EXOS Microservice Reference Architecture:  
Migration Guide

Version 1.1

PROPRIETARY AND CONFIDENTIAL  
FOR INTERNAL USE ONLY  
DO NOT DISTRIBUTE OUTSIDE THE COMPANY

## Table of Contents

[Table of Contents 1](#_Toc36198969)

[Revision History 1](#_Toc36198970)

[Approvals 1](#_Toc36198971)

[1 Introduction 1](#_Toc36198972)

[2 Project Structure 1](#_Toc36198973)

[2.1 Prefixes and Suffixes 1](#_Toc36198974)

[2.2 Folder Structure 1](#_Toc36198975)

[2.3 Project files 1](#_Toc36198976)

[2.4 Database Projects 1](#_Toc36198977)

[2.5 Deleting files 1](#_Toc36198978)

[2.5.1 Unused misc files 1](#_Toc36198979)

[2.5.2 Unused Docker Files 1](#_Toc36198980)

[2.5.3 Unused appsetting.json.xml file 1](#_Toc36198981)

[2.6 Generated code 1](#_Toc36198982)

[2.7 Project Namespaces and Assembly Names 1](#_Toc36198983)

[2.8 Update .gitignore. 1](#_Toc36198984)

[2.9 Update \*.sln file 1](#_Toc36198985)

[2.10 Update \*.csproj files 1](#_Toc36198986)

[2.10.1 Directory.Build.props changes 1](#_Toc36198987)

[2.10.2 Obsolete Tags 1](#_Toc36198988)

[2.10.3 Test csproj files 1](#_Toc36198989)

[2.11 Solution Folders 1](#_Toc36198990)

[2.12 Readme file 1](#_Toc36198991)

[2.13 Classes and Names. 1](#_Toc36198992)

[2.13.1 Class names files, and folders 1](#_Toc36198993)

[2.13.2 Model class names 1](#_Toc36198994)

[2.13.3 Comments 1](#_Toc36198995)

[2.14 Naming final comments 1](#_Toc36198996)

[3 Upgrade to ASP.NET Core 3.1 1](#_Toc36198997)

[4 Enabling Code any Style Analysis 1](#_Toc36198998)

[5 Wrap any Synchronous Calls with Retry 1](#_Toc36198999)

[6 Separate Borrower Information and Encrypt 1](#_Toc36199000)

[7 Replace Synchronous Updates/Inserts/Deletes with Async ServiceBus Messages 1](#_Toc36199001)

[8 Add ServiceBus Message Listeners 1](#_Toc36199002)

[9 Fix Logging and Instrumentation 1](#_Toc36199003)

[10 Create Unit Tests 1](#_Toc36199004)

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Jacob Slusser | 3/7/2020 | Initial draft | 1.0 |
| Willie Slepecki | 3/16/2020 | Additions to project and repo setup | 1.1 |
|  |  |  |  |

## Approvals

|  |  |  |
| --- | --- | --- |
| **Approval Name** | **Approval Date** | **Comments** |
|  |  |  |

# Introduction

This document describes the most common actions a development team must take to make sure they follow the EXOS reference architecture for microservices.

The following tools are required:

* Visual Studio 2019
* Git – with Windows shell extensions installed

# Project Structure

We are going to strictly enforce a standardized naming convention for projects from this point on. These will include the following

**When moving to this project structure you MUST use the “git mv” command from a Git bash shell to preserve Git history and avoid merge conflicts. DO NOT simply rename them on the files system or through Visual Studio.**

## Prefixes and Suffixes

Git repo suffix

For example, a “VendorProfile” service would have a Git repo named “VendorProfileSvc”.

* If the project is a service, the repo will end with “Svc”
* If the project is a User Interface, the repo will end with “Ui”

Project prefixes and suffixes.

If the project is part of the Exos platform, then the Project name and folder must begin with “Exos.\*” Notice the period. So for the Vendor Profile service. The proper name for the project and the folder would be “Exos.VendorProfileSvc”.

All projects are grouped together and identified by their naming convention.

* “\*Svc” = a rest service project
* “\*WebJob” = an Azure Webjob project
* “\*Listener” = an event listener project
* “\*Lib” = generic library projects

## Folder Structure

The root of a repository is only permitted to have 4 directories

.deploy\*

data

src

tests

All primary source code for a project must be under the “src” directory. The “.deploy” folder is used for the release team. The “tests” directory is to contain all tests that require the source code of the project. This includes Unit tests, Integration tests and performance tests. So for the following project

VendorComplianceSvc/ tree -L 1 -d

.

|-- AscGovLicenseValidatioJob

|-- Exos.VendorComplianceApi

|-- LicenseVerification.ServiceProxy

|-- VendorComplianceApiTests

|-- VendorComplianceListeners

|-- VendorComplianceModels

|-- VendorComplianceProcessor

`-- VendorComplianceSchedulerWebjob

The proper renaming scheme would look like this

VendorComplianceSvc/ tree -L 2 -d

.

