Markdown

This section describes Julia's markdown syntax, which is enabled by the Markdown standard library. The following Markdown elements are supported:

Inline elements

Here "inline" refers to elements that can be found within blocks of text, i.e. paragraphs. These include the following elements.

Bold

Surround words with two asterisks, `**`, to display the enclosed text in boldface.

```
A paragraph containing a **bold** word.
```

Italics

Surround words with one asterisk, `*`, to display the enclosed text in italics.

```
A paragraph containing an *italicized* word.
```

Literals

Surround text that should be displayed exactly as written with single backticks, `` ` `` .

```
A paragraph containing a `literal` word.
```

Literals should be used when writing text that refers to names of variables, functions, or other parts of a Julia program.

Tip

To include a backtick character within literal text use three backticks rather than one to enclose the text.

```
A paragraph containing ``` `backtick` characters ```.
```

By extension any odd number of backticks may be used to enclose a lesser number of backticks.

$\LaTeX$

Surround text that should be displayed as mathematics using $\LaTeX$ syntax with double backticks, ` `` ` .

```
A paragraph containing some ``\LaTeX`` markup.
```

Tip

As with literals in the previous section, if literal backticks need to be written within double backticks use an even number greater than two. Note that if a single literal backtick needs to be included within $\LaTeX$ markup then two enclosing backticks is sufficient.

Note

The `\` character should be escaped appropriately if the text is embedded in a Julia source code, for example, ```"``\\LaTeX`` syntax in a docstring."```, since it is interpreted as a string literal. Alternatively, in order to avoid escaping, it is possible to use the `raw` string macro together with the `@doc` macro:

```
@doc raw"``\LaTeX`` syntax in a docstring." functionname
```

Links

Links to either external or internal targets can be written using the following syntax, where the text enclosed in square brackets, `[ ]`, is the name of the link and the text enclosed in parentheses, `( )`, is the URL.

```
A paragraph containing a link to [Julia](http://www.julialang.org).
```

It's also possible to add cross-references to other documented functions/methods/variables within the Julia documentation itself. For example:

```
"""
    tryparse(type, str; base)

Like [`parse`](@ref), but returns either a value of the requested type,
or [`nothing`](@ref) if the string does not contain a valid number.
"""
```

This will create a link in the generated docs to the `parse` documentation (which has more information about what this function actually does), and to the `nothing` documentation. It's good to include cross references to mutating/non-mutating versions of a function, or to highlight a difference between two similar-seeming functions.

Note

The above cross referencing is *not* a Markdown feature, and relies on Documenter.jl, which is used to build base Julia's documentation.

Footnote references

Named and numbered footnote references can be written using the following syntax. A footnote name must be a single alphanumeric word containing no punctuation.

```
A paragraph containing a numbered footnote [^1] and a named one [^named].
```

Note

The text associated with a footnote can be written anywhere within the same page as the footnote reference. The syntax used to define the footnote text is discussed in the Footnotes section below.

Toplevel elements

The following elements can be written either at the "toplevel" of a document or within another "toplevel" element.

Paragraphs

A paragraph is a block of plain text, possibly containing any number of inline elements defined in the Inline elements section above, with one or more blank lines above and below it.

```
This is a paragraph.

And this is *another* paragraph containing some emphasized text.
A new line, but still part of the same paragraph.
```

Headers

A document can be split up into different sections using headers. Headers use the following syntax:

```
# Level One
## Level Two
### Level Three
#### Level Four
##### Level Five
###### Level Six
```

A header line can contain any inline syntax in the same way as a paragraph can.

Tip

Try to avoid using too many levels of header within a single document. A heavily nested document may be indicative of a need to restructure it or split it into several pages covering separate topics.

Code blocks

Source code can be displayed as a literal block using an indent of four spaces as shown in the following example.

````
This is a paragraph.

    function func(x)
        # ...
    end

Another paragraph.
````

Additionally, code blocks can be enclosed using triple backticks with an optional "language" to specify how a block of code should be highlighted.

~~~
A code block without a "language":

```
function func(x)
    # ...
end
```

and another one with the "language" specified as `julia`:

```julia
function func(x)
    # ...
end
```
~~~

Note

"Fenced" code blocks, as shown in the last example, should be preferred over indented code blocks since there is no way to specify what language an indented code block is written in.

Block quotes

Text from external sources, such as quotations from books or websites, can be quoted using > characters prepended to each line of the quote as follows.

````
Here's a quote:

> Julia is a high-level, high-performance dynamic programming language for
````

```
> technical computing, with syntax that is familiar to users of other
> technical computing environments.
```

Note that a single space must appear after the `>` character on each line. Quoted blocks may themselves contain other toplevel or inline elements.

