst:1.1.1 project_name
```

This method does not require downloading the template files, just import them in the document.

1.2.2 Offline Usage

Using extracted files

You can find and download the Scripst templates in the Scripst GitHub repository.

You can choose `<> code` → `Download ZIP` to download the Scripst templates. When using them, just place the template files in your document directory and import the template files at the beginning of your document.

> ⚠
> Consider the project directory structure to correctly import the template files.
>
> ```
> project/
>  ├──── src/
>  |     ├──── main.typ
>  |     ├──── ...
>  |     └──── components.typ
>  ├──── pic/
>  |     ├──── ...
>  ├──── main.typ
>  ├──── chap1.typ
>  ├──── chap2.typ
>  ├──── ...
> ```
>
> If the project directory structure is as shown above, then the way to import the template files in `main.typ` should be:
>
> ```
> #import "src/main.typ": *
> ```

The advantage of this method is that you can adjust some parameters in the template at any time. Since the template is designed modularly, you can easily find and modify the parts you need to change.

Local package management

A better way is to refer to the official local package management documentation and place the template files in the local package management directory `{data-dir}/typst/packages/{namespace}/{name}/{version}`, so you can use the Scripst templates anywhere.

Of course, you don't have to worry about not being able to modify the template files. You can directly use `#set, #show` commands in the document to override some parameters in the template.

For example, the template should be placed in

```
~/.local/share/typst/packages/preview/scripst/1.1.1            # in Linux
%APPDATA%\typst\packages\preview\scripst\1.1.1                 # in Windows
~/Library/Application Support/typst/packages/local/scripst/1.1.1  # macOS
```

You can execute the following command:

```
```

```
cd ~/.local/share/typst/packages/preview/scripst/1.1.1
git clone https://github.com/An-314/scripst.git 1.1.1
```

If the directory structure is like this, then the way to import the template files in the document should be:

```
#import "@preview/scripst:1.1.1": *
```

The advantage of this is that you can directly use `typst init` to create a new project with the template:

```
typst init @preview/scripst:1.1.1 project_name
```

After importing the template, create an `article` file in this way:

```
#show: scripst.with(
  title: [How to Use Scripst],
  info: [This is the article template],
  author: ("Author1", "Author2", "Author3"),
  time: datetime.today().display(),
  abstract: [Abstract],
  keywords: ("Keyword1", "Keyword2", "Keyword3"),
  contents: true,
  content-depth: 3,
  matheq-depth: 2,
  counter-depth: 3,
  header: true,
  lang: "en",
)
```

See Section 2 for the meaning of these parameters.

Then you can start writing your document.

II Template Parameter Description

Scripst template provides some parameters to customise the style of the document.

```
#let scripst(
  template: "article",  // str: ("article", "book", "report")
  title: "",            // str, content, none
  info: "",             // str, content, none
  author: (),           // array
  time: "",             // str, content, none
  abstract: none,       // str, content, none
  keywords: (),         // array
  font-size: 11pt,      // length
  contents: false,       // bool
  content-depth: 2,      // int
  matheq-depth: 2,       // int: (1, 2, 3)
  counter-depth: 3,      // int: (1, 2, 3)
  cb-counter-depth: 2,   // int: (1, 2, 3)
  header: true,          // bool
  lang: "en",            // str: ("zh", "en", "fr", ...)
  body,
) = {
  ...
}
```

2.1 template

Parameter	Type	Optional Values	Default Value	Description
template	`str`	`("article", "book", "report")`	`"article"`	Template type

Currently, Scripst provides three templates: article, book, and report.

This template uses the article template.

- article: Suitable for daily documents, assignments, tiny notes, light papers, etc.
- book: Suitable for books, course notes, etc.
- report: Suitable for lab reports, papers, etc.

Passing other strings will cause a `panic`: `"Unknown template!"`.

2.2 title

Parameter	Type	Default Value	Description
title	`content`, `str`, `none`	`""`	Document title

The title of the document. (If not empty) it will appear at the beginning and in the header of the document.

2.3 info

Parameter	Type	Default Value	Description
info	`content`, `str`, `none`	`""`	Document information

The information of the document. (If not empty) it will appear at the beginning and in the header of the document. It can be used as a subtitle or supplementary information for the article.

2.4 author

Parameter	Type	Default Value	Description
author	`str`, `content`, `array`, `none`	`()`	Document authors

The authors of the document. Pass a list of `str` or `content`. Or, simply pass a `str` or `content` object.

Note
Note, if there is only one author, you can simply pass a `str` or `content`, and for multiple authors, a list of one `str` or `content`, for example: `author: ("Author I", "Author II")`

It will be displayed at the beginning of the article with $\min(\#\text{authors}, 3)$ authors per line.

2.5 time

Parameter	Type	Default Value	Description
time	`content`, `str`, `none`	`""`	Document time

The time of the document. It will appear at the beginning and in the header of the document.

You can choose to use Typst's `datetime` to get or format the time, such as today's date:

