Using SmartAssembly with single-file executables (.NET Core 3)

Page last updated 01 December 2020

.NET Core 3.0 introduced the ability to create single-file executables. This allows for distribution of only one application file, as all configs and dependencies are included within the binary itself.

The feature provides a native way for dependencies embedding which is most beneficial when publishing self-contained applications generating hundreds of assemblies. It can be used for framework-dependent or self-contained applications, but requires setting a runtime identifier in both cases to target a specific environment and bitness.

> You can download a working example of protecting a single-file .NET Core application with SmartAssembly.
>
> See our GitHub repository: https://github.com/red-gate/SmartAssembly-demos/tree/master/msbuild-integration-demos/netcore3-single-file

Getting started

First, let's see how a regular process of publishing a .NET Core application looks like.

Suppose we have a .NET Core 3.1 application called *ConsoleApp*, defined by the following project file:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
</Project>
```

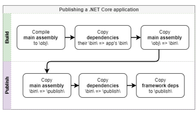
To publish this application we would use the following command:

```
dotnet publish .\ConsoleApp.csproj -c Release -r win-x64
```

The command above will **build** the application in "Release" mode, and then **publish** as self-contained and targeting Windows 64-bit systems. All dependencies (including .NET Core framework assemblies) and configs will be copied to the published directory.

After the publishing is done, go to the output directory (this would be *\ConsoleApp\bin\Release\netcoreapp3.1\win-x64\publish* for the app above). Take notice of how many files are present in the directory.

The process of publishing the self-contained .NET Core application can be visualized as follows:



Using single-file executable

Now let's publish our application as single-file. All we need to do is add a **PublishSingleFile** property with value set to **true**.

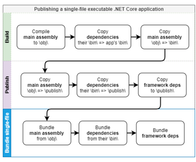
```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <PublishSingleFile>true</PublishSingleFile>       <!-- add this line -->
  </PropertyGroup>
</Project>
```

Let's issue the publish command again:

```
dotnet publish .\ConsoleApp.csproj -c Release -r win-x64
```

After the process is done, navigate to the published directory (*\ConsoleApp\bin\Release\netcoreapp3.1\win-x64\publish* for the app above). You should only see 1 .exe file and optionally a .pdb file.

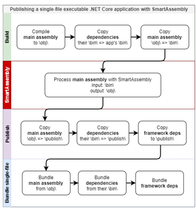
Let's see how the publishing process has changed after enabling single-file:



Integrating SmartAssembly

For the final step let's protect our assembly before it's bundled into single-file executable.

SmartAssembly will be executed after your application is built, but before publishing and bundling to single-file. This is shown on the diagram below:



Step 1: Creating a SmartAssembly project

1. Open SmartAssembly.
2. Click **New Project**.
3. Click **Browse Assembly**.
4. Navigate to your application's output directory, choose the appropriate assembly (for the app above it'd be: *\ConsoleApp\bin\Release\netcoreapp3.1\win-x64\ConsoleApp.dll*) and **Open**.
5. Click **Set Destination**.
6. Navigate to your application's \obj\ directory (for the app above it'd be: *\ConsoleApp\obj\Release\netcoreapp3.1\win-x64\ConsoleApp.dll*) and **Save**. If the file already exists click **Yes** to overwrite.
7. Configure the project as needed. Enable any protections your application may need.
8. Click **Save** to save the project. It's recommended to save the project in the same location as Visual Studio's project file, under the same name (for the app above it'd be: *\ConsoleApp\ConsoleApp.saproj*).

> "Dependencies Merging" and "Dependencies Embedding" should not be used when protecting a single-file assembly. Merging or embedding assemblies can result in unexpected behavior and isn't recommended for this type of applications.

Step 2: Integrating SmartAssembly into the publish process

1. Open your application's project in Visual Studio.
2. **Right-click** the project name and select **Manage NuGet packages...**
3. Switch to the **Browse** tab.
4. Type *RedGate.SmartAssembly.MSBuild* and click **Install** next to it.

That's it! After performing the steps above, your project file should look like the following:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <PublishSingleFile>true</PublishSingleFile>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="RedGate.SmartAssembly.MSBuild" Version="7.4.0.3402">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</
    </PackageReference>
  </ItemGroup>
</Project>
```

Now let's issue the **publish** command once again:

```
dotnet publish .\ConsoleApp.csproj -c Release -r win-x64
```

> If you're running multiple consecutive builds of the same project you should clean the project before or after each build. Because SmartAssembly obfuscates the assembly into \obj\ directory, the obfuscated version might be picked up by the build command next time leading to double-obfuscation and other unexpected results.
>
> To clean the project either manually remove the \obj\ directory or issue the following command (using the same configuration and runtime identifier used for publish):
>
> ```
> dotnet clean .\ConsoleApp.csproj -c Release -r win-x64
> ```

If everything went well, you should see additional output messages in the command line confirming that SmartAssembly has successfully protected your application (paths were shortened for clarity):

```
  Microsoft (R) Build Engine version 16.5.0+d4cbfca49 for .NET Core
  Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 206.38 ms for C:\[..]\ConsoleApp\ConsoleApp.csproj.
  ConsoleApp -> C:\[..]\ConsoleApp\bin\Release\netcoreapp3.1\win-
  x64\ConsoleApp.dll
  Executing SmartAssembly from: C:\PROGRA~1\Red Gate\SmartAssembly
  7\SmartAssembly.com
  Using project: C:\[..]\ConsoleApp\ConsoleApp.saproj
  SmartAssembly v7.4.0.3402 Personal
  Copyright c Red Gate Software Ltd 2005-2020
  Loading project C:\[..]\ConsoleApp\ConsoleApp.saproj
  Input=C:\[..]\ConsoleApp\bin\Release\netcoreapp3.1\win-x64\ConsoleApp.dll
  Loading...
  Starting...
  Analyzing...
  Preparing...
  Creating assembly...
  Copying additional files...
  OK

  ConsoleApp -> C:\[..]\ConsoleApp\bin\Release\netcoreapp3.1\win-x64\publish\
```