ChatGPT

Migration Guide: BearGIS to .NET 7 and .NET Framework 4.8

This guide walks through updating the BearGIS Grasshopper plugin (originally .NET Framework 4.6.1 targeting Rhino 5) to a modern multi-target project that supports both Rhino 7 (.NET 4.8 runtime) and Rhino 8 (.NET 7+ runtime). We will cover project file changes, third-party library updates, Rhino runtime debugging, NuGet package usage for RhinoCommon/Grasshopper, code compatibility adjustments, and testing for each target environment.

1. Convert to SDK-Style Project and Multi-Targeting

Begin by converting **BearGIS.csproj** to the SDK-style format with multi-targeting. This involves removing old Visual Studio specific settings (e.g. `<ToolsVersion>` and explicit `<Reference>` entries) and using the **.NET SDK** project style. In the new csproj, define **two target frameworks**: .NET 7.0 (for Rhino 8) and .NET Framework 4.8 (for Rhino 7). For example:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>net7.0-windows;net48</TargetFrameworks>
    <OutputType>Library</OutputType>
    <RootNamespace>BearGIS</RootNamespace>
    <AssemblyName>BearGIS</AssemblyName>
    <Platforms>AnyCPU</Platforms>
    <!-- Include Windows Forms support since plugin uses WinForms-based
libraries -->
    <UseWindowsForms>true</UseWindowsForms>
  </PropertyGroup>

  <!-- Conditional debug start settings for Rhino 7 and Rhino 8 -->
  <PropertyGroup Condition="'$(TargetFramework)'=='net48'">
    <StartProgram>C:\Program Files\Rhino 7\System\Rhino.exe</StartProgram>
  </PropertyGroup>
  <PropertyGroup Condition="'$(TargetFramework)'=='net7.0-windows'">
    <StartProgram>C:\Program Files\Rhino 8\System\Rhino.exe</StartProgram>
  </PropertyGroup>

  <!-- AfterBuild target to copy output DLL to .gha (Grasshopper assembly) -->
  <Target Name="AfterBuild" AfterTargets="Build">
    <Copy SourceFiles="$(TargetPath)" DestinationFiles="$(OutputPath)$
(AssemblyName).gha" />
  </Target>
```

```
  ...
</Project>
```

Key points: We use `TargetFrameworks` to list both `net7.0-windows` and `net48`. The `-windows` qualifier is included for .NET 7 because BearGIS relies on Windows-only APIs (WinForms, GDI etc.). We also set `<UseWindowsForms>true</UseWindowsForms>` so that System.Windows.Forms and related assemblies are available in the .NET 7 build. The `<Platforms>AnyCPU</Platforms>` and omission of `<Prefer32Bit>` ensure the plugin builds as AnyCPU – Rhino 8 can run on ARM64 or x64, so an AnyCPU build will work on both architectures (Rhino will load it in the appropriate mode) [1] [2].

The new project eliminates the old `packages.config`; instead, we will use **PackageReference** entries for NuGet dependencies (covered in the next section). All old `<Reference>` elements pointing to hard-coded DLL paths (e.g. RhinoCommon/Grasshopper from Rhino 5 installation) should be removed. We will use NuGet packages for RhinoCommon and Grasshopper instead.

Finally, we added an **AfterBuild** step to handle Grasshopper plugin output. In the original .csproj, a post-build event copied the DLL to a `.gha` file (and deleted the .dll) [3]. In the new SDK-style project, we replicate this by copying `$(TargetPath)` to `$(AssemblyName).gha`. This will produce `BearGIS.gha` alongside the compiled DLL for each target framework. Each framework will output to its own build folder (e.g. *bin\Debug\net48\BearGIS.gha* and *bin\Debug\net7.0-windows\BearGIS.gha*), so there's no conflict. You can adjust or remove the copy step as needed (e.g. if you prefer to manually rename or if using a packaging tool).

Multi-targeting benefits: With this setup, you can build the project for both .NET 4.8 and .NET 7.0 simultaneously. This is the recommended approach for Rhino 8 migration – "multi-target your plugin(s) for .NET 4.8 and .NET 7.0 so that it can run in either runtime on Windows" [4]. Multi-targeting ensures any code incompatibilities are caught at compile time for the respective frameworks [5].

