API Documentation

Overview

I decided to do all of my development in Salesforce with Apex. This is mostly because that is what I am most comfortable with in terms of programming. I haven't had to build out my own API calls to this degree before, so I wanted to simplify my approach by using a tool I know well.

As for specific decisions, these came from my own interpretations of the requirements that were provided. Let's break down each part of the requirements.

CSV Parser

We were given the structure of the csv files we should be testing with. I added another column called "Related Object" to make creating relationship fields easier. My code also assumes that the name of the file is the API name of the object that the fields will be created for, which is also explained in the requirements.

Since I wanted this tool to work in Salesforce, I decided the best way to upload the file would be to upload it as a File field in Salesforce. This can be done easily through the UI once you have the CSV downloaded. Similarly, I decided to use a separate CSV for the permissions that should be applied to the new fields. There are no requirements for the name of this file as long as the user who will be using the tool knows the name of it. I use similar logic to parse both files. I use skeleton classes for both parsed fields and parsed permissions that just contain the fields I want to use from these files. The parser turns the columns/rows into values for these fields.

API

My code utilizes the Metadata API for creating the new fields, as well as updating the permissions for Profiles. For Permission Sets, I used standard dml to insert field permissions, but this is not allowed with Profiles so I had to utilize the API. I chose to use the Metadata API because based on my research, it is better suited for the task of uploading multiple fields in one single call.

Another requirement was to allow the possibility of different APIs being used in the future. This means I needed to completely separate my logic from the actual API calls. I achieved this by using an interface and a Factory class that uses the API specified in a Custom Metadata Type. Right now, the package only contains code for the Metadata API, but if someone were to build a solution using a different API, all they would have to do is substitute the Custom Metadata Type and add logic to the Factory to use the correct API. The fundamental logic would remain the same.

As per the instructions, my code works with multiple common field types. But for the sake of time and simplicity, I stuck to only the data types specified in the instructions. Other data types may not work if tested, but I could add them to my logic if needed. It does work with both Lookup and MasterDetail relationships.

Permissions

I mentioned this earlier, but I decided to use a csv file for the permissions, and I decided to use Apex code and the Metadata API for creating the permissions. My reasoning for this is similar to my reasoning for using APex in the first place. But I also wanted to keep everything relatively together and easy to package. The permissions csv uses the columns Name, Profile or Permission Set, Readable, and Editable. Also as stated before, I updated the Profile permissions via Metadata API since dml is not allowed, and used dml for the Permission Sets. I make sure to call the dml after the API call as to prevent errors. And since I am returning the API results in the code, those callouts are made synchronously and the dml has no issue finding the newly created fields.

Packaging

Keeping with the theme of making everything usable in Salesforce, I decided to bundle all of my code into an unmanaged package. As of now, there are no concerns for protecting this code and it may be good to allow someone to edit it if they want to use a different API, so unmanaged seemed like the best approach to me. It contains all the classes and test classes as well as the Custom Metadata Type. Once it’s installed in an org, the running user only needs to upload their own files and create an instance of the Custom Metadata Type called “Default” so the code will know which API to use. In this case, they will have to make it use the Metadata API.