The Custom Scalar Checklist

I hope you find this checklist useful! It's a repeatable process for refining your GraphQL Schema to make your Elm app easier to maintain and less likely to have bugs!

Here is some background on why: https://incrementalelm.com/custom-scalar-checklist

Now let's get to the steps!

There are three sections:

1. Turn your schema into a todo list!
2. Identify the "Contract Category" (if any)
3. Use Some Custom Scalars!

1. Turn your schema into a todo list!

In this section we'll copy-paste your entire GraphQL Schema into your favorite editor, and clear out some distractions! That will be our todo list for finding potential Custom Scalars.

- [ ] Get your GraphQL Schema in SDL notation. If you don't have it in a file that you can copy easily, you can just follow these two steps
  - [ ] Open Graphiql (or GraphQL playground) so you can run a query against your GraphQL API
  - [ ] Run this introspection query
  - [ ] Copy-paste the JSON from the introspection query into Sandbox - CodeSandbox
- [ ] Copy-paste the SDL notation of your entire GraphQL Schema into a text editor
- [ ] Delete everything from the Schema **except for the fields and arguments that include one of the following Scalars**: `String`, `Float`, `Int`, `Id`

You should be left only with things like this:

```
url: String
phoneNumbers: [String!]!
rating: Float!
id: Id
```

Now this is our todo list of items that could potentially be improved by introducing a

Custom Scalar!

2. Identify the "Contract Category" (if any)

Custom Scalars are a way to describe **where** a field can be used, **how** to use it, or **what** the field represents. This section describes all the **Contract Categories** to help you easily identify what might be a good fit for a Custom Scalar.

For each field you have left now after the last section:

- [ ] Write down whether any of the contracts below applies

Unit Contract

It has a **Unit Contract** if...

- The docs for the field mention a unit `height: Float! # patient's height in inches`
- The field name mentions a unit `degreesCelsius: Float!`
- Note that Units don't just have to be temperatures, lengths, speeds, positional coordinates. Star ratings are also a unit of measurement (would you rate a book "5", or "5 stars"? "5" isn't very meaningful without a unit). "OutOf10" is also a type of unit.

Format Contract

- The docs mention a particular format or Standard, example: `countryCode: String! # ISO 3166-1 alpha-2 code for this country.`
- `createdAt: Int! -- POSIX time`

Context Contract - "Make sure you use this in the right place"

- `url: String!` if this were a `URL` Scalar, it would allow you to enforce that it can **only** be used where a URL should be used (like in an `href` attribute).
- `imageUrl: String!` (we want to make sure this is only used as `<img src="..." >`).
- `discountCode: String!` (I sure hope we don't try to apply a discount code using something other than a discount code! That would lead to some frustrating bugs!)
- `authToken: String!`

Id Contract

GraphQL provides an `Id` Scalar by default. I recommend against using this type as it is to general. Instead, I recommend creating a Custom Scalar for each type of Id in your system. There's really not a lot of overhead, and there are a lot of benefits!

You can think of this as a special case of a Context Contract.

- `type Product { productId: Id! }` - better to use `type Product { id: ProductId! }`. That way you can't
- `userId: String!`

Otherwise, perhaps it represents one of these things which may be great as is and not as a Custom Scalar:

- `count: Int!`
- `description: String!`

Enumeration Contract

- Its comment **enumerates** a discrete number of options: `price: String # either $, $$, $$$, $$$$, or null if we don't have price available for business`
- You know that it can have discrete values, like `sortOrder: String!`, or `dayOfWeek: Int!`

All of these are **contracts**. And **Custom Scalars are just a way to represent contracts**!!! Use them, you'll love it!!

Everything Else - Raw Data

Anything else essentially just represents raw data, with no special format, no special units, no finite amount.

Here are some examples of raw data that has no unit, format, or discrete options:

- `firstName: String`, `lastName: String`
- `address1: String!`, `address2: String`
- `city: String!`
- `totalCount: Int!`
- `articleBody: String!`
- `title: String!`

That doesn't mean it's **bad** to represent any of these things as Custom Scalars if you have some reason to. You might want to start with the low hanging fruit of all the others **Contract Categories**, though, and then sleep on the fields in this category. Instead, I would make sure that these are in a nicely named GraphQL `Object`... Objects can give semantic meaning in a way that is similar to a Custom Scalar.

3. Use Some Custom Scalars!

Now comes the fun part! Go through everything that met one of the above **Contract**

Categories, and change its type.

- [ ] Take all of your **Enum Contracts** and turn them into **GraphQL Enums**.
- [ ] Take all the other contracts and turn them into Custom Scalars.