U601ELFEG : but if it is more complex (a dialog with lots of controls and state) - then I don't want to put that at "top"... but nor does it really seem to belong to sub-model...
U601ELFEG : Wondering how people approach this kind of structure
U601ELFEG : Do I just need to make the dialog module's `update` function `: Msg -> Model -> (Model, Cmd Msg, Maybe FinalThingToDo)` ??
U601ELFEG : er - `Result` was a bad choice, edited...
U3SJEDR96 : yeah. Or a `{ model : Model, effects : Cmd Msg, someOtherTing : Foo }`
U0J1M0F32 : I've found the sub module structure to not scale very well, leading to some cumbersome update functions. I try to not nest the Model/Update/View triplet more than once or maybe twice, if I can work around it.
U3SJEDR96 : that `update` is just a function that doesn't need to conform to any particular convention or signature
U601ELFEG : I have a few `Task.process SomeExportedMsgType <| Task.succeed resultOfTheDialog` --- but that has a pretty bad code smell
U3SJEDR96 : But yeah, keeping things flat is generally nice. Passing in functions for giving reusable views a proper way to interact with your code is also a pretty neat "pattern"
U3SJEDR96 : and yeah, using tasks to force a round-trip through the runtime is a pretty smelly thing indeed; generally splitting of whatever functionality you need and just calling that function tends to work out much more nicely :slightly_smiling_face:
U64FYS317 : I'm looking for a syntax to get around this problem: ```
update : Msg -> Model
update msg =
    case msg of
        FirstType -> msg
        -- throws: subUpdate is expecting `UnionType`, but it is `Msg`
        UnionType -> subUpdate msg

```

U64FYS317 : where UnionType would be something like `type UnionType = A String | B | C`
U2LAL86AY : <@U64FYS317> is very strange to name your constructors UnionType - because a union Type by definition contains one or more constructors. But anyway - that will not work because you pass in `msg` to `subUpdate` and msg is of type `Msg` not of type `UnionType`
U2LAL86AY : ah i see.
U64FYS317 : <@U2LAL86AY> It's just been adapted to a short example. Is there a way to hint at the compiler type that it IS necessarily a UnionType, considering it's already in that branch of the case statement?
U2LAL86AY : yes but you need to do something like this
U2LAL86AY : ```type Msg =
  Click
  | OnEnter
  | MsgFor_DatePicker  DatePickerMsg  -- or whatever you have
```
then your datepicker is that union type you want.
```
type DatePickerMsg =
     Select
     | Hover
     | DoStuff
```

U2LAL86AY : and in your main update case statement you hace
U2LAL86AY : ```case msg of
  Click -> -- do stuff.
  OnEnter -> --do stuff
  MsgFor_DatePicker datePickerMsg ->
         subUpdate datePickerMsg
```

U2LAL86AY : so for your case - you can leave your `type UnionType = A String | B | C` alone -> he will be like the `DatePickerMsg `
U2LAL86AY : but you need to build another tag for it
U64FYS317 : thank you.
```

U64FYS317 : exactly what i was looking for. I had my types like:type Msg = A | B | UnionType

U64FYS317 : but I needed A | B | UnionType UnionType
U64FYS317 : I guess I'm not as comfortable with the idea of tagging as I hoped
U2LAL86AY : it's not possible to do it without this.yes but just a trick that i use -> i always use `MsgFor_` and whatever follows -> this way i know this i need to delegate