U5QJW0DDE : ah so there is a not function specifically written to hand Bools
U0FP80EKB : <@U1ANV9FML> that would be an interesting graph.
U41NK9BM4 : I believe it exists. I recall I've seen something like that
U5QJW0DDE : if i have type Msg = Bool and I want to match on msg, are the cases branches not True -> and False -> ? the compiler is complaining
U5QJW0DDE : oh, Msg needs to be a union
U0FP80EKB : <@U5QJW0DDE> you would need `type alias Msg = Bool` if you want that
U0FP80EKB : `type` defines a new type, whereas `type alias` let's you set a new name (Msg) for existing type (Bool)
U0FP80EKB : If you used `type alias Msg = Bool` then the compiler won't complain, and you should be able to go fine
U5QJW0DDE : from what I can understand about the Architecture, all messages that could change application state (for purposes of view rendering) are grouped together into a single type, right? If so, that must mean that for some large applications, you'd have something like a union with maybe dozens or hundreds of options
U5QJW0DDE : if essentially every single action a user can take on a page is an option of the same type
U604S603Y : I'm dealing with a challenge concerning nested models again: I have a `Record a`, containing a `Record b`, containing a `Maybe c`, containing a `Maybe Float` of which I want the string representation of the inner value in the Just case. When I'm using ugly nested `case ... of ...` to match the `Maybe`s I get the `Float` as `String` in the end just fine, let's say "42".
But when making it look nicer using `Maybe.map` I end up with "Just 42" in the end.
And using `|> Maybe.withDefault ""` does not work, because Float and String don't match. And `toString`ing the Float beforehand gets me "Just 42" again.
The complete helper methond in the let block is:
```getErgebnisZahl schnellcheckData =
        case schnellcheckData.rechenergebnis of
            Just re -&gt;
                case re.ergebnis of
                    Just ergebnis -&gt;
                        ergebnis |&gt; toString

                    Nothing -&gt;
                        ""

            Nothing -&gt;
                ""```

U0FP80EKB : Yeah
U0FP80EKB : <@U5QJW0DDE> so, you have one top-level `type Msg = ....` that contains all the messages that can be triggered.
U0FP80EKB : <@U5QJW0DDE> Here's a sample of my top-level `Msg` on one of my apps (this is an editor for customizing our embed modules) ```
type Msg
    = UpdateEmbed FieldHelpers.FieldInfo
    | Save
    | SaveResult (Api.ApiResult Embed)
    | EmbedMsg Embed.Msg
    | ImageSelected ImagePicker.ImageControl FieldHelpers.FieldInfo
    | ImageControlClicked ImagePicker.ImageControl
    | ImageStateChange { id : String, state : String, data : String, bytes : Int, totalBytes : Int }
    | ToggleEmbedCodeVisibility
    | IgnoreEmbedMsg Embed.Msg
```

U5QJW0DDE : <@U0FP80EKB> would be nice if elm-format allowed you to put empty lines between parts of that long list, for annotating different categories of related messages
U0FP80EKB : As you start noticing groupinng of messages (such as ones that update the data for our `Embed`), you can pull them into a new one and collapse them, such as `EmbedMsg Embed.Msg`
U0FP80EKB : If you start wanting to annotate, you might be at a point where you want to start grouping them.
U5QJW0DDE : oh i see. so EmbedMsg is itself a union
U0FP80EKB : Yeah, notice `FieldHelpers.FieldInfo`, as well. This is how we collapsed editing form fields
U0FP80EKB : We have `FieldInfo` further collapse```
type FieldInfo

```
    = ContentField ContentFieldInfo
    | AppearanceField AppearanceFieldInfo
    | GeneralField GeneralFieldInfo
```

U5QJW0DDE : that's' nice
U0FP80EKB : So, over time, there are techniques to keep your top-level `Msg` from being too cluttered. At some point, I always end up collapsing my form field update messages.
U5QJW0DDE : i dig it