

# bytefield manual



## Abstract

**bytefield** is a package for creating *network protocol headers*, *memory maps*, *register definitions* and more in typst.

Version: 0.0.4

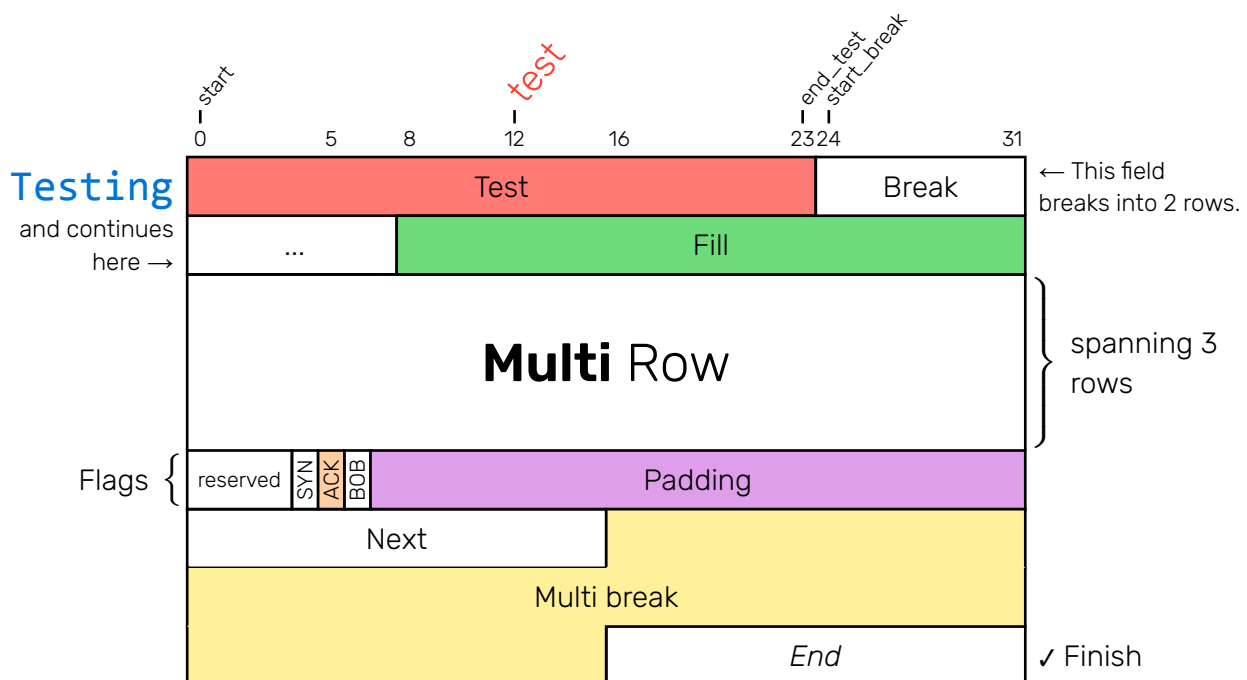
Authors: [jomaway](#) + community contributions.

License: MIT

## Contents

1. Example .....	1
2. Usage .....	2
3. Features .....	2
3.1. Data fields .....	2
3.2. Annotations .....	2
3.3. Headers [WIP] .....	3
4. Use cases .....	4
4.1. Protocol Headers .....	4
4.2. Memory Maps .....	4
4.3. Register Definitions .....	5
5. Reference .....	6
5.1. User API .....	6

## 1. Example



Bytefield 1: Random example of a colored bytefield.

Source and more examples can be found [here](#).

## 2. Usage

Import the package from the official package manager

```
#import "@preview/bytefield:0.0.4": *
```

or download the package and put it inside the folder for local packages.

## 3. Features

### 3.1. Data fields

By default a bytefield shows 32 bits per row. This can be changed by using the `bpr` argument. For example `bpr:16` changes the size to 16 bits per row.

You can add fields of different size to the bytefield by using one of the following field functions.

`bit`, `bits`, `byte`, `bytes`, `flag`

- Fields can be colored with a `fill` argument.

Multirow and breaking fields are supported. This means if a field does not fit into one row it will break automatically into the next one.

### 3.2. Annotations

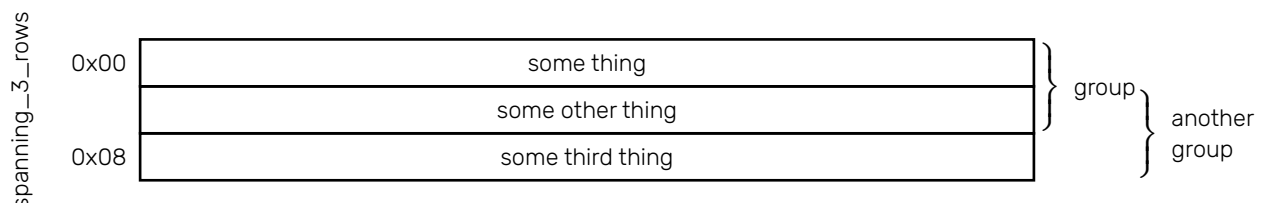
Define annotations in columns left or right of the bitfields current row with the helpers `note` and `group`.

