Configuring source code delivery for .NET for CAST Imaging

- Prepare the application source code
- What about delivering framework/external/custom assemblies?
  - .NET Framework assemblies
  - External assemblies (third-party DLLs)/custom assemblies
    - Default .NET Assemblies package created on source code root in AIP Console 1.26
- What about Nuget package dependencies?

Prepare the application source code

AIP Console expects the application source code to be delivered either via a **ZIP file** or via a **source code located in a folder** configured in AIP Console. Whichever option you chose, you should include in the ZIP/source code folder all of your .NET application source code. CAST highly recommends placing all the relevant files in a folder and using sub-folders where necessary. You can deliver other technologies at the same time (for example, database DDL). If you are using a ZIP/archive file, zip the folders in the "temp" folder as shown in the image below - but do not zip the "temp" folder itself, nor create any intermediary folders:

```
D:\temp
        |-----DotNET
        |-----OtherTechno1
        |-----OtherTechno2
```

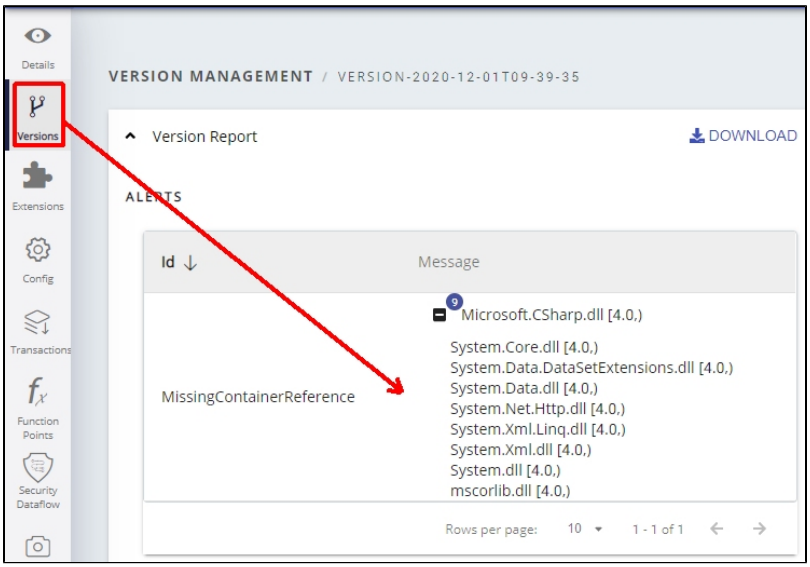
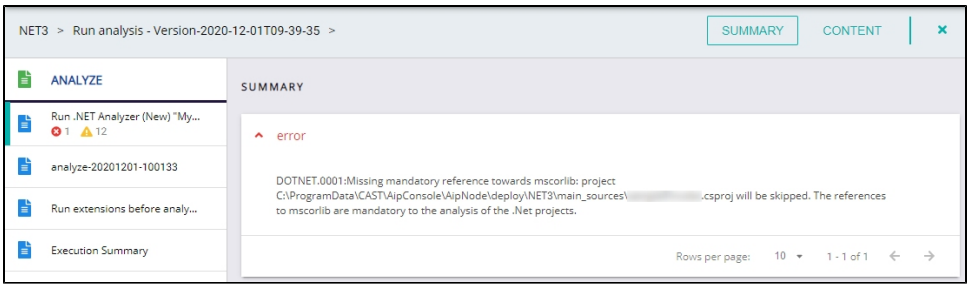
Any additional framework specific source code (such as **Entity Framework**, **Silverlight Framework**, **WCF**, **WPF**, **NoSQL** should also be provided in the ZIP/archive file or the source code folder.

What about delivering framework/external/custom assemblies?

