U623QK78C : I'm trying to create a module which uses ports and to integrate it a Main app.
U623QK78C : But subscriptions require some Msg type so the module has a Msg and the MAin has as well
U3SJEDR96 : <@U638BAUJZ> I tried to have a look but there's a whole lot of code and it doesn't seem all that easy to get the app working locally in order to reproduce it
U623QK78C : Is my thinking wrong here?
U3SJEDR96 : <@U623QK78C> yeah, so a subscription looks like this: port mySub : (IncomingType -&gt; msg) -&gt; Sub msg` right? In order to use that from your main app, you can just have a `subscriptions = mySub MyMsg` there - there is no need to insert an extra layer somewhere in between
U623QK78C : <@U3SJEDR96> in the Main? or the Port?
U3SJEDR96 : in the port module, you'd have `port mySub : (IncomingType -&gt; msg) -&gt; Sub msg`. In the main module, you'd have `subscriptions = mySub MyMsg`
U623QK78C : yes, that is what I currently have.
U623QK78C : It works, but the Msg type of the Main now needs to have every MyMsg as well as the actual app commands.
U623QK78C : So the Msg type of the Main became unwieldy, so I was hoping to off-load a few message definitions to the Port module.
U623QK78C : ```type Msg = PortMsg Port.Msg | &lt;rest...&gt;```
U623QK78C : seemed not really the solution
U3SJEDR96 : Ah. Well that's a possibility too, but then you're spreading out the logic of your application; which might make sense, depending on your exact use-case. To do that, you'd create a `subscribeToThis : (Msg -&gt; wrapper) -&gt; Sub wrapper` function in the port module, and give it an implementation like `subscribeToThis wrapper = mySub SomePortMessage |&gt; Sub.map wrapper`
U623QK78C : I see
U623QK78C : If I wanted to put the port logic into a separate module as a reusable package is that the way to do it?
U3SJEDR96 : Yeah, pretty much. Though keep in mind that port modules cannot be shared on <http://package.elm-lang.org|package.elm-lang.org>
U623QK78C : Thanx
U61RNCASK : Anyone know when Elm 0.19 is expected roughly? 2017? 2018?
U3SJEDR96 : <https://github.com/elm-lang/projects/blob/master/roadmap.md#what-is-the-timeline>
U0CLDU8UB : If I had to guess I'd say this year, based on nothing in particular :slightly_smiling_face:
U0FP80EKB : I'm hoping for October-ish :slightly_smiling_face: We could start a betting pool
U61RNCASK : Glad to hear that 2017 is more likely than 2018.
U0FP80EKB : Mine, like <@U0CLDU8UB>'s, is based on pure wishing, not on any actual knowledge or anything.
U61RNCASK : Is the work in progress publicly accessible on GitHub?
U0FP80EKB : <https://github.com/elm-lang>
U0FP80EKB : Although my understanding is that there is A LOT of work being done that isn't in there yet.
U61RNCASK : You probably have a better feel for it
U0FP80EKB : Experimentation, thinking, planning, etc.
U0FP80EKB : I wouldn't count on me having a better feel for it. :slightly_smiling_face:
U61RNCASK : <@U0FP80EKB> Ok that makes sense.
U0FP80EKB : Just don't want you to get the wrong impression about any knowledge I have. :slightly_smiling_face:
U41NK9BM4 : Look at the `dev` branches.
U61RNCASK : Is elm-node presently the best way to run on Node?
U0FP80EKB : I'd actually love it in September, as we are going to be going very heavy on having our app hosted on our customers' websites as embeds. The work to shrink the size will be happy for us.
U0FP80EKB : I think we'll benefit a lot from the DCE stuff
U0CLDU8UB : There is no good way to run Elm on Node
U0CLDU8UB : What are you hoping to do?
U61RNCASK : <@U0CLDU8UB> Read a file and write another. I'm aware this isn't anywhere near supported.
U61RNCASK : This looks like it might work <https://github.com/ElmCast/elm-node>
U0CLDU8UB : Yeah, it's going to involve a lot of ports and depending on what you're doing, maybe not that much Elm :slightly_smiling_face:
U0FP80EKB : I would think of it as "reading from in, writing to out" then pipe, rather than thinking of it in terms of files.
U0CLDU8UB : I would even claim you'd have a better time doing that sort of stuff in Haskell, but we all have our hobbies
U61RNCASK : Thanks. The native interop story seems pretty good with ports and native modules.
U61RNCASK : <@U0CLDU8UB> Haskell or F# have this covered but I need Elm in this case.
U3SJEDR96 : ~native~ _kernel_ code is about the easiest way to throw all guarantees that Elm can give you. If at all possible, use ports :slightly_smiling_face:
U61RNCASK : <@U3SJEDR96> Native has been renamed to Kernel as of late? I think I saw a commit to that effect

somewhere?
U3SJEDR96 : <@U61RNCASK> yes. Well, reframed is a better word. "Native" sounds as if it belongs there. Kernel implies low level and "extreme caution is advised".
U0CLDU8UB : <@U61RNCASK> If you do want to do it, you could take a look at `stoeffel/elm-verify-examples` on GitHub. It's mostly written in Elm, even though it deals with text files.
U5R6L5MT4 : Does anyone has an example of Dict.update usage?
U5R6L5MT4 : I am trying something like that
U5R6L5MT4 : { model | books = Dict.update isbn (\book -&gt; { book | review = review }) model.books }
U5R6L5MT4 : But it doesn't seems valid
U5R6L5MT4 : Ok I didn't handled the Maybe case
U5R6L5MT4 : Going with
U5R6L5MT4 : SetReview isbn review -&gt;   { model | books = Dict.update isbn (updateBook review) model.books }

```
updateBook : Review -&gt; Maybe Book -&gt; Maybe Book
updateBook review book =
    case book of
        Nothing -&gt;
            Nothing

        Just book -&gt;
            Just { book | review = review }
```