```
datetime.today().display()
```

2.6 abstract

Parameter	Type	Default Value	Description
abstract	`content`, `str`, `none`	`none`	Document abstract

The abstract of the document. (If not empty) it will appear at the beginning of the document.

It is recommended to define a `content` before using the abstract, for example:

```
#let abstract = [
  This is a simple document template used to generate simple daily documents
to meet the needs of documents, assignments, notes, papers, etc.
]

#show: scripst.with(
  ...
  abstract: abstract,
  ...
)
```

Then pass it to the `abstract` parameter.

2.7 keywords

Parameter	Type	Default Value	Description
keywords	`array`	`()`	Document keywords

The keywords of the document. Pass a list of `str` or `content`.

Like `author`, the parameter is a list, not a string.

Keywords will only appear at the beginning of the document if `abstract` is not empty.

2.8 font-size

Parameter	Type	Default Value	Description
font-size	`length`	`11pt`	Document font size

The font size of the document. The default is `11pt`.

Refer to the `length` type values, you can pass `pt`, `mm`, `cm`, `in`, `em`, etc.

2.9 contents

Parameter	Type	Default Value	Description
contents	`bool`	`false`	Whether to generate a table of contents

Whether to generate a table of contents. The default is `false`.

2.10 content-depth

Parameter	Type	Default Value	Description
content-depth	`int`	`2`	Depth of the table of contents

The depth of the table of contents. The default is `2`.

2.11 matheq-depth

Parameter	Type	Optional Values	Default Value	Description
matheq-depth	`int`	`1`, `2`, `3`	`2`	Depth of math equation numbering

The depth of math equation numbering. The default is `2`.

> **Note**
> For detailed behavior of counters, see Section 2.12.

2.12 counter-depth

Parameter	Type	Optional Values	Default	Description
counter-depth	`int`	`1`, `2`, `3`	`2`	Counter depth

The counter depth for images (`image`), tables (`table`), and code blocks (`raw`) within `figure` environments. Default is `2`.

Note Counter Details

When a counter has depth `1`, its numbering will be global and unaffected by chapters/sections, i.e., `1`, `2`, `3`, ...

When a counter has depth `2`, its numbering will follow level-1 headings, i.e., `1.1`, `1.2`, `2.1`, `2.2`, ... However, if the document contains no level-1 headings, Scripst will automatically treat it as depth `1`.

When a counter has depth `3`, its numbering will follow level-1 and level-2 headings, i.e., `1.1.1`, `1.1.2`, `1.2.1`, `1.2.2`, `2.1.1`, ... However:

- If the document has level-1 headings but no level-2 headings, Scripst will treat it as depth `2`.
- If the document has no level-1 headings, Scripst will treat it as depth `1`.

2.13 cb-counter-depth

Parameter	Type	Optional Values	Default	Description
cb-counter-depth	`int`	`1`, `2`, `3`	`2`	Counter depth for countable elements

The counter depth for countable elements. Default is `2`.

If you change the default depth of the `countblock` counter, you will also need to specify the changed depth when using it, or rewrap the function. See Section 3.6.4 for details.

2.14 header

Parameter	Type	Default	Description
header	`bool`	`true`	Enable header

Whether to generate headers. Default is `true`.

Note

The header displays the document title, metadata, and current chapter/section title:

- If all three exist, they will be displayed in the header in three equal parts.