|-- .deploy

|-- src

| |-- Exos.AscGovLicenseValidationWebJob

| |-- Exos.VendorComplianceListener

| |-- Exos.LicenseVerificationLib

| |-- Exos.VendorComplianceModelsLib

| |-- Exos.VendorComplianceProcessorLib

| |-- Exos.VendorComplianceSvc

| `-- Exos.VendorComplianceSchedulerWebjob

`-- tests

|-- Exos.VendorComplianceSvc.UnitTests

`-- Exos.VendorComplianceSvc.IntegrationTests

## Project files

Once the folder structure is corrected, next is to check the location of the Visual Studio and standard ServiceLink files. Copy the files at the following share into the root of your repo (alongside the solution file): [\\servicelink.com\slfiles\Development\Architecture Team\Project Files](file:///\\servicelink.com\slfiles\Development\Architecture%20Team\Project%20Files)

The rules for project files are fairly straight forward.

1. The solution file must be in the root of the project
2. The solution file should follow the namespace naming, starting with “Exos.”
3. The individual project files must be in the root of that individual project
4. A “.gitignore” and “.gitattributes” file is generated by VS19 in the root of the project, those are fine.

Once all required files are copied in place, your basic directory tree should look very close to this.

VendorComplianceSvc/ tree -a

.

|-- .deploy

|-- .editorconfig

|-- .gitignore

|-- .gitattributes

|-- Directory.Build.props

|-- Directory.Build.targets

|-- Exos.ruleset

|-- README.md

|-- Exos.VendorComplianceSvc.sln

|-- src

| |-- Exos.AscGovLicenseValidatioWebJob

| | `-- Exos.AscGovLicenseValidationWebJob.csproj

| |-- Exos.LicenseVerificationServiceProxy

| | `-- Exos.LicenseVerificationServiceProxy.csproj

| |-- Exos.VendorComplianceListener

| | `-- Exos.VendorComplianceListener.csproj

| |-- Exos.VendorComplianceModelsLib

| | `-- Exos.VendorComplianceModelsLib.csproj

| |-- Exos.VendorComplianceProcessorLib

| | `-- Exos.VendorComplianceProcessorLib.csproj

| |-- Exos.VendorComplianceSchedulerWebjob

| | `-- Exos.VendorComplianceSchedulerWebjob.csproj

| `-- Exos.VendorComplianceSvc

| `-- Exos.VendorComplianceSvc.csproj

|-- stylecop.json

`-- tests

|-- Exos.VendorComplianceSvc.IntegrationTests

| `-- Exos.VendorComplianceSvc.IntegrationTests.csproj

`-- Exos.VendorComplianceSvc.UnitTests

`-- Exos.VendorComplianceSvc.UnitTests.csproj

## Database Projects

All SQL and CosmosDB projects will be moved to the data folder.

AzureSQL databases are to be moved to the /data/AzureSql folder and Cosmos database seed data will be moved to the /data/CosmosDB folder. Your final directory structure should look something similar to this

VendorComplianceSvc/ tree -a

.

|-- .deploy

|-- .editorconfig

|-- Directory.Build.props

|-- Directory.Build.targets

|-- Exos.ruleset

|-- Exos.VendorComplianceSvc.sln

|-- data

| |-- AzureSql

| | `-- TitleMartDB

| `-- CosmosDB

| `-- EXOSEntityManagementDB

|-- src

| |-- Exos.AscGovLicenseValidatioWebJob

| | `-- Exos.AscGovLicenseValidationWebJob.csproj

| |-- Exos.LicenseVerificationServiceProxy

| | `-- Exos.LicenseVerificationServiceProxy.csproj

| |-- Exos.VendorComplianceListener

| | `-- Exos.VendorComplianceListener.csproj

| |-- Exos.VendorComplianceModelsLib

| | `-- Exos.VendorComplianceModelsLib.csproj

| |-- Exos.VendorComplianceProcessorLib

| | `-- Exos.VendorComplianceProcessorLib.csproj

| |-- Exos.VendorComplianceSchedulerWebjob

| | `-- Exos.VendorComplianceSchedulerWebjob.csproj

| `-- Exos.VendorComplianceSvc

| `-- Exos.VendorComplianceSvc.csproj

|-- stylecop.json

`-- tests

|-- Exos.VendorComplianceSvc.IntegrationTests

| `-- Exos.VendorComplianceSvc.IntegrationTests.csproj

`-- Exos.VendorComplianceSvc.UnitTests

`-- Exos.VendorComplianceSvc.UnitTests.csproj

## Deleting files

There are a list of things that should not be included in the repo.