U3SJEDR96 : Yeah, so the easiest way to do that is `Dict.update isbn (Maybe.map (\book -&gt; { book | review = review })) model.books`
U638BAUJZ : In Visual Studio Code, how do I observe output from Debug.log?
U638BAUJZ : I checked the "Debug Console" tab, however I don't see any debugging info.
U3SJEDR96 : It appears in your browser's dev console
U3SJEDR96 : not sure if VC has some sort of integration with the browser?
U638BAUJZ : I'm using elm-live, if that means anything...
U638BAUJZ : Found it. Thanks!
U638BAUJZ : I'm new to web dev. So there's a lot of fundamentals that I don't have yet.
U3SJEDR96 : No worries! :slightly_smiling_face:
U1NME8MS8 : Hi!
U1NME8MS8 : Does someone has a recommendation how to make <https://gist.github.com/dawehner/0d1ce58a402af5c959ce3597dc731788> nicer, aka. not having a lot of case -&gt; Just else Nothing ...
U0CLDU8UB : Oh wow. This might be a case for `andThen`. Hang on a bit.
U1NME8MS8 : yeah right, this can't be right :slightly_smiling_face:
U5R6L5MT4 : Is it possible to decode this kind of JSON: <http://openlibrary.org/api/books?bibkeys=ISBN:0385472579,LCCN:62019420&amp;jscmd=data&amp;format=json>
U5R6L5MT4 : in a dict
U3SJEDR96 : <@U5R6L5MT4> it's not a straight up key-value list, so it would depend on what exactly you need
U1NME8MS8 : <@U5R6L5MT4> I don't think there should be a problem. Do you know already which data you actually need?
U3SJEDR96 : it's definitely feasible to make a nice decoder for that
U5R6L5MT4 : Basically, at the end I want a Dict ISBN Book
U5R6L5MT4 : ISBN being the part after the colum in the key (so I would need to map at some point)
U5R6L5MT4 : And then the I want to decode the value as a book
U5R6L5MT4 : type alias Book =   { title : String

```
    , subtitle : String
    , authors : List String
    , publish_date : String
    , url : String}
```

U1NME8MS8 : <@U5R6L5MT4> so start with defining a decoder for a single book
U5R6L5MT4 : ok
U5R6L5MT4 : I can do that
U3SJEDR96 : The do something like `keyValues bookDecoder |&gt; Json.map (List.map (Tuple.mapFirst extractISBN)

&gt;&gt; Dict.fromList) `
U0CLDU8UB : <@U1NME8MS8> I am not entirely sure if I did this correctly without the rest of the code, but this compiles, at least:
```
parseSubrequestsResponse response =
    Dict.get "Content-Type" response.headers
        |&gt; Maybe.andThen
            (\headerValue -&gt;
                List.head &lt;| Regex.find (Regex.AtMost 1) (Regex.regex "boundary=\\\"([^\"]+)\\\"") headerValue
            )
        |&gt; Maybe.andThen (\match -&gt; List.head match.submatches)
        |&gt; Maybe.andThen identity
        |&gt; Maybe.andThen
            (\boundary -&gt;
                String.split ("--" ++ boundary) response.body
                    |&gt; List.filter (\string -&gt; string /= "" &amp;&amp; string /= "--")
                    |&gt; List.map (String.split "\x0D \n")
                    |&gt; List.map secondElement
                    |&gt; filterNothing
            )
        |&gt; Maybe.map Ok
        |&gt; Maybe.withDefault (Err "no content-type specified")
```

U5R6L5MT4 : <@U3SJEDR96> ok I will try that
U1NME8MS8 : <@U0CLDU8UB> oh I see, Maybe.andThen, this is super powerful!!
U0CLDU8UB : Some of the anonymous functions could be made a little shorter too, like `(\match -&gt; List.head match.submatches)` --&gt; `(List.head &lt;&lt; .submatches)`, but whichever you prefer!
U1NME8MS8 : <@U0CLDU8UB> oh yeah I'm not into composing functions like that yet, but maybe this is a good place getting more familiar with it
U0CLDU8UB : There's no reason to, just use the form that makes most sense to _you_.
U1NME8MS8 : <@U0CLDU8UB> This is why I love elm, it has all those reasonable / practical people!
U3SJEDR96 : `Maybe.andThen identity` is particularly nice, actually haven't had to use that before :stuck_out_tongue:
U0CLDU8UB : Me neither, to be honest :smile:
U3SJEDR96 : Then again, `Maybe (Maybe v)` doesn't happen _that_ often. Should have a `type alias Maaaaaybe a = Maybe (Maybe a)`