The needed number of columns is determined automatically, but can be forced with the `pre` and `post` arguments.

The helper `note` takes the side it should appear on as first argument, an optional `rowspan` for the number of rows it should span and an optional `level` for the nesting level.

The helper `group` takes the side it should appear on as first argument, as second argument `rowspan` for the number of rows it should span and an optional `level` for the nesting level.

```
1 #bytefield(  
2   pre: (1cm,auto),  
3   post: (auto,1.8cm),  
4   note(left, rowspan:3, level:1)[  
5     #align(center,rotate(270deg)[spanning_3_rows])  
6   ],  
7   note(left)[0x00],  
8   group(right,2)[group],  
9   bytes(4)[some thing],  
10  
11 // note(left)[0x04],  
12 group(right,2,level:1)[another group],  
13 bytes(4)[some other thing],  
14 note(left)[0x08],  
15 bytes(4)[some third thing],  
16 )
```



### 3.3. Headers [WIP]

▲ The new bitheader api is still a work in progress and might change a bit in the next version.

The current API is described here:

The `bitheader` function defines which bit-numbers and text-labels are shown as a header. Currently **only the first** `bitheader` per `bytefield` is processed, all others will be ignored.

There are some *named* arguments and an arbitrary amount of *positional* arguments which you can pass to a header.

Show or hide numbers

- `numbers: none` hide all numbers
- `numbers: auto` show all specified numbers *default*

Some common use cases can be set by adding a `string` value. *positional*

- `"all"` will show numbers for all bits.
- `"bytes"` will show every multiple of 8 and the last bit.
- `"bounds"` will show begin and end of each field in the first row.
- `"offsets"` will show begin of each field in the first row.

Showing a number. *positional*

- Just add an `int` value with the number you would like to show.

Showing a text label for a number *positional*

- Add a content field after the int value which the label belongs to.

#### i Info

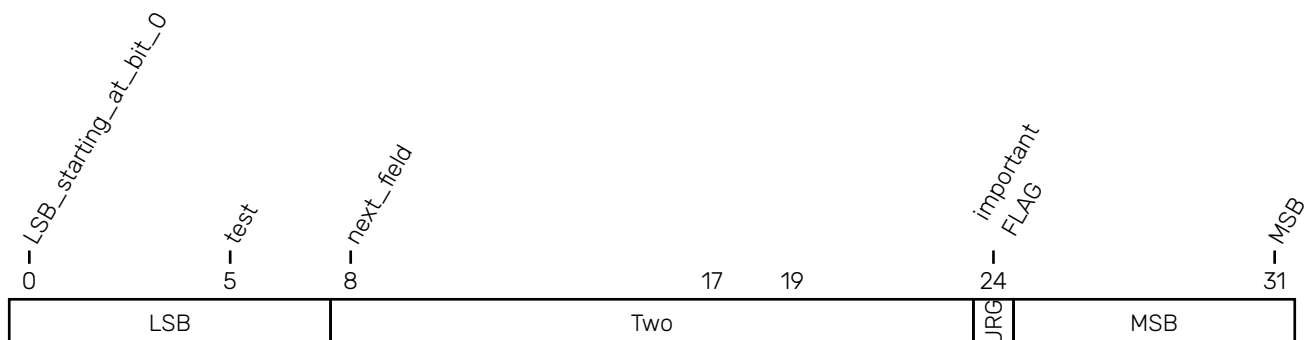
Set the order of the bits with the `msb` argument directly on the `bytefield`.

- `msb:right` displays the numbers from (left) 0 – to – msb (right) *default*
- `msb:left` displays the numbers from (left) msb – to – 0 (right)

### Numbers and Labels example

You can also show labels and indexes by specifying a `content` after an `number` (`int`).

```
2 bitheader(  
3     0,[LSB_starting_at_bit_0],  
4     5, [test],  
5     8, [next_field],  
6     24, [important FLAG],  
7     31, [MSB],  
8     17,19,  
9 )
```