2. Updating Third-Party Dependencies (DotSpatial & Harlow)

BearGIS depends on DotSpatial (GIS libraries) and Harlow (shapefile-to-GeoJSON utility). These need adjustments to support both .NET Framework and .NET 7:

- **DotSpatial** – The project originally used DotSpatial v1.9 (via multiple assemblies like DotSpatial.Controls, DotSpatial.Data, etc.) targeting .NET 4.x. DotSpatial has since evolved: **DotSpatial 3.0** was the last version for .NET Framework (targeting 4.7.2), and **DotSpatial 4.0** targets .NET 6/7 (as "DotSpatial is written for .NET Framework (V1–V3) and .NET Core (V4+)" [6]). Crucially, DotSpatial 3+ replaced its internal geometry library with **NetTopologySuite (NTS)** [7]. This means classes in the `DotSpatial.Topology` namespace were removed in favor of NTS types. We will need to update both the **package versions** and some **code** accordingly.

NuGet Packages: Use **conditional PackageReference** to pull in DotSpatial 3.0.1 for the .NET 4.8 build and DotSpatial 4.0.656 for the .NET 7 build. For example in the .csproj:

```
<ItemGroup Condition="'$(TargetFramework)'=='net48'">
  <PackageReference Include="DotSpatial.Controls" Version="3.0.1" />
  <PackageReference Include="DotSpatial.Data" Version="3.0.1" />
  <PackageReference Include="DotSpatial.Data.Forms" Version="3.0.1" />
  <!-- (include other DotSpatial.* refs as needed, matching 3.0.1) -->
</ItemGroup>
<ItemGroup Condition="'$(TargetFramework)'=='net7.0-windows'">
  <PackageReference Include="DotSpatial.Controls" Version="4.0.656" />
  <PackageReference Include="DotSpatial.Data" Version="4.0.656" />
  <PackageReference Include="DotSpatial.Data.Forms" Version="4.0.656" />
  <!-- ... other DotSpatial.* packages at 4.0.656 ... -->
  <!-- DotSpatial 4 brings in NetTopologySuite as a dependency -->
</ItemGroup>
```

DotSpatial's NuGet packages for v4 target .NET 6.0 (with `net6.0-windows7.0` TFM) but are **compatible with .NET 7.0** [8]. In other words, referencing v4.0.656 in a net7.0-windows project will work. DotSpatial 3.0.1 targets .NET Framework (4.7.2 and above), which is compatible with net48. By using matching versions across all DotSpatial packages per framework, you shouldn't need binding redirects (the new PackageReference model will resolve proper versions).

Code Changes for DotSpatial (NTS integration): Because DotSpatial 3+ uses NetTopologySuite internally, you may need to update how geometry data is accessed: - The original code uses `DotSpatial.Data.Shapefile.OpenFile(...)` to get a shapefile, then iterates `featureSet.Features`. In BearGIS's `ReadDotShp` component, for example, it accessed coordinates via `currentFeature.GetBasicGeometryN(i).Coordinates` which returned `DotSpatial.Topology.Coordinate` objects [9]. In DotSpatial 3/4, `GetBasicGeometryN` may return an NTS geometry (e.g. `NetTopologySuite.Geometries.Geometry`) or similar. **Replace** such usages with NTS equivalents. For instance, you can get the NTS geometry via `currentFeature.Geometry` (if provided) or cast to NTS types. NTS's `Geometry` class provides a `Coordinates` array as well (of type `NetTopologySuite.Geometries.Coordinate`), so you might change the loop to use `NetTopologySuite.Geometries.Coordinate` instead of the old `DotSpatial.Topology.Coordinate`. Because NTS uses double-precision coordinates, the logic of converting to `Rhino.Geometry.Point3d` remains the same. - If using any `DotSpatial.Topology` types (like `DotSpatial.Topology.Point` or `Envelope`), switch to their NTS counterparts (`NetTopologySuite.Geometries.Point`, etc.). The DotSpatial release notes confirm the switch: *"Switched from DotSpatial.NetTopologySuite/DotSpatial.GeoAPI to NetTopologySuite 2.4.0"* [10]. You may need to import NTS namespaces (`using NetTopologySuite.Geometries;`) and adjust method calls. In many cases, the DotSpatial API remains similar but returns NTS interfaces.

Overall, updating to DotSpatial 3/4 allows BearGIS to work in .NET 7, but it requires ensuring your code interacts with the new geometry types. Test the shapefile import/export functions thoroughly in both targets after updating.

