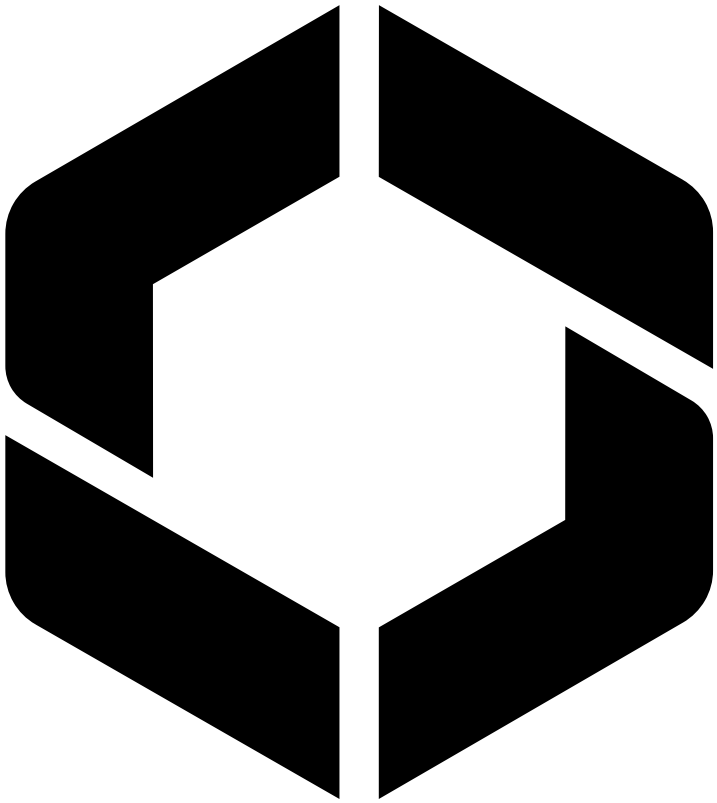

Cities: Skylines II Modding Instructions

July 19, 2024

Table of Contents

Introduction

This document contains very detailed step-by-step instructions for creating a Cities: Skylines II (CS2) code modification. A code modification is known as a “mod”. A mod adds on to or modifies existing game play. This document is not for creating a map or other asset. Hereinafter, “mod” refers to the CS2 code mod being created.

There are ways to organize your code and do things other than the methods detailed in this document. Other mod authors will have different preferences on how to do things. You probably have your own preferences. But following these instructions will result in a basic working mod to serve as starting point for adding your own functionality and making changes to suit your own style and preferences.

For more information on CS2 itself, see:
https://cs2.paradoxwikis.com/Cities_Skylines_II_Wiki

For more information on CS2 modding, see:
https://cs2.paradoxwikis.com/Modding
https://cs2.paradoxwikis.com/Community-made_guides

For help with CS2 modding, see the following, especially the sections on Code Mods and Development:
https://discord.com/channels/1169011184557637825

Assumptions

This document assumes the following regarding your environment. If your environment is different, then you will need to make adjustments to the instructions in this document for your environment differences.

- CS2 was installed from Steam on a Windows PC. This implies you already have a Steam account.
- Development will be with Microsoft Visual Studio on a Windows PC. As of this writing, author is using “Microsoft Visual Studio Community 2022 (64-bit)”, which is free, and “Version 17.10.2”.
- Mod source code will be stored on GitHub.
- No mod manager (e.g. r2modman, Skyve) is being used during development. Feel free to use a mod manager outside of development.

This document assumes the reader is familiar with the following. Reader does not need to be an expert. But at least some familiarity is very helpful. If you are unfamiliar with any of these, search the internet for more information.

- Using Microsoft Visual Studio.
- Programming in C#.
- Entity Component System (ECS) software architecture. For more information on ECS, see:
  https://cs2.paradoxwikis.com/ECS_-_Entity_Component_System
- For creating a user interface:
  - HyperText Markup Language (HTML).
  - Programming in TypeScript. TypeScript is a strongly typed programming language that builds on JavaScript. For more information on TypeScript, see:
    https://www.typescriptlang.org/

Accounts

You will need a Paradox account and a GitHub account. These are described in the following sections. Even if you already have these accounts, read thru the sections anyway to complete any steps beyond just creating the accounts.

Paradox Account

Paradox Mods is the only official place for CS2 mods. You should obtain CS2 mods only from Paradox Mods. You can browse for mods on Paradox Mods without an account. But a Paradox account is required to use a mod and to publish a mod.