The .NET Analyzer needs to know the location of any **assemblies** such as the **.NET Framework assemblies, external assemblies (third-party DLLs)** and other **custom assemblies** that are used by your application source code. There are various ways to declare the location of these assemblies when using AIP Console, but the action you choose also depends on the release of the .NET Analyzer you are using. This is explained in more detail below:

.NET Framework assemblies

.NET Framework assemblies are used by all .NET applications and therefore the .NET Analyzer needs to have access to these assemblies in order to resolve references correctly. Here there is a choice of options:

Using .NET Analyzer 1.1	When using .NET Analyzer **1.1**, the .NET Framework assemblies are provided in the extension itself (as listed in the section **Dependent fra extension** in the **extension documentation**) and they will be used to resolve references to specific assemblies that have been used by the declare the location of these .NET Framework assemblies. **.NET Analyzer 1.4.3** In these releases, a file called **ressources.json** is provided in the root of the extension which lists all .NET Framework assemblies shipped the source code delivery process and any **"missing references" type** alerts related to .NET Framework assemblies shipped with the exter **.NET Analyzer 1.1.0 - 1.4.2** In these releases, **"missing references" type alerts** will be generated during the source code delivery process for any .NET Framework as the extension, for example as shown below. These alerts can be **safely ignored**. The resulting analysis will use the .NET Framework assem *Click to enlarge*  If you prefer not to have missing references type alerts, then you can, optionally, declare the location of the .NET Framework assemblies us Node responsible for analyzing your application, ensuring that the AIP Node can access the declared location (i.e. a folder on the AIP Node used **in priority** over the .NET Framework assemblies provided in the extension itself.
Using .NET Analyzer 1.0	When using .NET Analyzer **1.0**, the .NET Framework assemblies are NOT provided in the extension, therefore you MUST declare the locati `node-app.properties` file, ensuring that the AIP Node responsible for analyzing your application can access the declared location (i.e. a you do not declare them, then the analysis will fail with a **missing mandatory reference error**: 

External assemblies (third-party DLLs)/custom assemblies

> **When using .NET Analyzer 1.1**
>
> CAST provides some specific frameworks and third party packages in the extension itself which will automatically be used. Therefore if your source code uses these specific frameworks and third party packages, it is not necessary to deliver these items. However, **missing library /assembly alerts** for these items will be generated during the delivery. The alerts can be safely ignored if the alert references an item that CAST provides in the extension. See the section **Dependent frameworks and third-party packages provided in the extension documentation** for more information.

If your application uses external assemblies provided by third parties or your own custom assemblies, the .NET Analyzer also needs to know the location of them:

- if these assemblies **are stored in the correct location as specified in the project definition file** and are delivered with the application source code (in the **ZIP file** or via a **source code located in a folder**), then the .NET Analyzer will find them during the analysis without you needing to do anything.

> If there are projects that use "HintPath" to reference assemblies compiled from other projects within the application, then it is advised to also deliver them in the correct location (i.e. as specified in the project definition file). They will be used where circular references of assemblies exist (e.g: project A depends on B.dll and project B depends on A.dll).

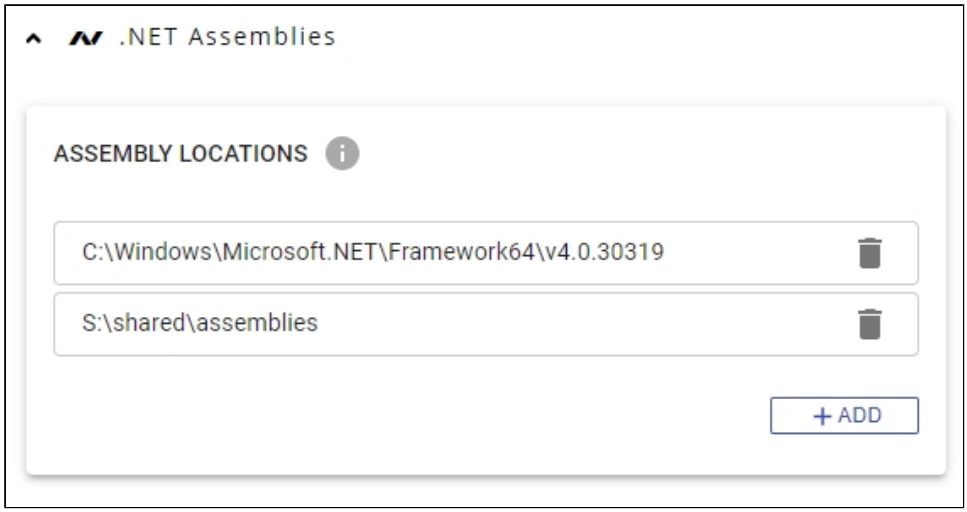
- if these assemblies **are not stored in the correct location**, then there are two options available to you to ensure that the .NET Analyzer is aware of their location:

Declare the location to AIP Console

This option involves:

- placing the assemblies in a folder that the AIP Node responsible for analyzing your application can access - a shared network folder is preferred.
- then **declaring** this folder using either:

- the **AIP Console UI** (available only in **AIP Console 1.26**, see **Administration Center - Settings - .NET Assemblies**) - this is valid for all AIP Nodes:



- or using the `aip-node-app.properties` file (located in `%PROGRAMDATA%\CAST\AipConsole\AipNode\aip-node-app.properties`) on the **AIP Node** (path must use **single forward slashes** or **double back slashes** - the single back slash is not valid and multiple paths must be separated by **semi-colons)**:

```
#scanner.dotnet.assembly.locations=C:/dotNet/v4.0;C:/dotNet/v4.5;C:/dotNet/v3.5
scanner.dotnet.assembly.locations=
```

When the analysis is run, the .NET Analyzer will search the folders you declared and find the assemblies required by your application source code.

Deliver with the application source code

> This option has been removed in **AIP Console 1.26**.

This option involves:

- delivering the assemblies in a dedicated folder (typically this is a sub folder or the "bin" folder) together with the application source code (i.e. in the **ZIP file** or via a **source code located in a folder**).
- ensuring that the `scanner.detect.dotnet.assemblies` option is set to **true** the `aip-node-app.properties` file on the AIP Node.

When the analysis is run, the .NET Analyzer will attempt to find the assemblies required by your application source code.

> This option is **limited in scope** however and if in doubt, you should use the alternative option described above (**Declare the location in aip-node-app.properties**): only one folder in the delivered source code will be detected by the .NET Analyzer - the folder containing the largest number of assemblies (dll files). Therefore if your assemblies are distributed in multiple folders throughout the source code, this method is not recommended.

Default .NET Assemblies package created on source code root in AIP Console 1.26

In **AIP Console 1.26** a change has been made: by default, a .**NET Assemblies package will always be automatically created on the application source code root folder**. This is to ensure that any assemblies **delivered with the application source code** (in the **ZIP file** or via a **source code located in a folder**) but not stored in the correct location as specified in the project definition file, are always taken into account - a kind of fail safe mechanism. This mechanism is governed by the following option in the the `aip-node-app.properties` file (located in `%PROGRAMDATA%\CAST\AipConsole\AipNode\aip-node-app.properties`):

```
# Option is defaulted to true
scanner.dotnet.assembly.create.from.root=true
```

If you do not want the default .NET Assemblies package to be created, you can change the option to **false** (restart the AIP Node so that the changes are taken into account).

What about Nuget package dependencies?

An extension (**NuGet Resources Extractor**) has been published that provides the means to configure an automatic extraction of **NuGet package dependencies** from a NuGet repository specifically for .NET application source code. In other words, NuGet package based source code that resides in a simple local or **nuget.org** location. For example, when your .NET application contains **.csproj files** which have package references defined, you can use this extractor to extract those NuGet packages from the NuGet repository.

Out of the box, if a **.csproj** file is detected in the delivered source code, the extension will be downloaded and installed as part of the analysis process. If the .csproj file contains **package dependency references**, these references will automatically be accessed and included in the analysis. Example package references shown below:

```
<ItemGroup>
    <!-- ... -->
    <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0" />
    <!-- ... -->
</ItemGroup>
```

The extractor will extract all NuGet package dependencies and place them inside a folder called "**nugetPck**" folder located in the Deploy folder.
The extractor is driven by the `%PROGRAMDATA%\CAST\AipConsole\AipNode\aip-node-app.properties` file attribute `scanner.nuget.repository`:

```
# HTTP V3 Nuget repository to download package dependencies https://api.nuget.org/v3/index.json or file system
like file://C:/Users/johndoe/.nuget/packages
scanner.nuget.repository=https://api.nuget.org/v3/index.json
```