- If the document has no metadata, the header will show the title on the left and chapter title on the right.
  - If the document also lacks a title, only the chapter title will appear on the

right.

- If the document has no level-1 headings, the header will show the title on the left and metadata on the right.
    - If there's no metadata, only the title will appear on the left.
- If none of these elements exist, the header will remain empty.

2.15 lang

Parameter	Type	Default Value	Description
lang	`str`	`"zh"`	Document language

The document language. The default is `"zh"`.

Accepts ISO_639-1 encoding format, such as `"zh"`, `"en"`, `"fr"`, etc.

2.16 par-indent

Parameter	Type	Default	Description
par-indent	`length`	`2em`	First-line paragraph indentation

Controls first-line paragraph indentation. Default: `2em`. Set to `0em` to disable indentation.

2.17 par-leading

Parameter	Type	Default	Description
par-leading	`length`	Language-dependent	Line spacing within paragraphs

Adjusts line spacing within paragraphs. Defaults to `1em` for Chinese documents.

Note
Default values changes according to language script type, with details shown below:

Script Category	Default
East Asian (Chinese/Japanese/Korean)	1em
Arabic Scripts (Arabic/Persian/etc.)	0.75em
Cyrillic Scripts (Russian/Bulgarian/etc.)	0.7em
South/Southeast Asian/Amharic (Thai/Hindi/etc.)	0.85em
Other Languages	0.6em

,

2.18 par-spacing

Parameter	Type	Default	Description
par-spacing	`length`	Language-dependent	Vertical spacing between paragraphs

Sets vertical spacing between paragraphs. Defaults to `1.2em` for Chinese documents.

Note
Default values changes according to language script type, with details shown below:

Script Category	Default
East Asian (Chinese/Japanese/Korean)	1.2em
Arabic Scripts (Arabic/Persian/etc.)	1.25em
Cyrillic Scripts (Russian/Bulgarian/etc.)	1.2em
South/Southeast Asian/Amharic (Thai/Hindi/etc.)	1.3em
Other Languages	1em

,

2.19 body

When using `#show: scripst.with(...)`, the `body` parameter does not need to be passed manually. Typst will automatically pass the remaining document content to the `body` parameter.

III Template Effect Display

3.1 Front Page

The beginning of the document will display the title, information, authors, time, abstract, keywords, etc., as shown at the beginning of this document.

3.2 Table of Contents

If the `contents` parameter is `true`, a table of contents will be generated, as shown in this document.

3.3 Fonts and Environments

Scripst provides some commonly used fonts and environments, such as bold, italic, headings, images, tables, lists, quotes, links, math formulas, etc.

3.3.1 Fonts

This is normal text. C'est un texte normal.

This is bold text. C'est un texte en gras.

This is italic text. C'est un texte en italique.

Install the CMU Serif font for better (LaTeX-like) display effects.

3.3.2 Environments

3.3.2.1 Headings

Level 1 headings are numbered according to the document language, including Chinese/Roman numerals/Greek letters/Kana/Numerals in Arabic/Hindi numerals, etc. Other levels use Arabic numerals.

3.3.2.2 Images

The image environment will automatically number the images, as shown below:

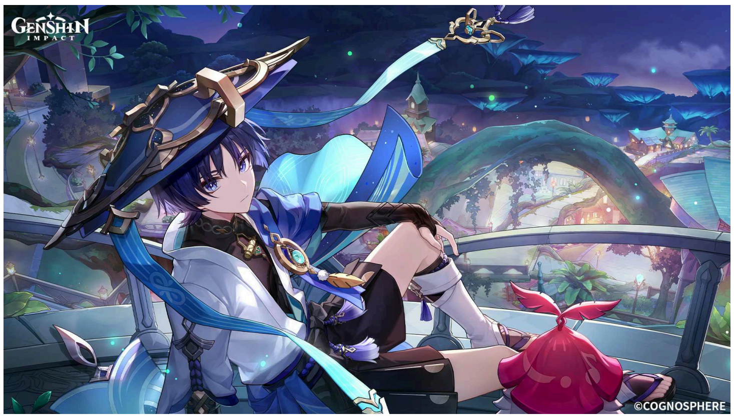

Figure 3.3.1: Little Scara

3.3.2.3 Tables

Thanks to the `tablem` package, you can write tables in Markdown style when using this template, as shown below:

```
#figure(
  three-line-table[
    | Name | Age | Gender |
    | --- | --- | --- |
    | Jane | 18 | Male |
    | Doe | 19 | Female |
  ],
  caption: [`three-line-table` table
example],
)
```

Name	Age	Gender
Jane	18	Male
Doe	19	Female

Table 3.3.1: `three-line-table` table example

```
#figure(
  tablem[
    | Name | Age | Gender |
    | --- | --- | --- |
    | Jane | 18 | Male |
    | Doe | 19 | Female |
  ],
  caption: [`tablem` table example],
)
```

Name	Age	Gender
Jane	18	Male
Doe	19	Female

Table 3.3.2: `tablem` table example

You can choose `numbering: none,` to make the table unnumbered, as shown above, the tables in the previous chapters did not enter the full text table counter.

3.3.2.4 Math Formulas

Math formulas have inline and block modes.

Inline formula: $a^2 + b^2 = c^2$.

Block formula:

$$\begin{gathered} a^2 + b^2 = c^2 \\ \frac{1}{2} + \frac{1}{3} = \frac{5}{6} \end{gathered} \tag{3.1}$$

are numbered.

Thanks to the `physica` package, Typst's math input method is greatly expanded while still retaining its simplicity:

$$\begin{aligned} \nabla \cdot \boldsymbol{E} &= \frac{\rho}{\varepsilon_0} \\ \nabla \cdot \boldsymbol{B} &= 0 \\ \nabla \times \boldsymbol{E} &= -\frac{\partial \boldsymbol{B}}{\partial t} \\ \nabla \times \boldsymbol{B} &= \mu_0 \left(\boldsymbol{J} + \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right) \end{aligned} \tag{3.2}$$

3.3.3 Lists

Typst provides a simple environment for lists, as shown:

```
- First item
- Second item
- Third item
```

- First item
- Second item
- Third item

```
+  First item
3. Second item
+  Third item
```

1. First item
3. Second item
4. Third item

First item 1
Second item 2

```
/ First item: 1
/ Second item: 2
/ Third item: 23
```

Third item 3

3.3.4 Quotes

```
#quote(attribution: "Einstein",
block: true)[
  God does not play dice with the
universe.
]
```

God does not play dice with the universe.

— *Einstein*

3.3.5 Links

```
#link("https://www.google.com/")
[Google]
```

Google

3.3.6 Hyperlinks and Citations

Use `<label>` and `@label` to achieve hyperlinks and citations.

3.4 `#newpara()` Function

By default, some modules do not automatically wrap. This is necessary, for example, if the explanation of the above math formula does not wrap.

But sometimes we need to wrap, and this is where the `#newpara()` function comes in.

Unlike the official `#parbreak()` function, the `#newpara()` function inserts a blank line between paragraphs, so it will start a new natural paragraph in any scenario.

Whenever you feel the need to wrap, you can use the `#newpara()` function.

3.5 labelset

Thanks to the `label` function in Typst, in addition to adding labels to this type, you can conveniently set styles for referenced objects using `label`.

Therefore, Scripst provides some commonly used settings, and you can set styles by simply adding a label.

```
== Schrödinger equation <hd.x>
The Schrödinger equation is as follows:
$
  i hbar dv(,t) ket(Psi(t)) = hat(H) ket(Psi(t))
$ <text.blue>
```

