

U2LAL86AY : ericgj: couple of days ago i asked how to do that multi dimensional chat layout - there was that problem of calculating the height - of boxes containing text - and i wasn't able to do it in elm. I followed your advice on `ports` - and not `ready/ready states` - and worked out pretty awesome. Just wanted to let you know that your advice was very helpful :smile: I will come back with a gif later today.

U0JFGGZS6 : glad it helped, but I think it was at least partially someone else's idea :wink: anyway talking it out often helps regardless!

U625M23DE : i'm working on a single-page app which produces SVG as an output. i'm contemplating how to get compile-time guarantees about reusable chunks of my views, specifically that they have appropriate aspect ratios.

U625M23DE : so with that in mind, i defined a type like this:

U625M23DE : ```type alias GroupBox ratio msg = { scale: Int, contents: List (Svg msg) }```

U625M23DE : and use it with a bunch of auxiliary types, like this:

U625M23DE : ```type Ratio11 = Ratio11type Ratio41 = Ratio41

button41 : String -> GroupBox Ratio41 msg

button41 label = ...```

U625M23DE : but this induces a large amount of repeating myself, in that `Ratio41` isn't actually tied to the ratio 4:1 anywhere. i considered just defining the `Ratio` type with parameters so that i could extract those parameters, but this would lose the compile-time guarantee that whenever i render a thing the parent and child must agree about the aspect ratio that should be used.

U625M23DE : so, given that elm doesn't have typeclasses, is there a tidier way to keep the compile-time safety in place?

U1CE9DL9H : <@U625M23DE> it looks like phantom types might be helpful here

U1CE9DL9H : ```type Ratio41 = Ratio41

type alias Element ratio msg = Svg msg

compose : Element ratio msg -> Element ratio msg -> Element ratio msg

```

U1CE9DL9H : depending on exactly what you want you'll probably have to extend/change that a bit.

U1CE9DL9H : the trick here is that `Element` takes an extra type variable `ratio` that isn't actually used

U1CE9DL9H : ah i see now that is what you use

U625M23DE : that's similar to how i'm doing it now: i have a function which takes a ratio-tagged element and a ratio-tagged parent request for rendering, and renders it. but the problem is that the actual ratio values aren't retrievable:

U625M23DE : ```render : GroupBox ratio msg -> ParentContext ratio -> Int -> Int -> Svg msgrender box pc actualW actualH = ...

render41 : GroupBox Ratio41 msg -> ParentContext Ratio41 -> Svg msg

render41 = render 4 1```

U1CE9DL9H : but, that is how it will always work, at some point the type has to be converted into a value, and you can't check that for correctness (unless dependent types)

U1CE9DL9H : actually, ```render : GroupBox ratio msg -> ParentContext ratio -> (ratio -> (Int, Int)) -> Int -> Int -> Svg msg``` would make it slightly nicer

U625M23DE : if i could write a function like this, i could abstract a lot of the boilerplate away:```extract : ratio -> (Int, Int)

extract r =

case r of

Ratio41 -> (4, 1)

\_ -> Debug.crash("Got request for ratio parameters of non-ratio object " ++ (toString r))```

U1CE9DL9H : are you familiar with the "scrap your typeclasses" approach?

U625M23DE : vaguely, but i don't see how it applies to this situation.

U625M23DE : (not saying it doesn't, just haven't played with that enough to see the connection)

U1CE9DL9H : just to be clear: yes, in comparison to haskell you'll have to write some more stuff.

U1CE9DL9H : so this is a little nicer ```

type alias Ratio r = (Int, Int)

render : Ratio ratio -&gt; GroupBox ratio msg -&gt; ParentContext ratio -&gt; Int -&gt; Int -&gt; Svg msg  
render = ...

ratio41 : Ratio Ratio41  
ratio41 = (4,1)

...

but, how many ratios do you want to support, surely there is a limited number that makes sense?

U625M23DE : oh, that's interesting: extend the phantom-ness of the type to the Ratio type itself, and then you can indeed write the ratio-extracting function i mentioned.