- **Harlow** – Harlow (v0.6.0) is a small library used by BearGIS's `ReadShp` component to convert shapefiles to GeoJSON. However, Harlow only targets **.NET Framework 4.5** [11] [12], and no newer or .NET Standard/Core version is available (it hasn't been updated since 2015). This means Harlow

cannot be used in a .NET 7 runtime, as .NET Core/7 cannot load a .NET Framework 4.5-only assembly. We have two choices:

- **Remove or replace the Harlow-based functionality** – The simplest path is to retire the `ReadShp` component (which uses Harlow) and rely on the DotSpatial-based `ReadDotShp` component for reading shapefiles. In fact, the plugin already has overlapping functionality: `ReadShp` internally converts to GeoJSON via Harlow (but "tends to run slower" as noted in its description [13]), while `ReadDotShp` uses DotSpatial to directly extract geometry and attributes (faster but had issues with some multipart geometry in older version) [14] [15]. If DotSpatial 4 handles your shapefiles correctly, you might drop the Harlow approach entirely.
- **Refactor `ReadShp` to a .NET 7-compatible solution** – If you want to preserve the ability to output raw GeoJSON, consider using **NetTopologySuite's IO** capabilities. For example, NetTopologySuite has a `ShapefileDataReader` (in the `NetTopologySuite.IO.ShapeFile` package) and a `GeoJsonWriter` in `NetTopologySuite.IO.GeoJSON`. You could read the shapefile into NTS geometries and then output a GeoJSON string or JTokens (via JSON.NET) similar to what Harlow was doing. This would be a code rewrite, but it keeps the functionality with fully supported libraries. Since DotSpatial now uses NTS, an alternative is to use DotSpatial to get a FeatureSet and then use NTS/GeoJSON to serialize – however, that may be overkill if `ReadDotShp` already provides geometry and attributes to Grasshopper.

Recommendation: Unless there's a specific reason to keep the Harlow-based JSON output, it's recommended to remove Harlow and unify on one shapefile import method using DotSpatial/NTS. This avoids introducing another dependency just for JSON conversion and simplifies maintenance (DotSpatial can provide attributes which you can always serialize to JSON if needed). If you do drop Harlow, be sure to also remove its NuGet package reference entirely for the net7.0 build (and possibly for net48 as well if not needed). In summary, **Harlow is not compatible with .NET 7**, so that component must either be disabled in the net7 target or reimplemented with a compatible library.

3. Debugging in Rhino 7 vs Rhino 8 (Switching Rhino Runtimes)

When debugging the plugin, you'll want to launch the appropriate Rhino version depending on the target framework:

- For **.NET 4.8 (Rhino 7)** builds, set the external program to Rhino 7's executable. For example, in Visual Studio's Debug properties or launchSettings, use:
  Program: `C:\Program Files\Rhino 7\System\Rhino.exe`
- For **.NET 7 (Rhino 8)** builds, use Rhino 8's executable (assuming Rhino 8 WIP or release is installed). For example:
  Program: `C:\Program Files\Rhino 8\System\Rhino.exe`

In the sample csproj snippet above, we included `<StartProgram>` entries under conditional PropertyGroups so that Visual Studio will automatically use the correct Rhino for each target. This way, if you switch the "Startup Project" target framework, or simply debug the net48 build vs net7 build, it will attach to the correct Rhino. Visual Studio 2022 uses the target framework to determine the runtime to debug [16], so it's important to pick the proper framework or use separate launch configurations.

Rhino 8 .NET Runtime Note: Rhino 8 uses .NET Core by default (initially .NET 7, and updated to .NET 8 in newer builds) [17]. It *can* still run using the .NET Framework runtime in Windows for compatibility if needed

[18] [19], but here we are targeting Rhino 8's default .NET Core mode. By multi-targeting, you have a plugin for each runtime, and Rhino will load the appropriate one. Rhino 8 on Windows can actually load either the .NET 7 plugin or a legacy .NET 4.8 plugin (if you force Rhino 8 into .NET Framework mode via the `SetDotNetRuntime` command), but our goal is to supply a .NET 7 plugin to leverage the new runtime's benefits.

When debugging, load Grasshopper after Rhino starts, and ensure it picks up your *.gha* from the build output location or copied into the Grasshopper Libraries folder. You may maintain two separate plugin files (one .gha for Rhino7, one for Rhino8). Naming them distinctly (e.g., BearGIS_R7.gha and BearGIS_R8.gha) is optional – Rhino/Grasshopper identifies plugins by GUID, so you usually wouldn't have both loaded in the same Rhino anyway. The key is to test in each environment:

- **Testing in Rhino 7**: After building the net48 target, place the `BearGIS.gha` (built under *bin\Debug\net48*) into Rhino 7's Grasshopper Libraries folder (`%AppData%\Grasshopper\Libraries\` or use Grasshopper's File > Special Folders > Components Folder). Launch Rhino 7, run Grasshopper, and verify BearGIS components load. Test various components (shapefile import, etc.) to ensure they work as expected under .NET 4.8.
