**SDN List Monitor – Design Rationale**

My initial idea for the SDN List Monitor (SLM henceforth) was to create a pollable service that would retrieve the remote SDN List XML file from the US Treasury Department website and persist to a local file. This is because the requirements stated that the list would need to be compared (existing vs. previous), as opposed to any other form of comparison. I also decided that I would limit the design to a service that was pollable as opposed to one that could independently monitor the list. My thinking here is that it would be comparatively straightforward to setup some sort of polling service, such as an Azure function app, that leveraged the SLM service. I decided to make use of .Net Core Web-API as the service’s interface, and standard C# HTTP client and XML serialisation tools, as initially there was no need to introduce any other tooling.

However, as I started to make progress with the SDN List retrieval functionality, I decided that I wanted to extend the SLM design to incorporate persisting and retrieval of the XML data in a local database. My reasoning was that I was already de-serialising the XML into domain models, so it seemed counter-productive to then serialise them to disk and then repeated this process each time I conducted a comparison. Moreover, having the local database opens up the possibility for future enhancements to the SDN service for storing a history of different versions in whole or diff format. Therefore, I introduce Entity Framework (EF) Core to the solution, as I decided I wanted to make use of a code-first approach to persisting and retrieving the domain models.

In order to create the domain models I made use of the SDN.xsd definition file that is also provided by the US Treasure Department, and I pointed it at the Microsoft XML Schema Definition Tool (xsd.exe). This gave me some POCO models that, at first, appeared ready for use with EF Core. However, I then discovered that the top-level model and two child models of the list did not have keys, so I had to add these, ensuring to instruct EF Core to generate these on insertion only. I also had to handle the ‘programList’ array of strings in the ‘SdnEntry’ model as these were also keyless. Following some exploration, a simple solution to this was to join them into a delimited string, and then explode the string on retrieval of the domain model.

At this stage I separated the solution into five distinct projects, fronted by interfaces as appropriate:

* Models (domain models)
* SdnMonitorApi (.Net Core Web API)
* SdnMonitorCore (XML list retrieval, de-serialisation, comparison, and storage)
* SdnMonitorData (EF Core related CRUD functionality)
* SdnMonitorTests (Project unit tests)

Upon confirming that I could retrieve, de-serialise and persist the SDN List to the local database, I moved onto the task of examining a newly retrieved and existing version of the list. I initially had a look in Nuget to see if there were any already-baked solutions for this, and I discovered ObjectsComparer (1). After having a play with it, I came to appreciate what it could offer to SLM service in future as it was able to clearly tease out the exact differences between SDN Entry records that appeared in both the remote and local versions of the list. However as the current requirements did not necessitate such functionality, I decide instead to focus on using overridden versions of ‘.Equals()’ and ‘.GetHashCode()’ for the ‘SdnEntry’ model and its subordinate models. I also applied the ‘IEquatable’ interface to each of these to ensure that these models could be compared like-for-like. I had to ensure that these methods did not make use of the aforementioned custom keys, as these would not be present in retrieved versions of the list.

Once I got all the model comparison logic in place, I decided that it would be most efficient to compare the models and build up three collections as required; one for added records, one for deleted records and one for modified records. The counts of these collections would then be returned as per the requirements to the consumer of the SLM service. In order to attempt to short-circuit the need for comparison, I decide to compare the header of the XML file for the ‘Record\_Count’ and ‘Publish\_Date’ attributes. If both of these matched between the retrieved and local versions of the list, then I would consider the list had not changed since the last time it was retrieved. This rationale may not be perfect however, and a complete comparison of the list may be needed, depending on how reliable the XML header proves to be.

The final part of the work was focused on ensuring that the detected changes were applied to the local version of the SDN list in the database. I decided that I would keep the header of the list up to date, separate from updating the SDN Entry records associated with the one SDN List record in the database. Whilst doing this I found that persisting data with EF Core was very easy, however deletion and updating proved more complex. With deletions I eventually discovered that EF does actually delete records when the ‘.Remove()’ function is used. It instead removes the record from the context, as part of change tracking. In order to get around this, I discovered that EF has an entity state model that has to be used instead (2). With updates of records, I discovered that because I am currently operating in disconnected mode, I would have to manually transfer each attribute from updated SDN Entries and their associated subordinate records into their corresponding records in the database. In order to avoid doing this, I instead tried to approach the update as a delete and then add approach. However, this approach also did not work as I discovered that I had not enabled cascade deleting (3) in the EF ‘ModelBuilder’. This is something that would need to be fixed in a future release of the service.

I unfortunately did not have an opportunity to create unit tests in the project due to time constraints. However, I did manage to create some rudimentary tests in the SDN Monitor using scaled-down versions of the XML. These can be seen in ‘SdnListMonitor > SdnMonitorCore > SdnMonitor.cs’ class. These were the tests I used to test the functionality of the SLM service. In future I would look to incorporate and expand upon these tests in the ‘SdnMonitorTests’ project.

**References:**

1. ObjectsComparer - <https://github.com/ValeraT1982/ObjectsComparer>
2. Delete Records in Entity Framework Core - <https://www.tektutorialshub.com/entity-framework-core/deleting-records-in-entity-framework-core/>
3. Cascade deleting with EF Core - <https://stackoverflow.com/questions/48692144/cascade-deleting-with-ef-core>