TaxCalcService is a .NET Core 3.1 REST Service that is used by client software to access external tax information providers without the client needing knowledge of the reference source. It allows for extensibility out of the box by designing in the ability to dynamically load replacement or new tax-api callers.

The skeleton implementation solution contains the following project structure:

* TaxCalcService – the main web service
* ExternalTaxApis – a folder which contains the client projects that implement Interface-defined calls to external tax API services. Presently this contains a working TaxJarApi client and a mock ZipTax (non-working).
* ClientDataModel – data definitions representing basic customer, product and order and other data structures for use by internal client applications, the TaxCalcService and the external api clients in their own separate DTO classes. This folder also has an Interfaces subfolder.
* ConsoleUserApp – simple .NET Core console app that does actual calls for tax rates for zip of location as well as submission of a dummy order for querying the taxable amount
* TaxServiceClientCallingTests – NUnit tests
* Documentation – any and all design docs, artifacts, UML diagrams

The TaxCalcService is built with and uses two interfaces. One is for defining what calls are available and implemented for client software to expect and the other is for definition of the dynamically loading of the external taxapi clients.

* ITaxCalculatorApi.cs: What TaxCalcService must implement for business logic
* ITaxApiCaller.cs: Information for each loadable api-client assembly

TaxCalcService has a class *TaxApiChooser.cs*, that does the dynamic assembly loading of api clients. It works off of a JSON file *taxapiproviders.json* which contains information on the assemblies themselves as well as a setting for which is the default to use, unless a customer choice overrides (future).

Figure 1 taxapiproviders.json

{

"defaulttaxapi": "TaxJarApi",

"taxapiproviders": [

{

"name": "TaxJarApi",

"assemblyfqn": "TaxJarApiClient.TaxJarApiCaller , TaxJarApiClient, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null",

"clrtype": "TaxJarApiCaller",

"geturl": "taxjar.com/v2/tax",

"posturl": "taxjar.com/tax",

"apitoken": "INSERTREALLICENSEDTOKENHERE"

},

{

"name": "ZipTaxMock",

"assemblyfqn": "ZipTaxMock.ZipTaxMock , ZipTaxMockClient, Version=1.0.0.0, Culture=neutral",

"clrtype": "ZipTaxMockCaller",

"geturl": "localhost:8888/v2/tax",

"posturl": "localhost:8888/tax/post",

"apitoken": "123abc"

}

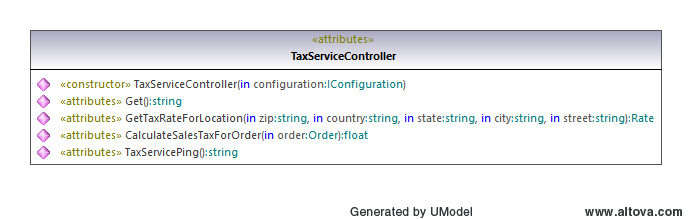
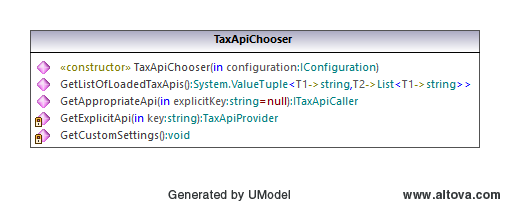
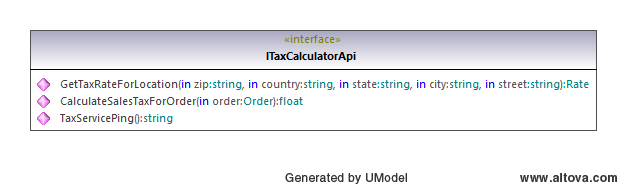
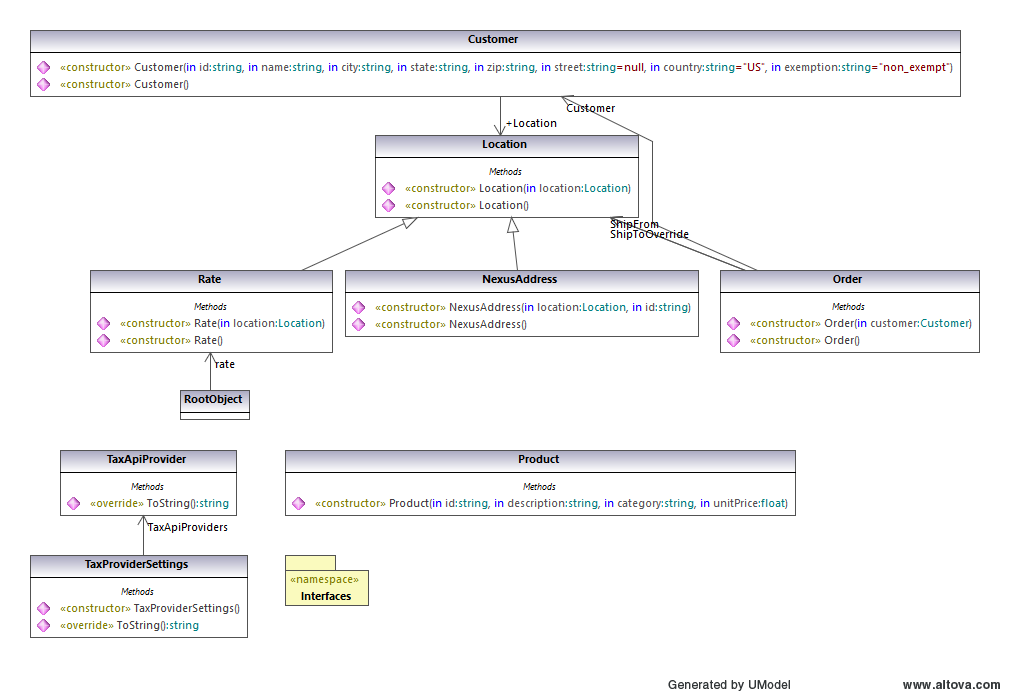
]

}

Note: There are a few different styles of implementation for the REST portion of the service. I chose to test out the .NET Core 3.1’s documented use of returning native values rather than IActionResult.

Diagrams

Below are some UML diagrams that describe some of the key elements of the solution

Sequence Diagrams