If you have these directories committed already, you must follow a procedure to delete them

1. close VS19
2. delete the folders
3. git add and commit the changes
4. verify the folders are listed in the .gitignore. If you copied the .gitignore file from the architect shared files folder as instructed, they are already listed.
5. Start VS19 and reopen the solution
6. Go back to a command line and verify that git is not attempting to re-add those files.

These are the directories that should be removed from the repositories

* Any IDE settings directories, this includes .vs, .idea, .project, .metadata and any other IDE files
  + With exception of the .editorconfig you copy in previous steps.
* \*\*/bin
* \*\*/obj
* \*\*/node-modules
* \*\*/.ng\_pkg\_build

Nuget is the one and only system to import binary 3rd party libraries for C# projects. If your project has a folder of libraries that have been previously downloaded and included, those need to be deleted and converted to nuget.

### Unused misc files

If your project has files listed here, delete them

GlobalSuppressions.cs – an old file to suppress scanner rules, no longer permitted

Application Insights/ConnectedService.json – No longer needed

### Unused Docker Files

In the root of the solution are a series of docker-compose\*.\* files. Those files are unused in our environment, they are outdated. Since they are unused, they must be deleted.

In the root of each project is a `Dockerfile`. At this moment, its unused. But there are plans to use this file shortly. Therefore those files can remain.

### Unused appsetting.json.xml file

If your project has an appsettings.json.xml file, delete it. It is no longer used for AKS deployments. Once you delete it, ensure you remove its reference from the \*.csproj file

<ItemGroup>

<None Remove="appsettings.json.xml" />

</ItemGroup>

<ItemGroup>

<Content Include="appsettings.json.xml">

<CopyToOutputDirectory>Always</CopyToOutputDirectory>

</Content>

</ItemGroup>

## Generated code

No generated code should be in source control. This includes code that is automatically generated by VS19, or by the build process of the particular language itself.

For our purpose, the definition of generated code is anything that is generated and compiled into the project without being touched by people. If you have some code generator that generates some classes for you, then you copy them into the src folder and modify them, those are no longer considered generated code and its appropriate to put them in source control.

## Project Namespaces and Assembly Names

After file, directory, and project names you must go into the code and make certain the namespace of the project matches the folder names. For a long time, people use the suffix “Api” instead of “Svc”. This issue is being corrected now. If the namespace does not match the new directory structure established previously, you must refactor the project. So for example

namespace Exos.VendorAttributesApi

{

public class Startup

{

public Startup(IConfiguration config, IHostingEnvironment env)

{

Configuration = config;

Environment = env;

}

Must be refactored to

namespace Exos.VendorAttributesSvc

{

public class Startup

{

public Startup(IConfiguration config, IHostingEnvironment env)

{

Configuration = config;

Environment = env;

}

## Update .gitignore.

Unless your project has special needs, the .gitignore file in the architects share folder should be perfectly adequate for your project. It was generated by Gitignore.io. If you need to add additional tech to your gitignore, follow the instructions on gitignore.io and copy the entire file into your .gitignore file.

This new ignore file will be updated periodically by an automated process. Take note of the block generated by gitignore.io:

# Created by https://www.gitignore.io/api/node,macos,<bla bla bla>

<Huge block of text>

# End of https://www.gitignore.io/api/node,macos,<bla bla bla>

If you need additional items in your ignore, add them either above or below this block. Automated update systems will search for these two lines and keep the block up to date on periodic basis. That automation system will ignore and preserve items above and below this block. If you modify anything inside the block, it will be overwritten the next time the file is refreshed.

## Update \*.sln file

In the global section of the solution file, if you see any areas for no style, or anything to bypass the style checks, they need to be deleted. So this:

Global

GlobalSection(SolutionConfigurationPlatforms) = preSolution

Debug - No Style|Any CPU = Debug - No Style|Any CPU

Debug|Any CPU = Debug|Any CPU

Release|Any CPU = Release|Any CPU

EndGlobalSection

GlobalSection(ProjectConfigurationPlatforms) = postSolution

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug - No Style|Any CPU.ActiveCfg = Debug - No Style|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug - No Style|Any CPU.Build.0 = Debug - No Style|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug|Any CPU.ActiveCfg = Debug|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug|Any CPU.Build.0 = Debug|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Release|Any CPU.ActiveCfg = Release|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Release|Any CPU.Build.0 = Release|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug - No Style|Any CPU.ActiveCfg = Debug - No Style|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug - No Style|Any CPU.Build.0 = Debug - No Style|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug|Any CPU.ActiveCfg = Debug|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug|Any CPU.Build.0 = Debug|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Release|Any CPU.ActiveCfg = Release|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Release|Any CPU.Build.0 = Release|Any CPU