The Paradox Mods website for CS2 mods is:
https://mods.paradoxplaza.com/games/cities_skylines_2

If you don't already have a Paradox account, create a new account now, which is totally free:

1. Start the game to the main menu.
2. On the right side of the screen, click on "Login". A login screen is displayed.
3. On the login screen, click on "Create Account". The "Create Paradox Account" screen is displayed.
4. On the "Create Paradox Account" screen, enter the requested information and then click on "Create Account". After the account is created, a confirmation message is displayed.
5. Click "OK" on the confirmation message. The game main menu is displayed.
6. Go to your email:
   a) Find the message from Paradox Interactive.
   b) In the message, click on "Confirm Account" to confirm your account. A browser is opened showing that your account is now confirmed.
   c) Close the browser.
7. On the main menu of the game, click on "Paradox Mods". It is okay if the button shows "Beta". Modding is considered to be in "Beta" testing status. The "Paradox Mods" screen is displayed.
8. On the "Paradox Mods" screen:
   a) Click on "Me". A message will be displayed that a username is required.
   b) Under the message, click on "Edit profile". A message is displayed asking permission to open a web browser. Click on "Yes" to allow the game to open the displayed link in a browser.
   c) In the browser:
      i. In the upper right, click on "Log in". A popup is displayed.
      ii. In the popup, click on "Log In". The log in screen is displayed.
      iii. Log in to your Paradox account with the email and password used to create your account. The "Choose username" screen is displayed.
      iv. On the "Choose username" screen, enter a username. Choose your username wisely because it can be changed only by deleting and recreating your account. You might want to use the same username as your Steam and/or GitHub accounts (see section "GitHub Account").
      v. Click on "Set username". The screen shows your mods, which will be none.
      vi. If desired, click on "Edit profile" to change your avatar, bio, or external links.
      vii. You may optionally link your Paradox account to other accounts like Steam, PlayStation, and Microsoft. In the upper right, click on your username and select "Account settings". The "Account settings" screen is displayed in a new browser. Under "Account Linking", click on "Link account" for the accounts you want to link.
      viii. Log out of your Paradox account and close the browser.
   d) In the game, close the "Paradox Mods" screen.
9. Click on "Paradox Mods" again. The "Paradox Mods" screen is displayed again.
10. On the "Paradox Mods" screen:
    a) Verify that the "Me" tab now shows your profile information, including your username.
    b) You now have full access to all the activities on the Paradox Mods screen. But importantly for this document, you will be able to publish your mod later.
    c) Close the "Paradox Mods" screen.
11. Exit the game to desktop.

Log into your Paradox account in the game:

1. Start the game to the main menu.
2. If not already logged in, log in to your Paradox account in the game:
   a) Under “Paradox Account”, click on “Login”. A log in screen is displayed.
   b) On the log in screen, enter your Paradox account email address (not username) and password.
   c) Click on “Login”. You are returned to the game main menu.
3. On the main menu under “Paradox Account”, verify your Paradox account username is displayed. If your username is not displayed, see the steps above for creating your username.
4. Once logged in, there is a “Logout” button to log out of your Paradox account. It is recommended to stay logged in to Paradox in the game. If you logout and log in again, all your subscribed mods will be downloaded and installed again.
5. Exit the game to desktop.

Create a Paradox account data file that will be used later when publishing your mod:

1. Use a text editor (e.g. Notepad) to create a new text file with 2 lines:
   a) Line 1 contains your Paradox account username (not email address).
   b) Line 2 contains your Paradox account password.
2. Save the file in a folder where you will remember where it is located.
   a) Do NOT save the file in your Visual Studio project folder. Placing the file in a folder separate from your project helps prevent the file from accidentally being included in the project and copied to GitHub for all to see.
   b) Possible file locations are your desktop or your Documents folder or (this author’s preference) one folder above all the project folders for your CS2 mods.
3. Save the file with the name “pdx_account.txt”.
4. Close the Paradox account data file.

GitHub Account

Your source code must be available for others to review. GitHub seems to be preferred by most mod authors and is easily linked from Paradox Mods.

If you don’t already have a GitHub account, create a new account now, which is totally free:

1. In a browser, go to the GitHub web site:
   https://github.com/
2. Click on “Sign up”. The “Welcome” screen is displayed.
3. On the “Welcome” screen, follow the prompts to create a new account.
4. For the username, you might want to use the same username as your Steam and/or Paradox accounts.
5. Log out of GitHub and close the browser.

In Visual Studio, set up access to GitHub:

1. TBD Detailed steps are not available here because GitHub is already set up in the author’s Visual Studio instance. But there should be a “Git” top-level menu item in Visual Studio. GitHub access might also get performed when the repository for your mod is first created.

Modding Toolchain

CS2 uses the Modding Toolchain to enable the creation of mods. The Modding Toolchain is required to create a mod (technically not *required*, but there is no reason to try to create a mod from scratch instead).

