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
U0FP80EKB : Oh, here is one of the bottom types, `AppearanceFieldInfo````
type AppearanceFieldInfo
    = BackgroundColor ColorPicker.ColorControl
    | SubmitButtonColor ColorPicker.ColorControl
    | AccentColor ColorPicker.ColorControl
    | QuestionBackgroundColor ColorPicker.ColorControl
    | HeaderTextColor ColorPicker.ColorControl
    | PrimaryTextColor ColorPicker.ColorControl
    | SecondaryTextColor ColorPicker.ColorControl
    | HeaderImage ImagePicker.ImageControl
    | FontFamily String
    | FontSize Int
```

U0FP80EKB : There you can see all the possible things that can be updated. So, when `FontSize` is edited, the message comes up like `FontSize 16` or something
U0FP80EKB : So, the payload has the new value
U0FP80EKB : Our situation has a bit of complexity because we used the same editor customizing 3 different types of embeds, so there are general fields and type-specific fields that need to be edited.
U5QJW0DDE : i suppose that sending a Msg then means wrapping the FontSize in an FieldInfo and then in another, etc
U0FP80EKB : Yup
U0FP80EKB : For example```
        generalControls =
            Html.map UpdateEmbed <|
                div [class "control-group"]
                    [CommonControls.textControl "Name" embed.displayName (FieldHelpers.GeneralField <<
FieldHelpers.DisplayName)
                    ]
```

U0FP80EKB : This edits the "Name" field, so the message ends up something like `UpdateEmbed (FieldHelpers.GeneralField (FieldHelpers.DisplayName "Corey"))`
U0FP80EKB : Luckily the constructors for a union type are composable, since they are just ordinary functions, so we can do `FieldsHelpers.GeneralField &amp;lt;&amp;lt; FieldHelpers.DisplaName` as the event handler
U5QJW0DDE : I like it
U0FP80EKB : Keep it in your pocket as a tool for when you start noticing groupings in your `Msg` constructors. That's often a flag that you'll want to collapse some of them.
U0FP80EKB : But, it definitely adds complexity to the system, so I will always say "extract to this, don't plan for it"
U0FP80EKB : Our system is constantly going through cycles of extraction. We're currently replacing our original, faulty design of `AppearanceFields` and `ContentFields` with `GeneralFields` and `TypeSpecificFields` :slightly_smiling_face: Luckily Elm makes it pretty safe to do these refactorings, although it will be a bit time-consuming.
U5QJW0DDE : i assume in your example embed would refer to a record somewhere
U5QJW0DDE : and CommonControls.textControl is a function returning Html?
U0FP80EKB : yeah. It is just a wrapper for a view function.
U0FP80EKB : ```textControl : String -&gt; String -&gt; (String -&gt; msg) -&gt; Html msg
textControl labelText value onInputMsg =
    controlWithLabel labelText &lt;|
        input
```

```
        [ class "text-control"
        , defaultValue value
        , onInput onInputMsg
        ]
        []
```

U0FP80EKB : `Embed` is a module that gives an entry point to building the embed modules. We use the same code for the actual live embed that sites on our customers' sites and the customization editor in our admin site.
U0FP80EKB : We have a bunch of records floating around, but, at the core, it looks like```
type alias Embed =
  { id : Id
  , displayName : String
  , embedType : String
  , apiUrls : ApiUrls
  , appearance : UI.Appearance
  , content : Content
  }
```

U0FP80EKB : We initialize it via flags at load time.
U5QJW0DDE : thanks, nice summary
U5QJW0DDE : reading more code like this from real-world examples will be good to understand all the syntax and architecture and best practices
U0FP80EKB : Yeah. I wouldn't call ours best practices, though. :slightly_smiling_face: We're constantly evolving the design as we feel pressure. There are a couple good ones, like the collapsing technique for `Msg`
U5QJW0DDE : i'm switching my mind from clojurescript so for me the first thing i'm always doing is trying to follow the types through the flow
U0FP80EKB : That can be a reasonable way to look at it.
U0FP80EKB : At any point in time, our codebase consists of what I call a bunch of "fish with goat heads". Halfway evolved to the final design. :slightly_smiling_face:
```