EndGlobalSection

GlobalSection(SolutionProperties) = preSolution

HideSolutionNode = FALSE

EndGlobalSection

GlobalSection(NestedProjects) = preSolution

{C0D496E0-923D-4AE7-8089-58C97D46E285} = {0644FDB8-DAA7-4E94-96C2-A9F8BC8DA9F8}

{BFE3333F-17F5-47D0-BE89-03C5164A427A} = {14C8B281-AE97-41A5-9775-54DE3C25899D}

{ED774915-E782-4379-8086-4882BE325685} = {21EDBD9D-CE43-4F26-8AFA-9BCE157D40F4}

{AA04CE32-6B5F-4367-81F9-8896A2EF2DCB} = {ED774915-E782-4379-8086-4882BE325685}

EndGlobalSection

GlobalSection(ExtensibilityGlobals) = postSolution

SolutionGuid = {AE07038A-7DFA-4303-859F-9052AF081B1D}

EndGlobalSection

EndGlobal

Becomes this:

Global

GlobalSection(SolutionConfigurationPlatforms) = preSolution

Debug|Any CPU = Debug|Any CPU

Release|Any CPU = Release|Any CPU

EndGlobalSection

GlobalSection(ProjectConfigurationPlatforms) = postSolution

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug|Any CPU.ActiveCfg = Debug|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Debug|Any CPU.Build.0 = Debug|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Release|Any CPU.ActiveCfg = Release|Any CPU

{C0D496E0-923D-4AE7-8089-58C97D46E285}.Release|Any CPU.Build.0 = Release|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug|Any CPU.ActiveCfg = Debug|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Debug|Any CPU.Build.0 = Debug|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Release|Any CPU.ActiveCfg = Release|Any CPU

{BFE3333F-17F5-47D0-BE89-03C5164A427A}.Release|Any CPU.Build.0 = Release|Any CPU

EndGlobalSection

GlobalSection(SolutionProperties) = preSolution

HideSolutionNode = FALSE

EndGlobalSection

GlobalSection(NestedProjects) = preSolution

{C0D496E0-923D-4AE7-8089-58C97D46E285} = {0644FDB8-DAA7-4E94-96C2-A9F8BC8DA9F8}

{BFE3333F-17F5-47D0-BE89-03C5164A427A} = {14C8B281-AE97-41A5-9775-54DE3C25899D}

{ED774915-E782-4379-8086-4882BE325685} = {21EDBD9D-CE43-4F26-8AFA-9BCE157D40F4}

{AA04CE32-6B5F-4367-81F9-8896A2EF2DCB} = {ED774915-E782-4379-8086-4882BE325685}

EndGlobalSection

GlobalSection(ExtensibilityGlobals) = postSolution

SolutionGuid = {AE07038A-7DFA-4303-859F-9052AF081B1D}

EndGlobalSection

EndGlobal

## Update \*.csproj files

Part of this project is to externalize certain project configuration settings to allow more centralized management.

### Directory.Build.props changes

To do this, you must first close VS19. These files are much easier to edit with VS19 closed. Open the Directory.Build.props file in the root of your repo in a text editor. The build system is now applying all the settings in this file to every \*.csproj file in the solution. Now you must cycle thru all the csproj files in your solution, open each one in a text editor. If a tag appears in both the Directory.Build.props file AND your csproj file, delete the tag from your csproj file.

#### FxCop and StyleCop

These settings are now to be completely managed by the Directory.Build.props file. This has major impacts in the FxCop and StyleCop settings. If you have them specified in your csproj, you will get false hits on your scans.

When you encounter that, check your csproj file for things like this:

<PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|AnyCPU'">

<CodeAnalysisRuleSet>..\Complete.ruleset</CodeAnalysisRuleSet>

<TreatWarningsAsErrors>true</TreatWarningsAsErrors>

<WarningsAsErrors />

<NoWarn>1701;1702;1591;CA1303;CA2227;CA1707;CS1591;</NoWarn>

</PropertyGroup>

PropertyGroups like this are managing the scan settings outside of the standard. The standard is now being managed by the Exos.ruleset file in the root of your project, copied over from the architect share. The entire PropertyGroup needs to be removed from your csproj file.

You are not permitted to update the Exos.ruleset file.

If you find other rulesets in your code base, like the “Complete.ruleset” file referenced above example, those files should be deleted. The copied in files from the architect folder overrides everything now.

Under the various “ItemGroup” sections of your csproj file, look for references to FxCop and StyleCop, like this

<PackageReference Include="Swashbuckle.AspNetCore" Version="5.1.0" />

<PackageReference Include="Microsoft.CodeAnalysis.FxCopAnalyzers" Version="2.9.7" />

<PackageReference Include="SecurityCodeScan.VS2017" Version="3.3.0">

The FxCop section must be deleted

<PackageReference Include="Swashbuckle.AspNetCore" Version="5.1.0" />

<PackageReference Include="SecurityCodeScan.VS2017" Version="3.3.0">

### Obsolete Tags

The csproj file now contains many unused config settings that should be deleted now that the structure of the project has been corrected.