For more information on the Modding Toolchain, see:
https://cs2.paradoxwikis.com/Modding_Toolchain

Install the Modding Toolchain:

1. Start the game to the main menu.
2. From the game's main menu, click on "Options". The "Options" screen is displayed.
3. On the "Options" screen, on the left side, click on "Modding". It is okay if the button shows "Beta". The modding tools are considered to be in "Beta" testing status. The "Toolchain state" and "Dependencies" are displayed.
4. All of the listed "Dependencies" must be satisfied to create and publish a mod.
    a) Satisfied dependencies are indicated as "Installed", "Activated", or "Detected" with a green check mark.
    b) For each dependency not satisfied, click on the dependency to expand it and click on "Install" or "Update". Follow any in-game instructions for the installation or update.
    c) Some dependencies have a specific or minimum required version. If given a choice, be sure to install a compatible version.
    d) For the Unity Editor installation, the default folder is "C:\Program Files\Unity", but Unity will actually be installed in "C:\Program Files\Unity <version>" where "<version>" is the Unity version. This allows you to have more than one Unity version installed at the same time.
5. Verify that the "Toolchain state" indicates "Installed" with a green check mark.
6. Close the "Options" screen and exit the game to desktop.

Basic Mod Project

The following sections describe the steps for creating a basic Visual Studio project for your mod. Dependencies from the Modding Toolchain will be used, so be sure they are all satisfied before proceeding.

Choose Assembly Name

The assembly name will be used for many things in the mod including: default namespace, installation folder name, DLL file name, Product name in the DLL file properties, etc.

Choose an assembly name for your mod:

1. Choose an assembly name that is a few words representing your mod.
2. Choose an assembly name that will not be too much like any other existing CS2 mod.
    a) In a browser, go to: https://mods.paradoxplaza.com/games/cities_skylines_2
    b) Search for mods with similar names or features that are tagged "Code Mod".
3. Choose the assembly name wisely.
    a) After the mod project is created (see section "Create Mod Project"), it will be difficult to change the assembly name.
    b) The assembly name will likely be the basis for your mod title that users will see (see section "Configure Mod Project").
4. Include only letters and digits.
5. Include no spaces, underscores, or special characters.
6. Capitalize only the first character of each word in the assembly name (i.e. Pascal case).
7. Hereinafter, wherever "AssemblyName" is referenced, it means this chosen assembly name.

Create Mod Project

Create the mod project in Visual Studio using the mod template:

1. In Visual Studio, create a new project.
    a) If the startup screen is displayed, click on "Create a new project".
    b) Otherwise, in the menu, click on "File" -> "New Project...".
    c) Either way, the "Create a new project" screen is displayed.
2. On the "Create a new project" screen:

a) The Modding Toolchain installs a template that makes it much easier to start a new mod. Search the templates for “cities skylines”.
b) Click on the template for “Cities Skylines II mod (Colossal Order Ltd)” to highlight it.
c) Click on “Next”. The “Configure your new project” screen is displayed.

3. On the “Configure your new project” screen:
   a) For “Project name”, enter the AssemblyName chosen earlier.
   b) For “Location”, set as needed. This author uses: C:\Users\<name>\Documents\Visual Studio Projects\Cities Skylines 2 Mods\My Mods
   c) For “Solution” (if present), select “Create new solution”.
   d) For “Place solution and project in the same directory”, set to checked or unchecked according to your preference. This author prefers solution and project in the same directory (i.e. checked).
   e) Click on “Next”. The “Additional information” screen is displayed.
4. On the “Additional information” screen:
   a) For “Include mod settings”:
      i. It is easier to include settings code now and delete the settings code later if not needed than it is to include settings code later from scratch.
      ii. If your mod will or might have settings in the game’s “Options” menu, set to checked.
      iii. If your mod will not have settings in the game’s “Options” menu, set to unchecked.
   b) For “Include key bindings”:
      i. Key bindings can be included only if mod settings were included above.
      ii. It is easier to include key bindings code now and delete the key bindings code later if not needed than it is to include key bindings code later from scratch.
      iii. If your mod will or might have a key binding that you want the user to be able to set in the game’s “Options” menu, set to checked.
      iv. Otherwise, set to unchecked.
   c) For “Short description” and “Long description”, leave the default text. Descriptions will be updated later.
   d) Click on “Create”. The mod solution and project are created and opened in Visual Studio.

Configure Mod Project

Update the project properties:

1. In the Visual Studio menu, click on “Project” -> “AssemblyName Properties”. The project properties are opened.
2. In section “Build” -> “Errors and warnings”:
   a) Under “Treat warnings as errors”, check “Instruct the compiler to treat warnings as errors”. This is an author preference, but is recommended.
3. In section “Package” -> “General”:
   a) Set the “Title” to a human-friendly title of the mod. This usually is the AssemblyName with spaces between the words. The title will be seen by users. Do not include any double quotes (") in the title.
   b) Set the “Package Version” to “0.1.0” for initial development.
   c) Set the “Description” to a short one sentence description of the mod. This can be changed later. Do not include any double quotes (") in the description.
   d) Set the “Copyright” to “Copyright © <year>” where <year> is the current year.
   e) The “Assembly version” and “File version” both default to blank. Leave these blank so they automatically derive from the “Package Version” property, as desired.
   f) The “Title”, “Description”, and “Version” will be used later for publishing the mod to Paradox Mods.
4. Save and close the project properties.

