U23SA861Y : umm well in a sense yes. If you are using a union type an encoder would be simple enough to construct
U23SA861Y : if you have control over then endpoint you probably shouldn have different types of data going to the same place but if you have to then..
U2GTQM83A : Hmm. but I'm not using a union type. That's the thing. I know exactly the type I want. Sometimes I want a Book Author. Sometimes I want a Book String.
U23SA861Y : in elm it should be a union type
U23SA861Y : the actual record you encode doesn't have to be, but in elm you need something to tell you what data you have and the type you should use for that is a union
U2GTQM83A : Not in this case. I have a big codebase with different Elm programs. They all share the same types, but each will use the type with different content in it.
U2GTQM83A : Some might need to access the author of the book, so I ask the server to do a join in the database and send that. Another program will only need the book's own properties. So the author field doesn't need to be populated.
U2GTQM83A : Here is an example decoder from my production codebase:```
```
userStructure : Decoder position -&gt; Decoder contact -&gt; Decoder team -&gt; Decoder (UserStructure position contact team)
userStructure positionDecoder contactDecoder teamDecoder =
    decode UserStructure
        |&gt; required "_id" string
        |&gt; required "email" string
        |&gt; required "active" bool
        |&gt; required "permission" (adtDecoder userPermission)
        |&gt; optional "position" (nullable positionDecoder) Nothing
        |&gt; optional "contact" (nullable contactDecoder) Nothing
        |&gt; optional "team" (nullable teamDecoder) Nothing

```

U2GTQM83A : In this case, position team and contact may be strings or something else.
U2GTQM83A : I know that I can map from a JS string to my type. Now I need to map from my type to a specific Elm value.
U2GTQM83A : I can go from one decoder to a string. like this.```
```
projectToEmbed : Decoder Project -&gt; String
encodeProject a =
    "created_by"
```

U2GTQM83A : But I don't know how to do it if my decoder type is parameterised. Like `Decoder (Project a)`
U0CL0AS3V : &gt; What I need is a function that given the type `Book String` would return `[]` and given the type `Book Author` would return `["author"]`.&gt; [...]
&gt; I know exactly the type I want. Sometimes I want a Book Author. Sometimes I want a Book String.

since you know exactly the type you want, why not write two functions?

one takes `Book String` and the other takes `Book Author`

U2GTQM83A : Because I have at least 5 types which take three type parameters each.
U2GTQM83A : That would be a lot of functions
U2GTQM83A : Aaand. Because it would end up not being type safe.
U23SA861Y : I don't think it's possible to create a parameterized decoder....
U2GTQM83A : Because if something changes, like the string that a project should return. If I forget to replace in one of the functions, it would not create an error.
U23SA861Y : because that would cause a runtime error
U2GTQM83A : I have parameterised decoders.
U0CL0AS3V : what are the 5 types with the 3 type parameters each?
U2GTQM83A : What I need is a way to get a string depending on the decoder type.
U2GTQM83A : Yep.
U2GTQM83A : Here is an example: ```
```
type alias ProjectStructure user company contact =
    { id : String
    , code : String
```

```
    , name : String
    , description : String
    , type_ : ProjectType
    , startDate : Date
    , endDate : Maybe Date
    , budget : String
    , requiredFiles : List String
    , notes : String
    , company : company
    , owner : user
    , contacts : List (ProjectContact contact)
    , rates : List PositionRate
    , created_by : user
    , confidential : Bool
    }
```

U2GTQM83A : Like Project, I have Deliverable, Contact, User, Team, TimeRecording.....
U0CL0AS3V : oh
U0CL0AS3V : so you could presumably do something like
U0CL0AS3V : `ProjectStructure user company contact -&gt; String`
U2GTQM83A : Yes!
U0CL0AS3V : and inside it pattern matches on the argument 3 times
U2GTQM83A : That's what I want.
U0CL0AS3V : first one looks like this
U0CL0AS3V : oh, sorry