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<AssemblyName>Exos.VendorComplianceSvc</AssemblyName>

<RootNamespace>Exos.VendorComplianceSvc</RootNamespace>

<ApplicationInsightsResourceId>/subscriptions/97aeb7a6-ff38-40b1-9fc8-a4111cf894c2/resourcegroups/AZE2-D-EXO-rgMon01/providers/microsoft.insights/components/AZE2-D-EXO-ainApp01</ApplicationInsightsResourceId>

<ApplicationInsightsAnnotationResourceId>/subscriptions/97aeb7a6-ff38-40b1-9fc8-a4111cf894c2/resourcegroups/AZE2-D-EXO-rgMon01/providers/microsoft.insights/components/AZE2-D-EXO-ainApp01</ApplicationInsightsAnnotationResourceId>

<AssemblyVersion>1.0.0.0</AssemblyVersion>

<FileVersion>1.0.0.0</FileVersion>

<Configurations>Debug;Release;Debug - No Style</Configurations>

</PropertyGroup>

</Project>

The highlighted tags should be deleted. Groups like this should be deleted as well

<PropertyGroup>

<DocumentationFile>bin\Release\netcoreapp2.0\Exos.VendorCoverageSvc.xml</DocumentationFile>

</PropertyGroup>

<ItemGroup>

<WCFMetadata Include="Connected Services" />

</ItemGroup>

For these package references, the options for them need deleted

<PackageReference Include="SecurityCodeScan.VS2017" Version="3.3.0">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>

</PackageReference>

<PackageReference Include="AsyncFixer" Version="1.1.6">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>

</PackageReference>

### Test csproj files

In test project csproj files, you must configure them to generate documentation.

<PropertyGroup>

<TargetFramework>netcoreapp3.1</TargetFramework>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<IsPackable>false</IsPackable>

</PropertyGroup>

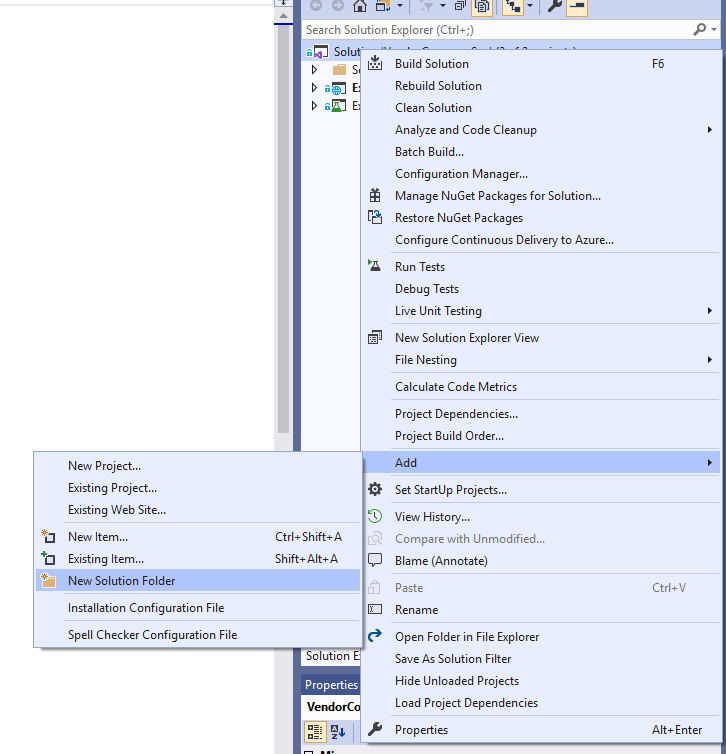
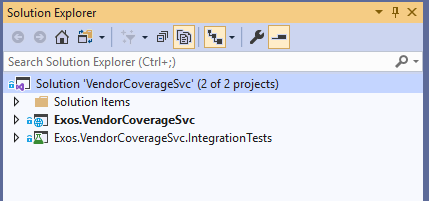
If there is a configurations section, it needs to be deleted.

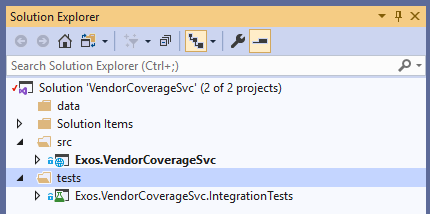
## Solution Folders

At this point you should have a running solution. Once your solution is opened in VS19, you need to organize the projects in the solutions view. So with the solution open and inside solution explorer you should see something like this

You need to organize the solution folders identical to the physical folders.