Images

The syntax for images is similar to the link syntax mentioned above. Prepending a `!` character to a link will display an image from the specified URL rather than a link to it.

```
![alternative text](link/to/image.png)
```

Lists

Unordered lists can be written by prepending each item in a list with either `*`, `+`, or `-`.

```
A list of items:

  * item one
  * item two
  * item three
```

Note the two spaces before each `*` and the single space after each one.

Lists can contain other nested toplevel elements such as lists, code blocks, or quoteblocks. A blank line should be left between each list item when including any toplevel elements within a list.

```
Another list:

  * item one

  * item two

    ```
    f(x) = x
    ```

  * And a sublist:

      + sub-item one
```

```
      + sub-item two
```

Note

The contents of each item in the list must line up with the first line of the item. In the above example the fenced code block must be indented by four spaces to align with the `i` in `item two`.

Ordered lists are written by replacing the "bullet" character, either `*`, `+`, or `-`, with a positive integer followed by either `.` or `)`.

```
Two ordered lists:

 1. item one
 2. item two
 3. item three

 5) item five
 6) item six
 7) item seven
```

An ordered list may start from a number other than one, as in the second list of the above example, where it is numbered from five. As with unordered lists, ordered lists can contain nested toplevel elements.

Display equations

Large $\LaTeX$ equations that do not fit inline within a paragraph may be written as display equations using a fenced code block with the "language" `math` as in the example below.

~~~
```math
f(a) = \frac{1}{2\pi}\int_{0}^{2\pi} (\alpha+R\cos(\theta))d\theta
```
~~~

Footnotes

This syntax is paired with the inline syntax for Footnote references. Make sure to read that section as well.

Footnote text is defined using the following syntax, which is similar to footnote reference syntax, aside from the `:` character that is appended to the footnote label.

~~~
[^1]: Numbered footnote text.

[^note]:

    Named footnote text containing several toplevel elements.

      * item one
      * item two
      * item three

    ```julia
    function func(x)
        # ...
    end
    ```
~~~

Note

No checks are done during parsing to make sure that all footnote references have matching footnotes.

Horizontal rules

The equivalent of an `<hr>` HTML tag can be achieved using three hyphens (`---`). For example:

```
Text above the line.

---

And text below the line.
```

Tables

Basic tables can be written using the syntax described below. Note that markdown tables have limited features and cannot contain nested toplevel elements unlike other elements discussed above – only inline elements are allowed. Tables must always contain a header row with column names. Cells cannot span multiple rows or columns of the table.

```
| Column One | Column Two | Column Three |
|:---------- | ---------- |:------------:|
| Row `1`    | Column `2` |              |
| *Row* 2    | **Row** 2  | Column ``3`` |
```

Note

As illustrated in the above example each column of `|` characters must be aligned vertically.

A `:` character on either end of a column's header separator (the row containing `-` characters) specifies whether the row is left-aligned, right-aligned, or (when `:` appears on both ends) center-aligned. Providing no `:` characters will default to right-aligning the column.

Admonitions

Specially formatted blocks, known as admonitions, can be used to highlight particular remarks. They can be defined using the following `!!!` syntax:

```
!!! note

    This is the content of the note.

!!! warning "Beware!"

    And this is another one.

    This warning admonition has a custom title: `"Beware!"`.
```

The type of the admonition can be any word made up of only lowercase Latin characters (a-z), but some types produce special styling, namely (in order of decreasing severity): `danger`, `warning`, `info`, `note`, and `tip`.

A custom title for the box can be provided as a string (in double quotes) after the admonition type. For that standard types (`danger`, `warning`... etc_, if no title text is specified after the admonition type, then the type title used will be the type of the block. E.g. `"Note"` in the case of the `note` admonition.

If you would like to define your own block, for example a `terminology` block used like so:

```
!!! terminology "julia vs Julia"
    Strictly speaking, Julia refers to the language,
    and julia the standard implementation.
```

Admonitions, like most other toplevel elements, can contain other toplevel elements.

Markdown Syntax Extensions

Julia's markdown supports interpolation in a very similar way to basic string literals, with the difference that it will store the object itself in the Markdown tree (as opposed to converting it to a string). When the Markdown content is rendered the usual `show` methods will be called, and these can be overridden as usual. This design allows the Markdown to be extended with arbitrarily complex features (such as references) without cluttering the basic syntax.

In principle, the Markdown parser itself can also be arbitrarily extended by packages, or an entirely custom flavour of Markdown can be used, but this should generally be unnecessary.

« Logging Memory-mapped I/O »