- **Testing in Rhino 8**: Similarly, take the `BearGIS.gha` from *bin\Debug\net7.0-windows* and place it into the Rhino 8 Grasshopper Libraries folder (which for Rhino WIP is likely `%AppData%\McNeel\Rhinoceros\8.0\Plug-ins\Grasshopper\Components\`). **Important**: Make sure you **do not mix up the versions** – the .NET 7 compiled GHA should only be used in Rhino 8, and the .NET Framework GHA in Rhino 7. Grasshopper might not give a clear error if the wrong runtime plugin is loaded, so keep them separate.

When both builds are working, you have the flexibility to ship a single YAK package or installer that includes both .gha files targeting their respective Rhino versions. Rhino's Yak packaging can handle multi-version plugins by bundling both and specifying compatibility.

4. RhinoCommon and Grasshopper NuGet Packages

Instead of referencing Rhino DLLs from the filesystem, use the official **RhinoCommon** and **Grasshopper** NuGet packages provided by McNeel. This simplifies managing the references for different Rhino versions:

- **RhinoCommon**: McNeel publishes RhinoCommon NuGet packages for each major version. For example, RhinoCommon 7.x for Rhino 7, and RhinoCommon 8.x for Rhino 8. The RhinoCommon 8.x NuGet (e.g. v8.23.25251) is **multi-targeted** – it includes builds for .NET 4.8 *and* .NET 7.0 [20]. *However*, **do not use the RhinoCommon 8 package for the Rhino 7 build** – the assembly version differs and Rhino 7 will refuse to load a plugin referencing RhinoCommon 8. Rhino 7's API expects RhinoCommon 7.x. Therefore, you should reference **RhinoCommon 7.**YY for the net48 target and **RhinoCommon 8.**YY for the net7 target.

Practically, you can add conditional PackageReferences similar to DotSpatial:

```
<ItemGroup Condition="'$(TargetFramework)'=='net48'">
  <PackageReference Include="RhinoCommon" Version="7.**.*" />
  <PackageReference Include="Grasshopper" Version="7.**.*" />
```

```
</ItemGroup>
<ItemGroup Condition="'$(TargetFramework)'=='net7.0-windows'">
  <PackageReference Include="RhinoCommon" Version="8.**.*" />
  <!-- Grasshopper reference for Rhino8 discussed below -->
</ItemGroup>
```

(Replace `7.**.*` and `8.**.*` with the latest build numbers available – typically match the Service Release of your Rhino installations. For example, RhinoCommon 7.36.x for a Rhino 7.36, and RhinoCommon 8.0+ for Rhino8 WIP.)

Using the official packages ensures you get the correct versions of RhinoCommon.dll. RhinoCommon 8.x NuGet specifically states it targets .NET 7.0 and 4.8 [20], aligning with our needs. RhinoCommon 7.x NuGet targets .NET 4.8 (since Rhino7 is Windows-only, no Core) [21].

- **Grasshopper**: McNeel also provides a **Grasshopper package** (which includes Grasshopper.dll and GH_IO.dll). For Rhino **7**, use the Grasshopper 7.x NuGet (e.g. Grasshopper 7.23 for Rhino 7.23) – it targets .NET 4.8 and depends on the matching RhinoCommon 7.x [22] [23]. This will supply the GH_Component classes, etc., for compile-time.

For Rhino **8 (Grasshopper in .NET 7)**, the situation is a bit evolving. At the time of writing, the Grasshopper 8.x NuGet package (for Rhino 8) was still targeting .NET Framework only in the latest stable release [24] – it hadn't yet published a .NET 7 target in the stable channel. (McNeel's WIP Grasshopper 9.0 package targets .NET 8.0 and 4.8, since Rhino 8 eventually moved to .NET 8 runtime [25] [26].) In short, you may not find an official Grasshopper NuGet **for .NET 7** at this moment.

How to proceed? You have two options: 1. Continue referencing Grasshopper via NuGet for Rhino7 (net48), but for Rhino8's build, reference Grasshopper.dll from your Rhino 8 installation manually. This means in the net7.0 `ItemGroup` you'd add something like:

```
<Reference Include="Grasshopper">
  <HintPath>C:\Program Files\Rhino 8\Plug-ins\Grasshopper\Grasshopper.dll</
HintPath>
  <Private>false</Private>
</Reference>
<Reference Include="GH_IO">
  <HintPath>C:\Program Files\Rhino 8\Plug-ins\Grasshopper\GH_IO.dll</HintPath>
  <Private>false</Private>
</Reference>
```

This uses Rhino 8's Grasshopper directly. Ensure those paths are correct for your installation. Using `<Private>false>` is recommended so you don't copy Rhino's assemblies locally (Rhino will provide them at runtime). 2. Target Rhino 8's Grasshopper via a prerelease NuGet if available. The Grasshopper **9.0 WIP** package (shown as 9.0.25252...-wip) multi-targets .NET 8.0 and .NET 4.8 [25]. If you are comfortable targeting .NET 8 instead of .NET 7 for the Rhino 8 build, you could use that. But since we stick to net7.0 (as asked), this WIP package might not help directly (and is intended for Rhino future version). It's safer to use the Rhino 8 installed assemblies for now.