* So projects need to go into a /src solution folder
* tests need to go into a /tests solution folder
* data projects need to go in a /data folder





## Readme file

In the root of the project should be a README.md file. You must update that file to be accurate. That file should list the following

Each project in the solution and the specific purpose of that project. Things like

Vendor Profile Service – Manages the profiles of the vendor

Will not be accepted. We want details. If a person asked what this service is and why it exists, answer it here. We want the following

* Overall description of what included in the solution.
* Project that’s included in the solution, then for each project
  + Description paragraph of what that project does
  + What databases it connects to, just the list, no other details needed than the list of databases it connects to
  + If it’s a service, the final URI pattern of that service

From now on, verifying the accuracy of the Readme file will be on the merge checklist. A simple cheatsheet for markdown is here

<https://www.markdownguide.org/cheat-sheet/>

## Classes and Names.

As per the standards document

<https://teams.microsoft.com/l/file/C911A252-EF46-4B71-891C-5E105FCC0E0D?tenantId=8a807b9b-02da-47f3-a903-791a42a2285c&fileType=docx&objectUrl=https%3A%2F%2Ffnfms.sharepoint.com%2Fsites%2FGRP-ArchitectureandEngineering%2FShared%20Documents%2FGeneral%2FCoding%20Standards%2FEXOS%20Microservices%20Development%20Standard%20DRAFT-v1.5.docx&baseUrl=https%3A%2F%2Ffnfms.sharepoint.com%2Fsites%2FGRP-ArchitectureandEngineering&serviceName=teams&threadId=19:f3b6a2c8ee11469d93fe37ff1643c368@thread.skype&groupId=bc66c5ca-a58e-4f97-a53e-bc53cf4d3c02>

We are beginning to check class naming and folder structure. The standards document is the final word, but here are what we will be checking

### Class names files, and folders

The class name must match the filename. A class like this:

using Exos.Platform.Persistence.Models;

using System.Collections.Generic;

namespace Exos.VendorCoverageSvc.Models.Coverage

{

public class RegionModel : BaseModel

{

private long \_subContractorId;\*/

public string VendorRegionId { get; set; }

public long VendorId { get; set; }

public long SubContractorId { get; set; }

public string State { get; set; }

public string CountyName { get; set; }

public List<string> ZipCodes { get; set; }

public override string CosmosDocType => "RegionStateCountyZipCodes";

}

}

Must be in a file named RegionModel.cs

In addition, the RegionModel.cs file must be in the folder Exos.VendorCoverageSvc/Models/Coverage. The namespaces must match the folder structure. This rule extends to all classes in your application. Every class in the application will be checked to ensure that these rules are followed.

Last note about this, as per the standard, you can only have one class per file.

### Model class names

All model class names must end with \*Model. In addition, they must all be located in the {project}/models directory.

### Comments

When checking your classes, delete any of these comments

* Todo’s. they were most likely forgotten and outdated anyways
* Commented code. It’s the SCM job to track history, not source code.
* Any “fixed for bug XYZ” type comments. that’s the SCM and Project Managements job to track things like that, not the code.

## Naming final comments

It is not likely you will ever get this much of a free hand opportunity to rename things again. Take a long hard look at each repository and make absolutely sure the names for each project are the names they should have. If you feel that something needs to change, ping the architect for your project and talk about it.

Again, you will not see an opportunity like this again, so take advantage of it now.

# Upgrade to ASP.NET Core 3.1

Change project files to reference netcoreapp3.1 and the Microsoft.Net.Sdk.Web project:

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>netcoreapp3.1</TargetFramework>

</PropertyGroup>

<!-- Truncated -->

</Project>

Remove package references to Microsoft.AspNetCore.\*. In .NET Core 3.1 these are implied by the SDK choice.

Remove references to any Serilog libraries. We are no longer using it in EXOS.

Remove references to Microsoft.ApplicationInsights.AspNetCore. This has been upgraded in the platform libraries.

Updated to the latest versions of EXOS platform libraries.

<ItemGroup>

<PackageReference Include="Exos.AspNetCore.Platform" Version="TBD" />

<PackageReference Include="Exos.Platform.DistributedCache" Version="TBD" />

<PackageReference Include="IdentityModel" Version="3.9.0" />

<PackageReference Include="IdentityServer4.AccessTokenValidation" Version="2.6.0" />

<PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.4.1" />

<PackageReference Include="Microsoft.AspNetCore.App" Version="2.2.0" />

<PackageReference Include="Microsoft.AspNetCore.WebSockets" Version="2.2.0" />

<PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.13.1" />

<PackageReference Include="Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version="3.1.0" />

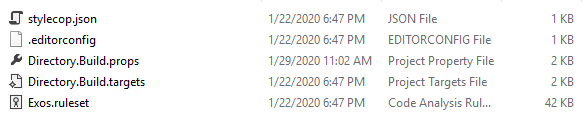
<PackageReference Include="Microsoft.Extensions.Caching.StackExchangeRedis" Version="2.2.0" />

<PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.3" />

<PackageReference Include="Serilog.Sinks.ApplicationInsights" Version="2.6.0" />

</ItemGroup>

# Enabling Code any Style Analysis



Projects (including test projects) should set the following properties:

<PropertyGroup>

<TargetFramework>netcoreapp3.1</TargetFramework>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<Description>Manages Vendor Profile resources.</Description>

</PropertyGroup>

<PropertyGroup>

<NoWarn>1701;1702;1591</NoWarn>

</PropertyGroup>

The Directory.Build.props file in the root of the repo contains the actual service version and references to Code Analysis and Style Analysis libraries. This saves you having to put them in each of your projects. Standard company information has already been placed in the file, but developers are responsible for maintaining the version number following SemVer:

<PropertyGroup>

<Version>1.0.0</Version>

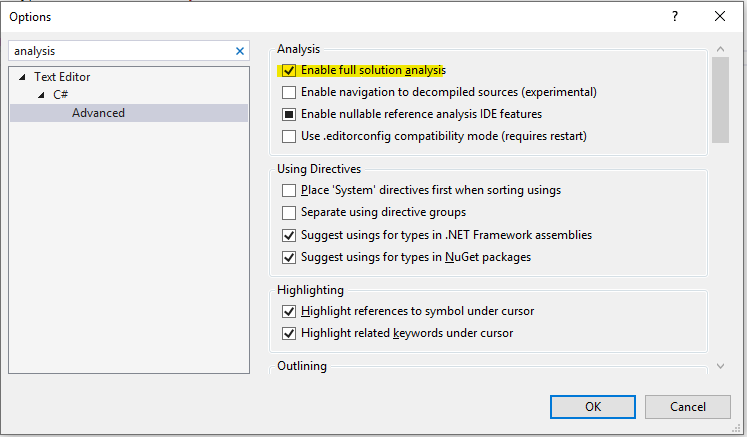
<Product>EXOS</Product>

<Company>ServiceLink IP Holding Company, LLC</Company>

<!-- Truncated -->

</PropertyGroup>

Enabled Full Solution Analysis for C# files in Visual Studio:



**Proceed to fix all the project warnings. Do not change or modify the Exos.ruleset file. Do not disable warnings without speaking to an Architect or Lead Engineer.**

# Wrap any Synchronous Calls with Retry

# Separate Borrower Information and Encrypt

Once the fields required for encryption, the following process is required to enable encryption/decryption.

Decorate each field in models and entities with the Encrypted attribute. See example

[Encrypted]

public string BorrowerName { get; set; }

In Startup.cs initialize the following services:

// Encryption

services.AddScoped<IDatabaseEncryption, AesGcmDatabaseEncryption>();

services.Configure<ClientEncryptionKeyOptions>(\_configuration .GetSection("ClientEncryptionKeyOptions"));

services.AddScoped<IEncryptionKeyFinder, AppSettingsEncryptionKeyFinder>();

Encryption is implementing following this pattern:

1. IDatabaseEncryption *interface, the interface will be used to implement the encryption/decryption methods*
2. AesGcmDatabaseEncryption *class that implements* IDatabaseEncryption *use the AES-GCM algorithm for encryption.*
3. IEncryptionKeyFinder *interface, this interface will be used to implement how to read the keys, keys can be read from settings file or directly from Key Vault.*
4. AppSettingsEncryptionKeyFinder *class that implements* IEncryptionKeyFinder, *this class finds the key values in the appsettings.json file.*
5. ClientEncryptionKey *and* ClientEncryptionKeyOptions *classes are used to read the keys from the settings file used in the* AppSettingsEncryptionKeyFinder *class.*

Additional Changes

In each DBContext inject the IDatabaseEncryption instance and pass it to the PlatformDbContext, see below example

public MultiTenantDBContext(

ILogger<MultiTenantDBContext> logger,

DbContextOptions<MultiTenantDBContext> dbcontextOptions,

IUserHttpContextAccessorService userHttpContextAccessorService,

IPolicyHelper policyHelper,

IPolicyContext policyContext,

IServiceProvider serviceProvider,

**IDatabaseEncryption** **databaseEncryption**) : base(

dbcontextOptions,

userHttpContextAccessorService,

logger,

policyHelper,

policyContext,

serviceProvider,

**databaseEncryption**)

{

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

\_userHttpContextAccessorService = userHttpContextAccessorService ??

throw new ArgumentNullException(nameof(userHttpContextAccessorService));

\_policyHelper = policyHelper ?? throw new ArgumentNullException(nameof(policyHelper));

\_policyContext = policyContext ?? throw new ArgumentNullException(nameof(policyContext));