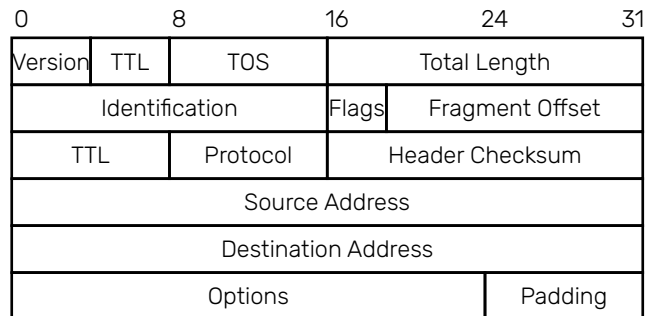


## 4. Use cases

### 4.1. Protocol Headers

Generate protocol headers like the one from the **ipv4** protocol.

```
1 #bytefield(  
2   bitheader("bytes"),  
3   bits(4)[Version], bits(4)[TTL], bytes(1)[TOS],  
4   bytes(2)[Total Length],  
5   bytes(2)[Identification], bits(3)[Flags],  
6   bits(13)[Fragment Offset],  
7   bytes(1)[TTL], bytes(1)[Protocol], bytes(2)  
8   [Header Checksum],  
9   bytes(4)[Source Address],  
10  bytes(4)[Destination Address],  
11  bytes(3)[Options], bytes(1)[Padding]  
12 )
```

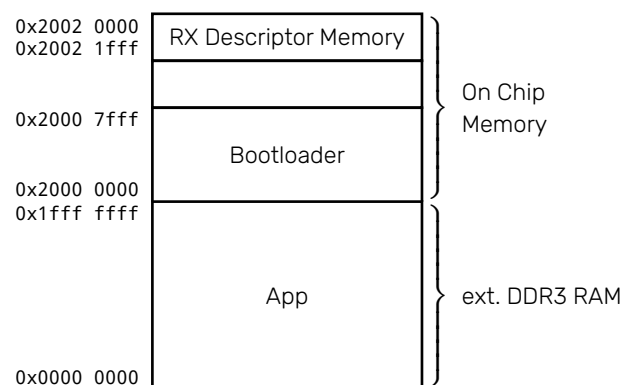


Bytefield 1: Common IPv4 Header.

### 4.2. Memory Maps

Generate memory maps. Currently possible with a little workaround using bits. Better support is on the roadmap.

```
1 #bytefield(  
2   bpr: 1,  
3   group(right,4)[On Chip Memory],  
4   section("0x2002 0000", "0x2002 1fff"),  
5   bit[RX Descriptor Memory],  
6   bit[],  
7   section("0x2000 7fff", "0x2000 0000", span: 2),  
8   bits(2)[Bootloader],  
9   group(right,4)[ext. DDR3 RAM],  
10  section("0x1fff ffff", "0x0000 0000", span: 4),  
11  bits(4)[App],  
12 )
```



Bytefield 2: A memory map example.

### 4.3. Register Definitions

Creating register definition like [Bytfield 3](#) is currently possible by using two `bytefields` and tweaking the header accordingly.

```
1 #let reg_field(body, size: 1, rw: "rw") = {
2   bits(size, table(columns: 1fr, rows: (2fr, auto), body, rw))
3 }
4
5 #let reserved(size) = bits(size)[Reserved]
6
7 #set text(8pt)
8 #bytefield(
9   bpr: 16,
10  msb: left,
11  bitheader(range: (16,32), ..range(16,32), msb: left),
12  reserved(4),
13  reg_field(rw: "r")[PLL I2S RDY],
14  reg_field[PLL I2S ON],
15  reg_field(rw: "r")[PLL RDY],
16  reg_field[PLL ON],
17  reserved(4),
18  reg_field[CSS ON],
19  reg_field[HSE BYP],
20  reg_field(rw: "r")[HSE RDY],
21  reg_field[HSE ON],
22 )
23 #bytefield(
24   bpr: 16,
25   msb: left,
26   bitheader("all", msb: left),
27   reg_field(size:8, rw: "r")[HSICAL[7:0]],
28   reg_field(size:5)[HSITRIM[4:0]],
29   reg_field[Res.],
30   reg_field(rw: "r")[HSI RDY],
31   reg_field[HSION],
32 )
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLL I2S RDY	PLL I2S ON	PLL RDY	PLL ON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
				r	rw	r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]				Res.	HSI RDY	HSION	
r								rw				rw	r	rw	

Bytfield 3: Register Definition from the STM32 manual recreated with `bytefield`

## 5. Reference

### 5.1. User API

- `bytefield()`
- `_field()`
- `bit()`
- `bits()`
- `byte()`
- `bytes()`
- `flag()`
- `padding()`
- `note()`
- `group()`
- `section()`
- `bitheader()`

### bytefield

Create a new bytefield.

**Example:** See [Section 4](#).

#### Parameters