In either case, **do not attempt to use Grasshopper 7.x assemblies in Rhino 8 .NET 7 mode** – Rhino 8 includes a newer Grasshopper, and while it's largely backward-compatible (Grasshopper 1.x continues in Rhino 8), the assembly version and runtime differ. The Grasshopper plugin in Rhino 8 will likely have a version number aligning with Rhino 8's cycle (the NuGet Grasshopper 8.0.x had .NET 4.8 only, implying they hadn't shipped a .NET Core version via NuGet as of Rhino 8's release). Thus, referencing the Rhino 8 copy directly is a pragmatic solution.

Summary of NuGet usage: - Use **RhinoCommon 7** and **Grasshopper 7** packages for the net48 build (Rhino 7). - Use **RhinoCommon 8** package for the net7 build (Rhino 8), and reference Grasshopper libraries appropriately for Rhino 8 (until an official net7/8 Grasshopper package is available). This conditional separation ensures the plugin built for Rhino 7 links against Rhino 7's API, and the Rhino 8 build links against the new API. An alternative approach some developers use is to define a property (e.g. `RhinoVersion=7 or 8`) and conditionally include the different package versions [27] – but since we're already differentiating by target framework, the above method is straightforward.

By using NuGet, you avoid hard-coding paths like the old BearGIS.csproj did (which pointed to Rhino 5's install path) [28]. It also makes it easier to resolve dependencies. For instance, RhinoCommon 8 NuGet will automatically pull in **System.Drawing.Common 7.0** for the net7 build (because RhinoCommon uses System.Drawing in Windows) [29].

5. Code-Level Compatibility Adjustments for .NET 7

Most of BearGIS's code (being C# logic and Rhino/Grasshopper API calls) will work with .NET 7 without changes. RhinoCommon and Grasshopper APIs are largely the same between Rhino 7 and Rhino 8 for Grasshopper 1. However, a few areas to watch and adjust:

- **RhinoCommon API differences**: Rhino 8's SDK is very similar to Rhino 7's for the parts BearGIS uses (Geometry, Grasshopper kernel classes, etc.). You should not need major code changes for RhinoCommon itself. Just ensure you compile separately against the correct version as discussed. If you use any deprecated functions, the compiler might flag them when building against Rhino 8's references – update those if necessary. For example, if any RhinoCommon functions changed signatures or were marked obsolete, handle those in an `#if NET48` vs `#if NET7_0` block accordingly (though unlikely for this plugin's scope).

- **Grasshopper API**: If Rhino 8 still uses Grasshopper 1.x, your GH_Component derived classes will continue to work. In case Rhino 8 transitions to Grasshopper 2 in the future (which has a different API), that would require a complete rewrite of components – but as of Rhino 8, Grasshopper 1 remains the primary (Grasshopper 2 is separate and not automatically running old components). So no changes expected here aside from referencing the correct assembly as above.

- **DotSpatial / NTS Usage**: As covered earlier, the biggest code adjustments will be in how GIS data is handled due to the DotSpatial upgrade. To recap:

- Remove any usage of `DotSpatial.Topology` classes and use NTS equivalents. For example, `DotSpatial.Topology.Envelope` -> `NetTopologySuite.Geometries.Envelope`, `DotSpatial.Topology.Coordinate` -> `NetTopologySuite.Geometries.Coordinate`, etc.

You might need to adjust how features are iterated; DotSpatial's `FeatureSet` may offer a list of `IFeature` where `IFeature.Geometry` is an NTS geometry. You can iterate coordinates via `((NetTopologySuite.Geometries.Geometry)feature.Geometry).Coordinates` to get points, or use NTS geometry's `NumGeometries` and `GetGeometryN(i)` for multipart features. This is analogous to what BearGIS was doing (iterating parts of a feature) but now with NTS. The conversion to `Point3d` and `GH_Point` remains straightforward.