\_serviceProvider = serviceProvider ?? throw new ArgumentNullException(nameof(serviceProvider));

\_**databaseEncryption** = **databaseEncryption** ?? throw new ArgumentNullException(nameof(databaseEncryption));

}

SQL Repositories:

1. If Dapper queries are executed inject the IDatabaseEncryption instance to have access to the Dapper Extensions.
2. To Execute a Dapper query with encrypted data pass the instance of IDatabaseEncryption to the extension class see below example:

var queryResult = await dbconnection.QueryAsync<TenantSqlModel>(tenantQuery, \_**databaseEncryption**, \_logger).ConfigureAwait(false);

1. No change required in create and update methods.

Cosmos Repositories

1. Inject the IDatabaseEncryption instance to have access to the encryption methods
2. Decorate with the attribute Encrypted the fields that needs encryption in the document model.

/// <summary>

/// Gets or sets the SSN.

/// </summary>

[Encrypted]

public string SSN { get; set; }

1. For the creation of documents the model should be encrypted before save and decrypted and before sending the results back see the following example

**TenantCosmosModel encryptedTenantCosmosModel = \_databaseEncryption.Encrypt<TenantCosmosModel>(tenantCosmosModel);**

document = await \_persistenceService.CreateDocumentAsync(encryptedTenantCosmosModel).ConfigureAwait(false);

var savedDocument = (TenantCosmosModel)(dynamic)document.Resource;

**TenantCosmosModel decryptedTenantCosmosModel = \_databaseEncryption.Decrypt<TenantCosmosModel>(savedDocument);**

return decryptedTenantCosmosModel;

1. To update documents the same operations needs to be executed, below example

// Encrypt before saving

**TenantCosmosModel encryptedTenantCosmosModel = \_databaseEncryption.Encrypt<TenantCosmosModel>(tenantCosmosModel);**

ResourceResponse<Document> updatedDocument = await \_persistenceService.ReplaceDocumentAsync(encryptedTenantCosmosModel).ConfigureAwait(false);

encryptedTenantCosmosModel = (TenantCosmosModel)(dynamic)updatedDocument.Resource;

// Decrypt before return.

**TenantCosmosModel decryptedTenantCosmosModel = \_databaseEncryption.Decrypt<TenantCosmosModel>(encryptedTenantCosmosModel);**

return encryptedTenantCosmosModel;

1. Same concept for queries, results needs to be decrypted before return

PlatformResponse<TenantCosmosModel> platformResponse = await \_persistenceService.ExecuteNextAsync<TenantCosmosModel, TenantCosmosModel>(stringWriter, sqlParameterCollection, "v.", null, queryOptions).ConfigureAwait(false);

var toReturn = platformResponse.List.FirstOrDefault();

**TenantCosmosModel decryptedTenantCosmosModel = \_databaseEncryption.Decrypt<TenantCosmosModel>(toReturn);**

return decryptedTenantCosmosModel;

1. appsettings.json file

Current implementation will find the key names / values from the setting file this will be changed to use key vault instead add the following entries in your settings file for testing

"ClientEncryptionKeyOptions": {

"ClientEncryptionKeys": [

{

"ClientKeyName": "WELLS",

"ClientKeyValue": "y$B&E)H@McQfTjWmZq4t7w!z%C\*F-JaN"

},

{

"ClientKeyName": "BOFA",

"ClientKeyValue": "5v8y/B?E(H+MbQeThWmYq3t6w9z$C&F)"

}

]

}

This configuration represent different client configurations, current implementation will also assume that there is a Claim called clientresource in the jwt authentication that will have the client key name based on that value the key value fill be found, this will change as well, when the final implementation is completed we will need to update the IEncryptionKeyFinder implementation and remove the ClientEncryptionKeyOptions configuration.

# Replace Synchronous Updates/Inserts/Deletes with Async ServiceBus Messages

# Add ServiceBus Message Listeners

# Fix Logging and Instrumentation

Any reference to previous Serilog needs to be removed, add the following Logging settings in the appsettings.json file under the .deploy directory and in the appsettings.json.xml

"Logging": {

"LogLevel": {

"Default": "${Serilog.Default}",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"ApplicationInsights": {

"InstrumentationKey": "#{ApplicationInsights:InstrumentationKey}"

},

In your local appsettings.Development.json add the following settings:

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft": "Warning",

      "Microsoft.Hosting.Lifetime": "Information"

    },

    "ApplicationInsights": {

      "LogLevel": {

        "Default": "Information",

        "Microsoft": "Warning",

        "Microsoft.Hosting.Lifetime": "Information"

      }

    }

  },

With these settings you will have traces with level info and above in appinsights for your local development, if the ApplicationInsights section is not included by default only warning and above traces are sent to appinsights.

# Create Unit Tests