U17R26VR8 : then i'd change your TogglePaused to be `TogglePaused Bool Call` so you have the call to change so you don't need to worry about if you're not on a call, because you can only toggle a state if you are in a call
U17R26VR8 : is CallState going to be:```
paused: Bool,
hold: Bool,
otherFlag: Bool
```

U17R26VR8 : where only one of them can ever be true at a time?
U37HUSJ4R : to make it more complex no :stuck_out_tongue:
U37HUSJ4R : multiple can be true
U17R26VR8 : ah ok, fair enough
U37HUSJ4R : for the union type did you mean to have `Call` in there?
U17R26VR8 : i think what you have makes sense then, it's just that because you've got 3 levels of nested records, it's going to be a pain to reach into it without helper functions
U17R26VR8 : the `Call` in the union type is a tag, it doesn't have anyhting to do with the record Call
U17R26VR8 : i thought it was a CallState can be in any of one state so the default state is you're talking to someone so i called it `Call`
U17R26VR8 : anyway, to get back to your original question, you can write a function like:```
updatePaused: Bool -&gt; Call -&gt; Call
updatePaused newValue ({controls} as call) =
  { call | controls = { paused = newValue } }
```

U17R26VR8 : (swapped args around to allow piping)
U37HUSJ4R : really nice :smile:
U37HUSJ4R : I guess I am going to need helper functions to update, this is ok though
U6EAT2Z37 : Why pull controls out if you're not going to use them?
U6EAT2Z37 : oh... you missed out `{ controls | ...`
U6EAT2Z37 : ```updatePaused: Bool -&gt; Call -&gt; Call
updatePaused newValue ({controls} as call) =
  { call | controls = { controls | paused = newValue } }
```

U6EAT2Z37 : What a mouthful!
U37HUSJ4R : I much prefer the union type way
U6EAT2Z37 : Just FYI, the "meaning" of a record is the same "meaning" as a union type, it's just the fields are named.
U1ZCL9GAX : new dreambuggy demo, 100% elm (+ glsl shaders):
<https://www.youtube.com/watch?v=RDFuTzPQ3Sc>
U23SA861Y : schweet
U1ZFF0E5P : any pointers on how to implement this? I can't get it to work ```everyDictDecoder : Decoder k -&gt; Decoder v -&gt; Decoder (EveryDict k v)everyDictDecoder keyDecoder valueDecoder =
```

U153UK3FA : <@U1ZFF0E5P> <http://package.elm-lang.org/packages/elm-lang/core/5.1.1/Json-Decode#map2>
U23SA861Y : so that will give you a dict with 1 value, but what you perhaps want is to create a decoder of `Decoder List (k,v)` in which case you can map over it with `EveryDict.fromList`
U6E03KDPE : Just curious, has anyone used elm as a stepping stone to learning Haskell? And if so, how big of a transition was it?