- Ensure to add `using NetTopologySuite.Features; using NetTopologySuite.Geometries;` as needed since DotSpatial 3/4 interop might provide features that implement NTS's `IFeature` interface (in `NetTopologySuite.Features`).

- **System.Drawing and Imaging**: .NET 7 separates System.Drawing into the `System.Drawing.Common` package. RhinoCommon 8 NuGet already accounts for this by requiring System.Drawing.Common on .NET 7 [29]. If BearGIS uses any GDI+ (e.g., creating any `Bitmap` or using System.Drawing colors), be mindful that on .NET Core, `System.Drawing.Common` is only supported on Windows and requires a runtime configuration flag if used in non-Windows environments. Since Rhino/Grasshopper is Windows (for now), this is fine. Just ensure the package is referenced (which RhinoCommon NuGet handles) and **do not use System.Drawing in any code that could run on Mac**. (For cross-platform Rhino compatibility, one would avoid System.Drawing; but Grasshopper plugins are generally Windows-only in Rhino 7, and Rhino 8 on Mac uses .NET Core – if you plan to support Rhino 8 on Mac, you might need to remove System.Drawing usage entirely. BearGIS likely doesn't explicitly use it, aside from what RhinoCommon/Grasshopper use internally.)

- **App.config / Binding Redirects**: The old project had an app.config with binding redirects for DotSpatial and Newtonsoft.Json [30] [31]. In the new .NET 7 environment, **binding redirects are not used** – assembly resolution is handled differently. For the .NET 4.8 build, if you use DotSpatial 3.0.1 and Newtonsoft.Json via NuGet, you likely won't need custom redirects as long as you compile against the correct versions (all DotSpatial assemblies will be 3.0.1, and JSON 11 or higher). You can remove app.config unless you encounter runtime binding issues in Rhino 7. If, for example, Rhino 7's Grasshopper or other components already load a different Newtonsoft.Json version, you might add a binding redirect for Newtonsoft.Json in Rhino 7 – but Rhino 7 by default includes Newtonsoft 12, which is usually compatible. In summary, drop the explicit redirects unless needed; keep the config file out of the .NET 7 build entirely.

- **Any CPU and Platform Targets**: The project should be set to AnyCPU (as shown in the SDK sample). Rhino 7 and 8 are 64-bit processes, so your plugin will run in 64-bit mode. By using AnyCPU and not preferring 32-bit, you satisfy Rhino 8's requirement that plugins be AnyCPU (especially for cross-platform Mac/Windows compatibility) [1]. If BearGIS had any 32-bit specific code or P/Invokes, those would need to be addressed, but it doesn't appear to have any unmanaged code.

- **Miscellaneous .NET API changes**: .NET 7 is largely compatible with code from .NET Framework, but if your code uses any of the following, consider adjustments:

- **Registry** or other Microsoft.Win32 APIs – supported on Windows, but ensure `<UsingMicrosoftWin32Registry>true</UsingMicrosoftWin32Registry>` if needed (not likely in this project).

- **OLE DB / System.Data** – DotSpatial uses `System.Data.OleDb` for reading .dbf files. We saw DotSpatial.Data has a dependency on `System.Data.OleDb` 6.0+ [32]. That means on .NET 7 it will use the modern OleDb package (which is Windows-only). Make sure the **Microsoft Access Database Engine** is installed if needed for .dbf access. This isn't a code change, but a runtime requirement. Rhino 8 running on .NET 7 can still use OleDb since we're targeting windows – just something to be aware of if a shapefile's DBF fails to read (installing the ACE OLEDB driver may be necessary on a new machine).
- **File Paths** – .NET Core is stricter about file path handling (e.g., `System.IO.Path` may throw on illegal characters where .NET Framework was lenient). Ensure that any file paths (like those passed into shapefile readers) are well-formed. This is usually fine.

In summary, **most code changes revolve around DotSpatial → NTS adjustments and removing Harlow**. The RhinoCommon and Grasshopper API usage remains virtually the same between Rhino 7 and 8 for Grasshopper 1, so your components (GH_Component subclasses, etc.) should compile and run in both environments without modifications.

6. Build & Test for Rhino 7 and Rhino 8

After making the above changes, follow these steps to build and test each target:

- **Build the solution** in Visual Studio 2022 (needed for .NET 7 development) or via `dotnet build`. The multi-target project will compile twice – once for net48, once for net7.0-windows. Verify that you get two sets of outputs and **no compiler errors** for either target. If there are errors in the net7 build, use the `#if NET7_0` (or corresponding constant) to conditionally compile code that differs from net48 (for example, if you still need to keep some DotSpatial.Topology code for net48, and different code for net7 – though if both targets use DotSpatial 3/4, you can likely use one unified code path).