Make assembly info easily accessible as constant strings in the code:

1. Under the project, add a new folder named “ModAssemblyInfo”.

2. Copy file “ModAssemblyInfo.csproj” from GitHub: https://github.com/rcav8tr/CS2-Modding
3. In Visual Studio, paste the file to the “ModAssemblyInfo” folder.
4. Click on the pasted “ModAssemblyInfo.csproj” file to view its properties.
5. In the properties for the “ModAssemblyInfo.csproj” file, verify “Build Action” is “None” and “Copy to Output Directory” is “Do not copy”. Set if needed.

Update the project file:

1. In the Visual Studio menu, click on “Project” -> “Edit Project File”. The project file is opened.
2. In the main “PropertyGroup”:
   a) Verify that the “Title”, “Version”, “Description”, and “Copyright” are correct.
3. Replace two “TreatWarningsAsErrors” entries with one entry:
   a) Find the two “PropertyGroup” entries for “TreatWarningsAsErrors”. There will be one for Debug and one for Release.
   b) Copy the “TreatWarningsAsErrors” tag from one of the two property groups and paste it at the end of the main “PropertyGroup”.
   c) Delete both the Debug and Release property groups for “TreatWarningsAsErrors”.
4. Define PDX account data file path:
   a) At the end of the main “PropertyGroup”, add an entry to define the PDX account data file that was created earlier. For example, if the file was saved on your desktop, the entry would be:
   ```
   <PDXAccountDataPath>$(USERPROFILE)\Desktop\pdx_account.txt</PDXAccountDataPath>
   ```
   b) The value of “$(USERPROFILE)” is typically:
   ```
   C:\Users\<name>
   ```
   so if your file is stored elsewhere, adjust the whole path or just the path after “$(USERPROFILE)”.
5. Automatically create the “ModAssemblyInfo.cs” file:
   a) In the “ItemGroup” at the end of the file, delete the entry for the “ModAssemblyInfo.csproj” file that was automatically added by Visual Studio:
   ```
   <None Include="ModAssemblyInfo\ModAssemblyInfo.csproj" />
   ```
   b) Add the following before the “</Project>” tag:
   ```
   <!-- Automatically build ModAssemblyInfo.cs file. -->
   <Import Project="ModAssemblyInfo\ModAssemblyInfo.csproj" />
   <ItemGroup>
       <None Include="ModAssemblyInfo\ModAssemblyInfo.csproj" />
   </ItemGroup>
   ```
   c) Save the project file. As soon as the project file is saved, Visual Studio immediately builds the “ModAssemblyInfo.cs” file in the “ModAssemblyInfo” folder.
      i. Open the “ModAssemblyInfo.cs” file and verify its contents.
      ii. In the file’s properties, verify the “Build Action” is “C# Compiler”. Set if needed.
      iii. Close the “ModAssemblyInfo.cs” file.
6. Close the project file.

If settings code was included, configure it now:

1. Some of these things seem to be sort of a standard used by some other mod authors. The exact naming and structure may differ from one author to another.
2. Under the project, add a new folder named “ModSettings”.
3. Move the “Setting.cs” file from under the project to the new “ModSettings” folder.
   a) If asked to confirm the move, click on “OK”.
   b) If asked to “Adjust namespaces for moved files”:
      i. This author prefers all code to be in one namespace, select “No”.
      ii. If you prefer your namespaces to match your folders, select “Yes”.
   c) If not asked to “Adjust namespaces for moved files”, you may need to adjust the namespaces manually to add or remove the “.ModSettings” suffix according to your preference.
4. Rename file “Setting.cs” to “ModSettings.cs”.

a) If asked “Would you also like to perform a rename in this project …”, select “Yes”.
b) Otherwise, find all occurrences of the “Setting” class in the solution and change to “ModSettings”.
c) In the “ModSettings.cs” file, the default class properties provide examples for creating various types of settings (e.g. button, slider, checkbox, drop down, keyboard binding, etc.). For now, leave all the default class properties. Later, you can make changes for whatever settings your mod requires.

