Part 5 - Early bound types using Dataverse-ify

This is part of the course 'Scott's guide to building Power Apps JavaScript Web Resources using TypeScript'.

In this fifth part we will cover creating a early bound types so that you can call the reference attribute names in form scripts and call the `WebApi` with ease.

`Dataverse-ify` is an open source library that aims to provide an interface to the Dataverse `WebApi` from TypeScript that works in a similar way to the C# `IOrganisationService` so that you do not need to code around the complexities of the native `WebApi` syntax.

Design Goals

- Implement an API for use from TypeScript that is close to `IOrganizationService` for use in Model-Driven Form JS Web-Resources or PCF controls.
- Cross-Platform - pure NodeJS - runs inside VSCode on Windows/Mac/Linux
- Early bound generation of Entities/Actions/Functions with customizable templates.
- Be as unopinionated as possible - but still promote TypeScript best practices.
- Allow integration testing from inside VSCode/NodeJS tests.

See https://github.com/scottdurow/dataverse-ify/wiki

Adding `dataverse-ify` to your JavaScript Webresource TypeScript Project

Authenticate against your environment

`Dataverse-ify` uses a authentication token that is encrypted and stored on your machine using a library called `dataverse-auth`.

At your VSCode command line run:

```
npx dataverse-auth myorg.crm.dynamics.com
```

Replace `myorg.crm.dyanmics.com` with the URL of your tenant (without the `https://`)

You will be prompted to login - if you have MFA enabled you will also need to provide the necessary details.

Once you have logged in, a authentication token will be saved and you should not need to login again unless it expires (usually 90 days) or is invalidated (e.g. by your administrator making changes)

Installing `Dataverse-ify`

To use the `WebApi` using early-bound types you will need to install the `dataverse-ify` library using the following at your VSCode command line:

```
npm install dataverse-ify --save
```

Generating early-bound types

To initialise your early-bound type generation, use the following at the VSCode command line:

```
npx dataverse-gen init
```

You will be asked to select from the environments you have authenticated against (if you have more than one).

Select the Entities that you wish to generate types for. In this example we will select `account`, `opportunity`, `email` & `activityparty`

```
√ Select server to connect to · 10
Using server: dev1demos14.crm3.dynamics.com
Fetching EDMX metadata
? Select entities to include [type to filter list, space to select] ... acc
√ account
√ accountleads
√ channelaccessprofileentityaccesslevel
√ msdyn_accountpricelist
√ msdyn_bpf_2c5fe86acc8b414b8322ae571000c799
√ msdyn_msdyn_functionallocation_account
√ msdyn_ocwhatsappchannelaccount
√ principalobjectaccess
√ principalobjectattributeaccess
[Space - Select] [Return Accept] [Scroll up and down to see more]
```

Next you will be prompted to select the actions you wish to generate types for, we will select `WinOpportunity`. You can also select from any custom actions or custom APIs you have created.

```
? Select actions to include [type to filter list, space to select] ... winop
√ WinOpportunity
[Space - Select] [Return Accept] [Scroll up and down to see more]
```

Lastly you are asked to select any functions – we will not select any just yet.

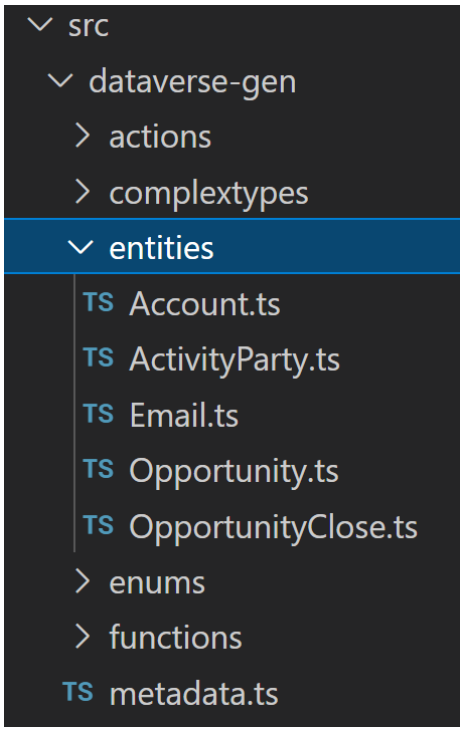
You are then prompted if you want to generate the types:
Select Yes

```
√ Would you like to generate the types now? (y/N) · true
```

In the root of your project you should now see a file called `.dataverse-gen.json` that looks like this:

```
{
  "entities": [
    "account",
    "activityparty",
    "email",
    "opportunity"
  ],
  "actions": [
    "WinOpportunity"
  ],
  "functions": [],
  "output": {
    "outputRoot": "./src/dataverse-gen"
  }
}
```

There will also be a new folder called dataverse-gen under your src folder. This will contain the early-bound types and attribute constants.



We can now update our `AccountForm.ts` with the attribute name constants `AccountAttributes.WebSiteURL`

```
static async onload(context: Xrm.Events.EventContext): Promise<void> {

 context.getFormContext().getAttribute(AccountAttributes.WebSiteURL).addOnChange
(AccountForm.onWebsiteChanged);
}
```

VSCode should automatically import the `AccountAttributes` for you:

```
import { AccountAttributes } from "../dataverse-gen/entities/Account";
```

Using this approach will result in less errors due to logical name inconsistencies. Additionally, if any attributes are deleted that are referenced by your code, when you re-run the dataverse-gen you will see build errors.

You can re-generate the types at any time using:

```
npx dataverse-gen
```

If you want to add to the metadata added, you can simply re-run:

```
npx dataverse-gen init
```

or you can simply edit the `.dataverse-gen.json` file and run `npx dataverse-gen`

Using custom templates!

If you wanted to just generate Attribute `enum` constants and stop there, you can easily customise the scripts to suit your needs by using:

```
npx dataverse-gen eject
```

This will create a step of templates ready to customise in the `_templates` folder. Once you have made your updates, just run `npx dataverse-gen` again. The templates use the awesome ejs project. E.g.

```
// Attribute constants
export const enum <%- locals.SchemaName %>Attributes {
<%locals.Properties && locals.Properties.forEach(function(property){ _%>
 <%- property.SchemaName %> = "<%- property.Name %>",
<%})_%>
}
```

If you wanted to revert back to the standard templates, just delete the `_templates` folder

Next Up

Now that we've created early bound types, we will use `dataverse-ify` to call the `Xrm WebApi` with ease!