- **Install/Load in Rhino 7**: Copy the net48 `BearGIS.gha` (and any required supporting files, though ideally everything is in the GHA) into Rhino 7's Grasshopper Libraries. Launch Rhino 7 and Grasshopper. The BearGIS plugin components should appear (likely under the *BearGIS* category as before). Test reading a shapefile, projecting coordinates, etc., to ensure all third-party pieces (DotSpatial 3.0.1, etc.) function inside Rhino 7. If something fails to load, check Rhino's command-line for error messages about missing assemblies. You might need to unblock the .gha (Windows may block downloaded DLLs) or install any dependency like the Access DB driver if .dbf reading fails. Usually, if DotSpatial was added via NuGet, the needed DLLs (DotSpatial*..dll, NetTopologySuite.dll, etc.) will be merged into the plugin folder next to the GHA. Grasshopper can load them if they are in the same folder as the GHA. If not, consider using a Yak** package to bundle or using AppDomain assembly resolve – but typically copying all required DLLs to the Grasshopper Libraries folder works. Ensure that the plugin isn't referencing any absolute old paths. With PackageReference, copy-local is true by default for needed assemblies, so the build's output folder should contain the DotSpatial and NTS DLLs for each target.
- **Install/Load in Rhino 8**: Similarly, take the net7 `BearGIS.gha` and its accompanying DLLs and place them in Rhino 8's Grasshopper Libraries. Note that Rhino 8 (in .NET Core mode) **isolates plugin assemblies** differently than Rhino 7. Rhino 8 uses a `.rhp` to load .gha, and the loading is more strict. If any assembly fails to load (e.g., a missing dependency or an untrusted certificate issue), Rhino 8 will report it. Make sure all NuGet dependencies for net7 (DotSpatial 4, NetTopologySuite, etc.) are present. Again, placing them in the same folder as the .gha is usually sufficient.

- Rhino 8, being .NET Core, does not use the GAC or machine-wide .NET Framework assemblies, so every dependency must be either in the plugin folder or in the Rhino 8's runtime packages. The RhinoCommon and Grasshopper assemblies are provided by Rhino, but DotSpatial/NTS are your responsibility to deploy. One advantage of Grasshopper plugins is they often bundle everything in the GHA (some developers use ILMerge or similar). If you prefer, you could consider using ILRepack to merge DotSpatial/NTS into one .gha to reduce number of files, but that's optional.

- **Switch Rhino 8 to use .NET Framework (optional)**: Rhino 8 on Windows can run using .NET 4.8 for compatibility with old plugins [18]. If for some reason you want to test your .NET Framework plugin in Rhino 8, you can launch Rhino 8 with the `/netfx` flag or use the `SetDotNetRuntime` command [33]. Then Rhino 8 will load the net48 BearGIS.gha. This isn't necessary since we have a net7 plugin, but it's a good way to verify that the .NET 4.8 plugin also works in Rhino 8's fallback mode. Generally, providing a net7 plugin for Rhino 8 is preferable for performance and future-proofing.

- **Functional testing**: Once loaded, re-run all typical usage scenarios:

- Importing various GIS formats (shapefiles via both import components, if Harlow version was kept for test, etc.).
- Exporting shapefiles/GeoJSON.
- Reprojection functions (DotSpatial Projections).
- Ensure that no exception is thrown due to missing methods (for example, if some DotSpatial API changed slightly – adjust accordingly if so).
- Check that icons (if any) and resources load properly (the .resx in BearGIS for icons should still work as before).

If any issues arise specific to .NET 7 (like P/Invoke or unsupported API), the Rhino 8 command-line **compatibility report** can be useful. Rhino 8 scans plugins for known issues when loaded in .NET Core and may list problematic calls [34]. Ideally, with the migration steps above, you will have resolved those by updating the libraries.

7. Additional Recommendations (Rhino 8 Specific)

- **Use Yak for multi-version deployment**: Consider creating a Yak package that includes both builds. McNeel's Rhino 8 plugin templates have an option to include yak spec for multi-target plugins [35]. You can manually create a yak package with a structure like:

```
/net48/BearGIS.rhp (this is basically the BearGIS.gha for Rhino7, renamed
and coupled with a RhinoCommon loader stub if needed)
/net7/BearGIS.rhp  (the Rhino8 plugin, which will internally load
BearGIS.gha or include it)
```

  However, since Grasshopper uses .gha, not .rhp, the packaging might differ. It might be simpler just to distribute two .gha files with clear labeling.