```
bytefield(  
    bpr: int,  
    msb,  
    pre: auto int relative fraction array,  
    post: auto int relative fraction array,  
    ..fields: bf-field  
) -> bytefield
```

**bpr**    int

Number of bits which are shown per row.

Default: 32

**pre**    auto or int or relative or fraction or array

This is specifies the columns for annotations on the **left** side of the bytefield

Default: auto

**post**    auto or int or relative or fraction or array

This is specifies the columns for annotations on the **right** side of the bytefield

Default: auto

**..fields**    bf-field

arbitrary number of data fields, annotations and headers which build the bytefield.

## **\_field**

Base for `bit`, `bits`, `byte`, `bytes`, `flag` field functions

⚠ This is just a base function which is used by the other functions and should **not** be called directly.

### **Parameters**

```
_field(  
    size: int,  
    fill: color,  
    body: content  
)
```

**size** `int`

The size of the field in bits.

**fill** `color`

The background color for the field.

Default: `none`

**body** `content`

The label which is displayed inside the field.

## **bit**

Add a field of the size of **one bit** to the bytearray

Basically just a wrapper for `_field()`

### **Parameters**

```
bit(..args: arguments)
```

**..args** `arguments`

All arguments which are accepted by `_field`

## **bits**

Add a field of a given size of bits to the bytearray

Basically just a wrapper for `_field()`

## Parameters

```
bits(  
    len: int,  
    ..args: arguments  
)
```

**len**    int

Size of the field in bits

**..args**    arguments

All arguments which are accepted by `_field`

## byte

Add a field of the size of one byte to the bytearray

Basically just a wrapper for `_field()`

## Parameters

```
byte(..args: arguments)
```

**..args**    arguments

All arguments which are accepted by `_field`

## bytes

Add a field of the size of multiple bytes to the bytearray

Basically just a wrapper for `_field()`

## Parameters

```
bytes(  
    len: int,  
    ..args: arguments  
)
```

**len**    int

Size of the field in bytes

**..args**    arguments

All arguments which are accepted by `_field`



## flag

Add a flag to the bytefield.

Basically just a wrapper for `_field()`

### Parameters

```
flag(  
  text: content,  
  ..args: arguments  
)
```

**text** `content`

The label of the flag which is rotated by `270deg`

**..args** `arguments`

All arguments which are accepted by `_field`

## padding

Add a field which extends to the end of the row

⚠ This can cause problems with `msb:left`

### Parameters

```
padding(..args: arguments)
```

**..args** `arguments`

All arguments which are accepted by `_field`

## note

Create a annotation

The note is always shown in the same row as the next data field which is specified.

### Parameters

```
note(  
  side: left right,  
  rowspan: int,  
  level: int,  
  inset: length,  
  bracket: bool,  
  content: content  
)
```

**side** `left` or `right`

Where the annotation should be displayed

**rowspan** `int`

Defines if the cell is spanned over multiple rows.

Default: `1`

**level** `int`

Defines the nesting level of the note.

Default: `0`

**inset** `length`

Inset of the the annotation cell.

Default: `5pt`

**bracket** `bool`

Defines if a bracket will be shown for this note.

Default: `false`

**content** `content`

The content of the note.

## group

Shows a note with a bracket and spans over multiple rows.

Basically just a shortcut for the `note` field with the argument `bracket:true` by default.

### Parameters

```
group(  
  side,  
  rowspan,  
  level,  
  bracket,  
  content  
)
```

## section

Shows a special note with a start\_addr (top aligned) and end\_addr (bottom aligned) on the left of the associated row.

**⚠ experimental:** This will probably change in a future version.

### Parameters

```
section(  
    start_addr: string content,  
    end_addr: string content,  
    span  
)
```

**start\_addr**    string or content

The start address will be top aligned

**end\_addr**    string or content

The end address will be bottom aligned

## bitheader

Config the header on top of the bytearray

By default no header is shown.

### Parameters

```
bitheader(  
    msb: left right,  
    range: array,  
    autofill: string,  
    numbers: auto none,  
    ..args: int content  
)
```

**msb**    left or right

This sets the bit order

Default: right

**range**    array

Specify the range of number which are displayed on the header. Format: (start, end)

Default: (auto, auto)

**autofill** `string`

Specify one of the following options and let bytefield calculate the numbers for you.

- `"bytes"` shows each multiple of 8 and the last number of the row.
- `"all"` shows all numbers.
- `"bounds"` shows a number for every start and end bit of a field.
- `"offsets"` shows a number for every start bit of a field.

Default: `auto`

**numbers** `auto` or `none`

if none is specified no numbers will be shown on the header. This is useful to show only labels.

Default: `auto`

**..args** `int` or `content`

The numbers and labels which should be shown on the header. The number will only be shown if it is inside the range. If a `content` value follows a `int` value it will be interpreted as label for this number. For more information see the *manual*.