5. Separate the settings logic from the locale logic:
   a) In the “ModSettings” folder, add a new C# class file named “LocaleEN.cs”. The file is opened.
   b) Remove all the “using” statements. They should all be grayed out to indicate unused.
   c) This author prefers all code to be in one namespace, so remove “.ModSettings” from the end of the namespace.
   d) If you prefer your namespaces to match your folders, then leave the namespace unchanged.
   e) Cut the “LocaleEN” class from the “ModSettings.cs” file and paste it to the new “LocaleEN.cs” file, replacing the default “LocaleEN” class.
   f) In “ModSettings.cs”, remove unused “using” statements.
   g) Save and close the “ModSettings.cs” file.
6. In the “LocaleEN.cs” file:
   a) Add needed “using” statements to fix errors.
   b) In the “ReadEntries” method:
      i. In the entry for “m_Setting.GetSettingsLocaleID()”, replace the quoted string of your AssemblyName (including the quotes) with “ModAssemblyInfo.Title” (without the quotes).
      ii. In the entry for “m_Setting.GetBindingMapLocaleID()”, replace the quoted string “Mod settings sample” with “ModAssemblyInfo.Title” (without the quotes).
      iii. Note that if your mod title is too long, then it will be cut off when displayed on the game’s Options screen. If that happens, replace “ModAssemblyInfo.Title” with some abbreviated version of your mod title enclosed in quotes.
      iv. If your mod will later support more than one language, then later you might want to use locale specific text instead of “ModAssemblyInfo.Title”.
   c) Save and close the “LocaleEN.cs” file.
7. For more information on mod settings, see: https://cs2.paradoxwikis.com/Options_UI

Separate logging from the Mod:

1. Copy the “LogUtil.cs” file from GitHub: https://github.com/rcav8tr/CS2-Modding
2. In Visual Studio, paste the file under the project.
3. In the properties for the “LogUtil.cs” file, verify “Build Action” is “C# compiler”. Set if needed.
4. In the “LogUtil.cs” file just added:
   a) Set the namespace to your AssemblyName.
   b) Save and close the “LogUtil.cs” file.
5. In the “Mod.cs” file:
   a) Replace each occurrence of “log.Info” with “LogUtil.Info”.
   b) Delete the code for getting a logger:
      ```
      public static ILog log = …
      ```
   c) Remove unused “using” statements.
   d) Save and close the “Mod.cs” file.
6. If settings code was included, then in the “ModSettings\ModSettings.cs” file:
   a) Replace each occurrence of “Mod.log.Info” with “LogUtil.Info”.
   b) Save and close the “ModSettings.cs” file.
7. The “LogUtil” class is specifically not called “Logger” to avoid confusion with the Unity logger “UnityEngine.Logger”, which will still be accessible in your mod. Use the LogUtil methods for all logging in your mod.

Build Mod Project

Verify solution builds successfully:

1. Save the entire solution.
2. Rebuild the entire solution.
3. Verify the build succeeded with no errors.

In Windows, verify the mod's files were created:

1. Go to the game's local mods folder:
   C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\Mods
2. Verify there is a folder for the AssemblyName.
3. In the AssemblyName folder, verify the following files are present:
   AssemblyName.dll
   AssemblyName.pdb
   AssemblyName_linux_x86_64.so
   AssemblyName_mac_x86_64.bundle
   AssemblyName_win_x86_64.dll
   AssemblyName_win_x86_64.pdb
4. Right-click on the "AssemblyName.dll" file and select "Properties". The "Properties" screen is displayed.
5. On the "Properties" screen, click on the "Details" tab. The file's details are displayed.
6. On the "Details" tab, verify the project configuration settings are correctly represented in the following DLL details:
   a) "File description" should be the AssemblyName.
   b) "File version" should be "0.1.0.0".
   c) "Product name" should be the AssemblyName.
   d) "Product version" should "0.1.0".
   e) "Copyright" should be the copyright text set earlier.
7. Close the "Properties" screen.

Test Mod

If your mod includes settings, test your mod's settings:

1. Start the game to the main menu.
2. On the main menu, click on "Options". The "Options" screen is displayed.
3. On the "Options" screen:
   a) On the left, verify there is an entry with your assembly title (not assembly name).
   b) Click on the entry for your mod.
   c) On the right:
      i. Verify your mod's default options are displayed.
      ii. If your mod includes key bindings, verify the mod's default key bindings are displayed.
      iii. Your mod's options were generated by the logic in the "ModSettings" folder of your project.
   d) Change the value of the Int slider.
   e) In Windows:
      i. Verify the mod settings file is present at:
         C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\AssemblyName.coc
      ii. The settings file includes only settings that are different than the defaults. So if the file is absent or empty, it means all the settings have their default value.
      iii. The folder also contains settings files for the game and any other mods.
4. Close the "Options" screen.

In Windows, verify your mod's log file:

1. If not already in the game, start the game to the main menu. This creates your mod's log file.
2. Exit the game to desktop.
3. Go to the game's logs folder:
   C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\Logs
4. Verify there is a log file for your mod named "AssemblyName.log".
5. Open your mod's log file.
   a) Verify there is an entry for "OnLoad" at the start.
   b) Verify there is an entry for "OnDispose" at the end.
   c) Close the log file.