- **Documentation and Versioning**: Document somewhere (readme or internal plugin info) which Rhino versions the plugin supports and ensure the Grasshopper `_ComponentGuid` remains the same for identical components across versions (so users opening a Grasshopper file in Rhino 8 will seamlessly use the new plugin). If you changed any assembly or component GUIDs, old definitions might break – but since we largely kept the same code, this shouldn't be an issue.

- **Performance Considerations**: .NET 7 can bring performance improvements automatically [36]. After migration, observe if any parts (like heavy computations or large data handling) run faster in Rhino 8. If so, you might advertise that as a benefit of using the Rhino 8 version of your plugin.

- **Troubleshooting**: If Grasshopper fails to load the .NET 7 plugin and throws an error about "unblock" or "not a Grasshopper assembly", ensure the .gha and all dependent DLLs are unblocked (Windows mark-of-the-web issue). Also, make sure you compiled in **Debug** or **Release** as appropriate; sometimes Grasshopper might not load a plugin built as AnyCPU/AnyConf – but generally it does. You can use the `GrasshopperDeveloperSettings` command in Rhino to manage plugin loading (there's an option for memory load vs COFF loading – for .NET 7 it should automatically use the new loading mechanism).

By following this guide, BearGIS will be modernized to run on both Rhino 7 and Rhino 8. In summary, we updated the project to multi-target .NET 4.8 and 7 (as recommended by McNeel [4]), switched to the newer DotSpatial libraries (with NTS) and removed incompatible dependencies, used NuGet for RhinoCommon/Grasshopper references, and provided instructions for debugging and deployment in each environment. This ensures continuity for Rhino 7 users while preparing the plugin for Rhino 8's .NET Core runtime future. Happy coding!

Sources:

- McNeel Developer Guide – *Moving to .NET Core (.NET 7) for Rhino 8* [4] [5]
- McNeel RhinoCommon NuGet (v8) – multi-targets .NET 4.8 & 7.0 [20]
- DotSpatial Release Notes – .NET 6/7 support and NTS integration [10]
- Harlow NuGet Info – targets .NET Framework 4.5 only (GeoJSON shapefile reader) [12]
- Example user forum (McNeel Discourse) on multi-targeting RhinoCommon 7 & 8 packages [27] (approach to conditionally reference correct versions)

---

1 2 4 5 16 17 18 19 33 34 35 36 Rhino - Moving to .NET Core
https://developer.rhino3d.com/guides/rhinocommon/moving-to-dotnet-core/

3 28 BearGIS.csproj
https://github.com/nicoazel/BearGIS/blob/42871b85ee3047201687d4610f0c661411092c6a/BearGIS/BearGIS.csproj

6 GitHub - DotSpatial/DotSpatial: Geographic information system library written for .NET
https://github.com/DotSpatial/DotSpatial

7 10 Releases · DotSpatial/DotSpatial · GitHub
https://github.com/DotSpatial/DotSpatial/releases

8 NuGet Gallery | DotSpatial.Controls 4.0.656
https://www.nuget.org/packages/DotSpatial.Controls

9 14 15 ReadDotShp.cs
https://github.com/nicoazel/BearGIS/blob/42871b85ee3047201687d4610f0c661411092c6a/BearGIS/Importers/ReadDotShp.cs

11 12 NuGet Gallery | Harlow 0.6.0
https://www.nuget.org/packages/Harlow

13 ReadShp.cs
https://github.com/nicoazel/BearGIS/blob/42871b85ee3047201687d4610f0c661411092c6a/BearGIS/Importers/ReadShp.cs

20 29 NuGet Gallery | RhinoCommon 8.23.25251.13001
https://www.nuget.org/packages/rhinocommon

21 22 23 NuGet Gallery | Grasshopper 7.23.22282.13001
https://packages.nuget.org/packages/Grasshopper/7.23.22282.13001

24 NuGet Gallery | Grasshopper 8.8.24170.13001
https://www.nuget.org/packages/Grasshopper/8.8.24170.13001

25 26 NuGet Gallery | Grasshopper 9.0.25252.12305-wip
https://www.nuget.org/packages/Grasshopper/9.0.25252.12305-wip

27 Nuget packages/supporting multiple Rhino version - Rhino Developer - McNeel Forum
https://discourse.mcneel.com/t/nuget-packages-supporting-multiple-rhino-version/178206

30 31 app.config
https://github.com/nicoazel/BearGIS/blob/42871b85ee3047201687d4610f0c661411092c6a/BearGIS/app.config

32 NuGet Gallery | DotSpatial.Data 4.0.656
https://www.nuget.org/packages/DotSpatial.Data/4.0.656