```
where
$
  ket(Psi(t)) = sum_n c_n ket(phi_n)
$ <eq.c>
is the wave function. From this, we can derive the time-independent
Schrödinger equation:
$
  hat(H) ket(Psi(t)) = E ket(Psi(t))
$
<text.teal>
where $E$<text.red> is #[energy]<text.lime>.
```

Schrödinger equation

The Schrödinger equation is as follows:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t} |\Psi(t)\rangle = \hat{H} |\Psi(t)\rangle \tag{3.3}$$

where

$$|\Psi(t)\rangle = \sum_n c_n |\varphi_n\rangle$$

is the wave function. From this, we can derive the time-independent Schrödinger equation:

$$\hat{H} |\Psi(t)\rangle = E |\Psi(t)\rangle \tag{3.4}$$

where E is energy.

Currently, Scripst provides the following settings:

Label	Function
`eq.c`	Removes numbering from equations in math environments
`hd.c`	Removes numbering from headings but still displays them in the table of contents
`hd.x`	Removes numbering from headings and hides them in the table of contents
`text.{color}`	Sets the text color `color in (black, gray, silver, white, navy, blue, aqua, teal, eastern, purple, fuchsia, maroon, red, orange, yellow, olive, green, lime,)`

Table 3.5.1: Label Set

⚠️
Note that the strings above have been used for styling settings. You can override their styles, but do not use these strings when working with labels and references.

3.6 countblock

Definition 3.1 countblock
Countblock is a counter module provided by Scripst for numbering countable elements in documents.

What you're seeing now is a `definition` block, which serves as an example of a counter module.

3.6.1 Default Countblocks

Scripst provides several default counters ready for use:

- Definition: `#definition`
- Theorem: `#theorem`
- Proposition: `#proposition`
- Lemma: `#lemma`
- Corollary: `#corollary`
- Remark: `#remark`
- Claim: `#claim`
- Exercise: `#exercise`
- Problem: `#problem`
- Example: `#example`
- Note: `#note`
- Caution: `#caution`

These functions share identical parameters and effects, differing only in counter names.

```
#definition(
  subname: [],
  count: true,
  lab: none,
  cb-counter-depth: 2,
)[
  ...
]
```

Parameter specifications:

Parameter	Type	Default	Description
`subname`	`array`	`[]`	Entry name
`count`	`bool`	`true`	Enable numbering
`lab`	`str`	`none`	Entry label
`cb-counter-depth`	`int`	`2`	Counter depth

Example usage:

```
#theorem(subname: [_Fermat's Last Theorem_], lab: "fermat")[

  No three $a, b, c in NN^+$ can satisfy the equation
  $
    a^n + b^n = c^n
  $
  for any integer value of $n$ greater than 2.
]
#proof[Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis
exiguitas non caperet.]
```

This creates a numbered theorem block:

Theorem 3.1 ***Fermat's Last Theorem***
No three $a, b, c \in \mathbb{N}^+$ can satisfy the equation

$$a^n + b^n = c^n \tag{3.5}$$

for any integer value of n greater than 2.

Proof. Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

■

3.6.1.1 `subname` Parameter

`subname` displays supplemental information after the counter, such as theorem names. In the example above, it shows "Fermat's Last Theorem".

3.6.1.2 `lab` Parameter

Use the `lab` parameter to create cross-references. For instance, reference the `fermat` theorem using `@fermat`:

```
Fermat never publicly proved @fermat.
```

Fermat never publicly proved Theorem 3.1.

Note that `proposition`, `lemma`, `corollary`, `remark`, and `claim` share the same counter:

Lemma 3.1
This is a lemma. Please prove it.

Proposition 3.2
This is a proposition. Please prove it.

Corollary 3.3
This is a corollary. Please prove it.

Remark 3.4

This is a remark. Please note it.

Claim 3.5
This is a claim. Please prove it.

Other counters operate independently.

3.6.1.3 `count` Parameter

Set `count: false` to disable numbering. `note` and `caution` default to `count: false`.

```
#note(count: true)[

  This is a numbered note.
]

#note[

  This is an unnumbered note.
]
```

Note 3.1
This is a numbered note.

Note
This is an unnumbered note.

3.6.1.4 `cb-counter-depth` Parameter

Detailed explanation in Section 3.6.4.

3.6.2 Global `cb` Variable

Scripst tracks all counters through the global `cb` variable, which includes default counter depth `cb-counter-depth`.

Default `cb` structure:

```
#let cb = (
  "def": ("Definition", mycolor.green, "def"),
  "thm": ("Theorem", mycolor.blue, "thm"),
  "prop": ("Proposition", mycolor.violet, "prop"),
  "lem": ("Lemma", mycolor.violet-light, "prop"),
  "cor": ("Corollary", mycolor.violet-dark, "prop"),
  "rmk": ("Remark", mycolor.violet-darker, "prop"),
  "clm": ("Claim", mycolor.violet-deep, "prop"),
  "ex": ("Exercise", mycolor.purple, "ex"),
  "prob": ("Problem", mycolor.orange, "prob"),
  "eg": ("Example", mycolor.cyan, "eg"),
  "note": ("Note", mycolor.grey, "note"),
  "cau": ("⚠️", mycolor.red, "cau"),
  "cb-counter-depth": 2,
)
```

3.6.3 Creating & Registering Countblocks

Use `add-countblock` to create counters and `reg-countblock` to register them. Add this at document start:

```
#let cb = add-countblock(cb, "test", "This is a test", teal)
#show: reg-countblock.with("test")
```

Note
This code first updates `cb`, then registers the counter.

3.6.3.1 `add-countblock` Function

Parameters for `add-countblock`:

```
#add-countblock(cb, name, info, color, counter-name: none) {return cb}
```

Parameter	Type	Default	Description
`cb`	`dict`		Counter dictionary
`name`	`str`		Counter name
`info`	`str`		Display text
`color`	`color`		Header color
`counter-name`	`str`	`none`	Counter ID

- `cb` is a dictionary with the format shown in Section 3.6.2. The function's purpose is to update `cb`, which requires explicit assignment during use.

  Note
  Since Typst's functions lack pointers or references, passed variables cannot be directly modified. We can only modify variables by explicitly returning values and passing them to subsequent functions. The author has not yet found a better approach.

- `name: (info, color, counter-name)` represents a counter's basic information. During rendering, the counter's top-left corner will display `info counter(counter-name)` (e.g., `Theorem 1.1`) as its identifier, with the color set to `color`.
- `counter-name` is the counter's identifier. If unspecified, it defaults to using `name` as the identifier.

3.6.3.2 `reg-countblock` Function

The parameters of the `reg-countblock` function are as follows:

```
#show reg-countblock.with(name, cb-counter-depth: 2)
```

Parameter specifications:

Parameter	Type	Default	Description
`counter-name`	`str`		Counter identifier
`cb-counter-depth`	`int`	2	Counter depth

- `counter-name` is the counter identifier, explicitly specified in `add-countblock` (uses `name` if unspecified). For example, the default `clm` counter uses `prop`.
- `cb-counter-depth` defines the counter depth, which can be `1`, `2`, or `3`.

After this, you can use the `countblock` function to implement the counter.

3.6.4 Counter Depth for countblock

This section details the `cb-counter-depth` parameter and its implementation, not previously mentioned.

The global variable `cb` has a default `cb-counter-depth` value of 2. Thus, default countblocks use depth 2.

> **Note**
> Directly modifying `cb-counter-depth` in the global variable will NOT affect existing counters. This is because counter creation uses the original `cb.at("cb-counter-depth")` as the default value. Updating `cb` does not retroactively change this value. You must re-register counters.

Counter logic aligns with Section 2.12.

To register a depth-3 counter:

```
#let cb = add-countblock(cb, "test1", "This is a test1", green)
#show: reg-countblock.with("test1", cb-counter-depth: 3)
```

Use `reg-default-countblock` to set default counters. For example, to **set all default counters to depth 3**:

```
#show: reg-default-countblock.with(cb-counter-depth: 3)
```

However, this alone is insufficient because the pre-packaged counters still default to depth 2. If you directly call:

```
```

```
#definition[
  This is a definition. Please understand it.
]
```

the counter depth remains 2:

Definition 3.2
This is a definition. Please understand it.

Explicitly specify depth 3:

```
#definition(cb-counter-depth: 3)[
  This is a definition. Please understand it.
]
```

Definition 3.6.3
This is a definition. Please understand it.

Alternatively, **create a custom wrapper**:

```
#let definition = definition.with(cb-counter-depth: 3)
```

Subsequent uses of `definition` will default to depth 3:

```
#definition[
  This is a definition. Please understand it.
]
```

Definition 3.6.4
This is a definition. Please understand it.

Note
In fact, the `cb-counter-depth` parameter mentioned earlier is set by calling the `reg-default-countblock` function when the document is initialized.

3.6.5 Using countblock

After defining and registering a counter, use the `countblock` function to create a block:

```
#countblock(
  name,
  cb,
  cb-counter-depth: cb.at("cb-counter-depth"), // default: 2
  subname: "",
  count: true,
  lab: none
)[
  ...
]
```

Parameter specifications:

Parameter	Type	Default	Description
`name`	`str`		Counter name
`cb`	`dict`		Counter dictionary
`cb-counter-depth`	`int`	`cb.at("cb-counter-depth")`	Counter depth
`subname`	`str`		Entry name
`count`	`bool`	`true`	Enable numbering
`lab`	`str`	`none`	Reference label

- `name`: Counter name, as specified in `add-countblock`.
- `cb`: Dictionary formatted as Section 3.6.2. Ensure it contains the latest counter by updating `cb` first.
- `cb-counter-depth`: Counter depth (`1`, `2`, or `3`).
- `subname`: Supplemental text displayed after the counter (e.g., theorem name).
- `count`: Set `false` to disable numbering.
- `lab`: Label for cross-referencing with `@lab`.

Example using the `test` counter created in Section 3.6.3:

```
#countblock("test", cb)[
  1 + 1 = 2
]
```

This is a test 3.1
$1 + 1 = 2$

Alternatively, create a wrapper function:

```
#let test = countblock.with("test", cb)
#test[
  1 + 1 = 2
]
```

This is a test 3.2
$1 + 1 = 2$

For the `test1` counter (depth 3 registered in Section 3.6.4), specify depth during use:

```
#countblock("test1", cb, cb-counter-depth: 3)[
  1 + 1 = 2
]
```

```
#let test1 = countblock.with("test1", cb, cb-counter-depth: 3)
#test1[
  1 + 1 = 2
]
```

This is a test1 3.6.1
$1 + 1 = 2$

This is a test1 3.6.2
$1 + 1 = 2$

3.6.6 Summary

Scripst provides a simple counter module system. You can use the `add-countblock` function to create counters, `reg-countblock` to register them, and `countblock` to implement them.

By default, all counters have a depth of 2. Use `reg-default-countblock` to configure default counters.

- If you want all countblocks to use depth 2, no special configuration is needed.
- If you want all countblocks to use depth 3, specify the depth during registration and usage.

Example
Example: A user wants all default countblocks to use depth 3, while making `remark` independent from the shared counter for `proposition`, `lemma`, `corollary`, and `claim`. Also create a depth-3 `algorithm` counter.

```
#show: scripst.with(
  // ...
  cb-counter-depth: 3,
)
#let cb = add-countblock(cb, "rmk", "Remark", mycolor.violet-darker)
#let cb = add-countblock(cb, "algorithm", "Algorithm", mycolor.yellow)
#show: reg-countblock.with("rmk", cb-counter-depth: 3)
#show: reg-countblock.with("algorithm", cb-counter-depth: 3)
#let definition = definition.with(cb-counter-depth: 3)
#let theorem = theorem.with(cb-counter-depth: 3)
// ...
#let remark = countblock.with("rmk", cb, cb-counter-depth: 3) // Re-
encapsulate due to counter changes
#let algorithm = countblock.with("algorithm", cb, cb-counter-depth: 3)
```

Place this code at the beginning of the document, after `#script`.

3.7 Some Other Blocks

3.7.1 Blank Block

Additionally, Scripst provides this type of block without a title, and you can use it by customizing the color.

For example:

```
#blankblock(color: color.red)[
  This is a red block.
]
```

This is a red block.

3.7.2 Proof and ∎ (Quod Erat Demonstrandum)

```
#proof[
  This is a proof.
]
```

Proof.

This is a proof.

∎

This provides a simple proof environment along with a tombstone symbol.

3.7.3 Solution

```
#solution[
  This is a solution.
]
```

Solution.

This is a solution.

This provides a simple solution environment.

3.7.4 Separator

```
#separator
```

You can use the `#separator` function to insert a separator.

IV Conclusion

The above documentation demonstrated Scripst, explained the template parameters, and showed the template effects.

I hope this document helps you better use Typst and Scripst.

You are also welcome to provide suggestions, improvements, and/or contribute code to Scripst.

Thank you for your support of Typst and Scripst!