Congratulations! Your mod was successfully loaded by the game and your mod's basic functionality is working!

Create Your Mod

Now go make your mod do what your mod must do! To this basic mod project, add the specific functionality your mod will provide in the game. The following sections may be helpful during the development of your mod.

When you are ready to publish your new mod to Paradox Mods, continue with section "Publish New Mod To Paradox Mods".

User Interface

Most mods will have some type of user interface (UI) to allow the user to interact with your mod and/or to display information to the user. An exception might be a mod that only changes game behavior and perhaps has only settings to change the mod's behavior. The Modding Toolchain includes a template for adding UI to a mod project.

For more information on UI modding, see:
https://cs2.paradoxwikis.com/UI_Modding

To include a UI for your mod from the UI template:
1. Open your project in Visual Studio.
2. On the Visual Studio menus, select "Tools" -> "Command Line" -> "Developer Command Prompt". A new command prompt window is opened.
3. In the command prompt window:
   a) Verify the current directory is set to your project folder. If not, change directory to your project folder.
   b) Run the following command to add the UI logic to your mod project from the UI template:
      `npx create-csii-ui-mod`
4. The npx command will prompt for some details:
   a) For "Project name", enter "UI". This seems to be the standard used by other mod authors.
   b) For "Author", enter your Paradox account username (not email).
   c) Wait for the process to complete. It might take about 15-20 seconds.
   d) Verify the command window shows "Project 'UI' created successfully".
   e) Close the command prompt window.
5. Verify there is now a "UI" folder in the project. The folder name is the "Project name" entered above.
6. Exclude "node_modules" folder:
   a) Expand the "UI" folder.
   b) Notice there is a "node_modules" folder that was created by the npx command. There are thousands of files in the "node_modules" folder. This makes solution-wide source code searches take a long

time because all these files are searched. The following steps will exclude the “node_modules” folder.

c) In the Visual Studio menu, click on “Project” -> “Edit Project File”. The project file is opened.
d) At the end of the main “PropertyGroup”, add the following entry:
```
<DefaultItemExcludes>UI\node_modules\**;$(DefaultItemExcludes)</DefaultItemExcludes>
```
e) Save the project file.
f) Under the UI folder, notice that the “node_modules” folder is no longer listed or if you have the “Show All Files” option turned on then the folder is shown as excluded. It might take a few seconds for the change to be shown.
g) Close the project file.

7. Automatically build the “mod.json” file and automatically build the UI with the project:
   a) Expand the “UI” folder.
   b) Open the “mod.json” file. In the “mod.json” file:
      i. Notice that the file contains the “id” and “author” you entered earlier.
      ii. Notice that the file defaults to version “1.0.0” which is different than your mod’s version of “0.1.0”.
   c) Rather than having to remember to update this file every time some of the data is changed (especially the version), the file will be automatically rebuilt.
   d) Copy the “mod.json.csproj” file from GitHub https://github.com/rcav8tr/CS2-Modding
   e) In Visual Studio, paste the file under the “UI” folder.
   f) In the Visual Studio menu, click on “Project” -> “Edit Project File”. The project file is opened.
   g) In the “ItemGroup” at the end of the file, delete the entry for the “mod.json.csproj” file that was automatically added by Visual Studio:
```
<None Include="UI\mod.json.csproj" />
```
   h) Add the following before the “</Project>” tag:
```
<!-- Automatically build mod.json file. -->
<Import Project="UI\mod.json.csproj" />
<ItemGroup>
    <None Include="UI\mod.json.csproj" />
</ItemGroup>

<!-- Automatically build the UI with the project. -->
<Target Name="BuildUI" AfterTargets="AfterBuild">
    <Exec Command="npm run build" WorkingDirectory="$(ProjectDir)/UI" />
</Target>
```
   i) Save the project file. As soon as the project file is saved, Visual Studio immediately builds the “mod.json” file, overwriting the existing file.
      i. Return to the “mod.json” file.
      ii. Verify:
          “id” is AssemblyName;
          “author” is your Paradox account username taken from your Paradox account data file;
          “version” is “0.1.0” taken from the assembly version;
          “dependencies” is “[]” (i.e. empty square brackets).
      iii. Close the “mod.json” file.
   j) Close the project file.
8. Rebuild the entire solution.
9. Go to the game’s local mods folder:
   C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\Mods
10. In the AssemblyName folder, verify the “AssemblyName.mjs” file is present.
11. To test that the UI in your mod is working, in section “Developer Modes”, see the steps to test UI developer mode.

