

Overview

Erlang can match binaries just as any list of things. `<<E1, E2, E3>> = Bin` divides the binary `Bin` into three elements of type `integer` of one byte each. This means that `Bin` has to be 24 bits long, or we get a `badmatch`. You can also make partial matches, in a `[Head | Tail]` fashion, by putting `/bitstring` on the last element, like so: `<<E1, E2, E3/bitstring>> = Bin`. This is a **type modifier** and tells Erlang that there are two 8-bit elements, in `E1` and `E2` respectively, and then an undetermined amount of bits stored in `E3`.

Type Modifiers

Type	Size in bits	Remarks
<code>integer</code>	As many as it takes	Default size is 8 bits
<code>float</code>	64 32 16	Need to specify length if other than default: <code><<A :16 /float>></code>
<code>binary bytes</code>	8 per chunk	Anything matched must be of size evenly divisible by 8 (this is default)
<code>bitstring bits</code>	1 per chunk	Will always match, use as Tail for a list
<code>utf8 utf16 utf32</code>	8-32, 16-32, and 32	<code><<"a bc"/utf8>></code> is the same as <code><<\$ a/utf8, \$b/utf8, \$c/utf8>></code>
<code>signed unsigned</code>	N/A	Default is unsigned
<code>big little native</code>	N/A	Endianness - native is resolved at load time to whatever the CPU uses
<code>unit:IntLiteral</code>	N/A	Define a custom unit of length 1..256

Examples

Expression	Result
<code><<97, 98, 99>></code>	<code><<"a bc">></code> (turn off With shell: <code>string s(f a lse))</code>
<code><<A :2/ unit:6, B:1/unit: 4>> = <<7, 42>></code>	<code>A = 114 B = 10</code>
<code><<A :16 /float>> = <<1, 17>></code>	<code>1.6272 068 023 681 64e-5</code>
<code><<A /signed>> = <<2 55>></code>	<code>-1</code>
<code><<A :16 /big>> = <<255, 0>></code>	<code>65280</code>
<code><<A :16 /little>> = <<255, 0>></code>	<code>255</code>
<code><<"p öpc örn " /utf8>></code>	How Erlang handles unicode

When constructing a binary, if the size of an integer `N` is too large to fit inside the given segment, the most significant bits are silently discarded and only the `N` least significant bits kept.

Segments

Binary Comprehension Example

Just like with lists, there is a notation for binary comprehension. Below is an example of how to convert a 32 bit integer into a hex representation:

```
int_as_hex(Int) ->
    Int AsBin = <<Int: 32> >,
    " 0x" ++ lists:flatten([byte_to_hex(
        > <= IntAsBin)).
    byte_to_hex( <<Nibble1:4, Nibble 2:4>>) ->
        [integer_to_list(Nibble1, 16), integer_to_list(Nibble2, 16)]).
```

You can mix list- and binary comprehension: if the generator is a list, use `<-`, if it's a binary, use `<=`. If you want the result to be a binary, use `<<>>`, if you want a list, use `[]` around the expression.

Troubleshooting

Use the Erlang shell to trial and error you way to a correct expression. / understanding why your binaries are badmatching is `bit_size`:

```
bit_size( <<1 /integer> >). => 8
bit_size( <<< <1 /string>> >). => 2
bit_size( <<1.0/ float> >). => 64
bit_size( <<< <1 /big>> >). => 16
```

A related one is `byte_size`:

```
MinByteSizeOfEncodedNumber = byte_size(binary_encode(
    EncodedNumber)).
```

Each segment in a binary has the following general syntax: `Value:Size/TypeSpecifierList`. The `Size` and `TypeSpecifier` can be omitted.

`Value` is either a literal or a variable, `Size` is multiplied by the unit in `TypeSpecifierList`, and can be any expression that evaluates to an integer¹. Think of 'Size' as the number of items of the type in the 'TypeSpecifierList'

Contrived example: `<<X :4/ little -signed-integer -unit: 8>>` has a total size of $4*8 = 32$ bits, and it contains a signed integer in little endian byte order.

¹ Mostly true, see Bit Syntax Expressions in Erlang documentation for complete picture.



By **Magnus (Fylke)**
cheatography.com/fylke/

Published 25th September, 2019.
Last updated 8th March, 2023.
Page 2 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>