Creating a New Project

Creating a new WinRT Component DLL and referencing it in another project

When creating a new DLL, it was really helpful to reference an existing DLL's `.vcxproj` like `TerminalControl.vcxproj`. While you should mostly try to copy what the existing `.vcxproj` has, here's a handful of things to double check for as you go along.

- ☐ Make sure to `<Import>` our pre props at the *top* of the vcxproj, and our post props at the *bottom* of the vcxproj.

```
<!-- pre props -->
<Import Project="..\..\..\common.openconsole.props" Condition="'$(OpenConsoleDir)'==''" />
<Import Project="$(OpenConsoleDir)src\cppwinrt.build.pre.props" />

<!-- everything else -->

<!-- post props -->
<Import Project="$(OpenConsoleDir)src\cppwinrt.build.post.props" />
```

- ☐ Add a `<ProjectReference>` to your new `.vcxproj` in both `WindowsTerminal.vcxproj` and `TerminalApp.vcxproj`
- ☐ Add a `<Reference>` to `TerminalAppLib.vcxproj` similar to this:

```
<Reference Include="Microsoft.Terminal.NewDLL">
  <HintPath>$(OpenConsoleCommonOutDir)\TerminalNewDLL\Microsoft.Terminal.NewDLL.winmd</H
  <IsWinMDFile>true</IsWinMDFile>
  <Private>false</Private>
  <CopyLocalSatelliteAssemblies>false</CopyLocalSatelliteAssemblies>
</Reference>
```

- ☐ Make sure the project has a `.def` file with the following lines. The `WINRT_GetActivationFactory` part is important to expose the new DLL's activation factory so that other projects can successfully call the DLL's `GetActivationFactory` to get the DLL's classes.

```
EXPORTS
DllCanUnloadNow = WINRT_CanUnloadNow                    PRIVATE
DllGetActivationFactory = WINRT_GetActivationFactory    PRIVATE
```

- For a bit more context on this whole process, the `AppXManifest.xml` file defines which classes belong to which DLLs. If your project wants class `X.Y.Z`, it can look it up in the manifest's definitions and see that it came from `X.Y.dll`. Then it'll load up the DLL, and call a particular function called `GetActivationFactory(L"X.Y.Z")` to get the class it wants. So, the definitions in `AppXManifest` are *required* for this activation to work properly, and I found myself double checking the file to see that the

definitions I expect are there.
- *Note*: If your new library eventually rolls up as a reference to our Centennial Packaging project `CascadiaPackage`, you don't have to worry about manually adding your definitions to the `AppXManifest.xml` because the Centennial Packaging project automatically enumerates the reference tree of WinMDs and stitches that information into the `AppXManifest.xml`. However, if your new project does *not* ultimately roll up to a packaging project that will automatically put the references into `AppXManifest`, you will have to add them in manually.

Troubleshooting

- If you hit an error that looks like this: `X found processing metadata file ..\blah1\Microsoft.UI.Xaml.winmd, type already exists in file ..\blah\NewDLLProject\Microsoft.UI.Xaml.winmd.` The `Microsoft.UI.Xaml.winmd` is showing up in the output folder when it shouldn't. Try adding this block at the top of your `.vcxproj` `<ItemDefinitionGroup> <Reference> <Private>false</Private> </Reference> </ItemDefinitionGroup>` This will make all references non-private, meaning "don't copy it into my folder" by default.
- If you hit a `Class not Registered` error, this might be because a class isn't getting registered in the app manifest. You can go check `src/cascadia/CascadiaPackage/bin/x64/Debug/AppX/AppXManifest.xml` to see if there exist entries to the classes of your newly created DLL. If the references aren't there, double check that you've added `<ProjectReference>` blocks to both `WindowsTerminal.vcxproj` and `TerminalApp.vcxproj`.
- If you hit an extremely vague error along the lines of `Error in the DLL`, and right before that line you notice that your new DLL is loaded and unloaded right after each other, double check that your new DLL's definitions show up in the `AppXManifest.xml` file. If your new DLL is included as a reference to a project that rolls up to `CascadiaPackage`, double check that you've created a `.def` file for the project. Otherwise if your new project *does not* roll up to a package that populates the `AppXManifest` references for you, you'll have to add those references yourself.