Harmony

Harmony is a library by Andreas Pardeike for patching, replacing, and decorating application methods during runtime. You can use Harmony to alter the functionality of all the available assemblies of that application. Harmony gives you:

- A way to keep the original method intact
- Execute your code before and/or after the original method
- Modify the original with IL code processors
- Multiple Harmony patches co-exist and don't conflict with each other
- Works at runtime and does not touch any files

For more information on Harmony and how to use it, see:
https://harmony.pardeike.net/
https://github.com/pardeike/Harmony

The "DependencyPack" mod will be used to include the Harmony DLL file "0Harmony.dll" into the game. Using this one mod for all mods in the game that need Harmony provides some benefits:

- Avoids the need for each mod to distribute the Harmony DLL file with their mod.
- Ensures all mods are using the same version of Harmony. Different mods using different versions of Harmony could cause problems in the game.

For more information on the "DependencyPack" mod and its benefits, see:
https://mods.paradoxplaza.com/mods/77209/Windows

The following steps are adapted from the instructions for "Mod Developers" that are at the end of the "DependencyPack" mod's description on Paradox Mods. To install Harmony in your mod:

1. Install the Harmony package in your project:
   a) In Visual Studio menus, select "Tools" -> "NuGet Package Manager" -> "Manage NuGet Packages for Solution…". The "Manage Packages for Solution" screen is displayed.
   b) On the "Manage Packages for Solution" screen:
      i. Click on "Browse". A list of packages is displayed.
      ii. In the search box, enter "lib.harmony". Matching packages are listed.
      iii. Click on the "Lib.Harmony" package to highlight it (not "Lib.Harmony.Ref" or "Lib.Harmony.Thin"). The package's information is displayed on the right side.
      iv. Place a check mark next to your project name.
      v. For "Version", select "2.2.2". This is the version supported by the "DependencyPack" mod.
      vi. Verify the Author is "Andreas Pardeike".
      vii. Click on "Install".
      viii. If the "Preview Changes" screen is displayed, click on "Apply".
      ix. Wait for the installation to complete.
      x. Close the "Manage Packages for Solution" screen.
   c) Under the project, expand "Dependencies". Verify there is an entry for "Packages".
   d) Expand the "Packages" entry. Verify "Lib.Harmony (2.2.2)" is listed.
2. By default, the Harmony DLL file will be included with your mod. One of the benefits of the "DependencyPack" mod is that the Harmony DLL file does not need to be and should not be distributed with your mod. To exclude the Harmony DLL file from your mod:
   a) Click on the "Lib.Harmony (2.2.2)" package to show its properties.
   b) In the "Excluded Assets" property, enter "runtime".
   c) Rebuild the entire solution.
   d) Go to the game's local mods folder:
   C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\Mods
   e) In the AssemblyName folder, verify the "0Harmony.dll" file is not present.

3. Later steps for publishing your mod will add a dependency on the “DependencyPack” mod.

Language Support

CS2 comes with multiple languages built in. If desired, your mod can also support multiple languages. If your mod supports any additional languages, it is recommended that your mod support all of the languages built into the base game.

The detailed steps for including language support are too much to include here. See my Performance Monitor mod for an example of how to include language support in your mod with NO dependencies on other mods. See especially the “Localization” folder, the “UITranslationKey” class (two places), and the UI components in the “UI\src\components” folder where the translations are actually used.
https://github.com/rcav8tr/CS2Mod-PerformanceMonitor

Some authors get translations thru Crowdin. This author uses Google translate to translate from English to the other languages. For each text that needs to be translated:

1. Translate from English to the other language.
2. Use the “Swap languages” button (two arrows) to translate the other language back to English.
3. Verify that the same text as the original English is obtained.
4. If the translation back to English is different and unacceptable, try using synonyms or different English wording/grammar to get good translations into other languages.
5. When the translation is good, copy the translation from the other language and paste it into the “Localization\Translation.csv” file.

The mod title and description should be obtained from your translations for each language, not from the mod assembly info which is for only one language.

1. Open the “ModAssemblyInfo.csproj” file.
2. Remove the lines for “Title” and “Description”.
3. Save and close the “ModAssemblyInfo.csproj” file. Visual Studio automatically updates the “ModAssemblyInfo.cs” file.
4. In the “ModAssemblyInfo.cs” file, verify the title and description entries are removed.
5. Everywhere in your project that the mod assembly title or description are used, update to instead use your translations:
```
string text = Translation.instance.Get(UITranslationKey.<key>);
```

BepInEx Console

Log messages can be a tool for debugging your mod. By default, there is no console to view log messages in real time. Log messages are viewed by opening a log file (see section “Log File Locations”). The Bepis Injector Extendable (BepInEx) includes a console to view log messages in real time.

If you want to use the BepInEx console:

1. Download the BepInEx x64 zip file from the releases page:
   https://github.com/BepInEx/BepInEx/releases
   This author uses version 5.4.22 (scroll down a bit to find this version).
2. Extract the zip file contents into the game's root folder:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II
3. There should now be a “BepInEx” folder in the game’s root folder.
4. Start the game to the main menu and exit the game to desktop. This will cause BepInEx to create a configuration folder and a default configuration file.
5. Open the BepInEx configuration file in a text editor:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II\BepInEx\config\BepInEx.cfg

6. Under “[Logging.Console]”, change “Enabled” to “true”. This causes the BepInEx logging console to be displayed when the game is run.
7. Save and close the configuration file.
8. Test the BepInEx console:
   a) Start the game to the main menu.
   b) Verify that the BepInEx console is displayed.
      i. If you run the game in full screen mode, you may need to use Alt+Tab to view the BepInEx console window.
      ii. While developing a mod, this author prefers to run the game in windowed mode so that the BepInEx console window is easily visible.
   c) In the BepInEx console, verify that your mod’s “OnLoad” log message is displayed.
9. By default, Windows sets the BepInEx console window size and position. The position is different each time the game is run. To set your own BepInEx console window size and position:
   a) Right-click on the BepInEx console window title bar and select “Properties”. The window’s properties are displayed.
   b) Click on “Layout”. The window’s layout properties are displayed.
   c) Remove the check mark for “Let system position window”.
   d) Adjust the window size and position to your preference. Size is in characters. Position is in pixels. This author uses size 150 x 40.
   e) Click on “OK”. The window’s size and position are changed and remembered by Windows.
   f) Exit the game to desktop.
   g) Start the game to the main menu.
   h) Verify the BepInEx console window is displayed with your size and position settings.
10. Exit the game to desktop.

If you want to run the game with BepInEx but without the BepInEx console:
1. Open the BepInEx configuration file in a text editor:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II\BepInEx\config\BepInEx.cfg
2. Under “[Logging.Console]”, change “Enabled” to “false”. This prevents display of the BepInEx logging console when the game is run.
3. Save and close the configuration file.

If you want to keep your BepInEx installation but run the game without BepInEx:
1. Rename the BepInEx folder by placing a period in front of the “BepInEx” folder name:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II\.BepInEx
2. To run the game again later with BepInEx, remove the period.

If you want to uninstall BepInEx:
1. Delete the “BepInEx” folder from the game’s installation folder:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II
2. From the game’s installation folder, also delete the following files that were installed with BepInEx:
   changelog.txt
   doorstop_config.ini
   winhttp.dll

Log File Locations

Log files may be useful to help find where errors are occurring in your mod or in the game. Many log files are created. In the following, “<CS2DataFolder>” means:
C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II

Log file locations:

1. Game main log files (current and previous):
   <CS2DataFolder>\Player.log
   <CS2DataFolder>\Player-prev.log
2. Game other log files:
   <CS2DataFolder>\Logs\Discord.log
   <CS2DataFolder>\Logs\FileSystem.log
   <CS2DataFolder>\Logs\InputManager.log
   <CS2DataFolder>\Logs\Localization.log
   <CS2DataFolder>\Logs\Modding.log
   <CS2DataFolder>\Logs\PdxSdk.log
   <CS2DataFolder>\Logs\Radio.log
   <CS2DataFolder>\Logs\SceneFlow.log
   <CS2DataFolder>\Logs\UI.log
   (may be more from the game)
3. Your mod log file:
   <CS2DataFolder>\Logs\AssemblyName.log
   (may be more from other mods)
4. BepInEx log file (if BepInEx was installed):
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II\BepInEx\LogOutput.log

Developer Modes

The game includes two developer modes that may be useful while developing mods.

Adjust the game's launch options to activate the two developer modes:

1. Open the Steam app.
2. Select Cities: Skylines II game.
3. Click on "Manage" (gear) and then "Properties". The game's properties are displayed, defaulted to the "General" properties.
4. On the "General" tab, change "Launch Options" to include the following launch options after the executable:
   `-developerMode -uiDeveloperMode %command%`
5. The "%command%" launch option allows the desktop icon for the game to launch the game directly without first displaying the Paradox launcher where you need to click the "Play" button.
6. Close the properties.

The game's developer mode screen is activated by the "-developerMode" launch option. Test developer mode:

1. Start the game to the main menu.
2. Press the "Tab" key. Verify that the developer mode screen is displayed.
3. The developer mode screen provides access to some of the game's internal workings.
4. Press "Tab" again to close the developer mode screen.
5. For more information on developer mode, see:
   https://cs2.paradoxwikis.com/Developer_mode

The game's UI developer mode is activated by the "-uiDeveloperMode" launch option. Test UI developer mode:

1. With the game started, in a browser, go to the following URL to access the game's UI developer mode page:
   http://localhost:9444
   The "Inspectable pages" screen is displayed.
2. On the "Inspectable pages" screen, click on the "Coherent Labs" icon. The "DevTools" page is displayed. The "DevTools" page is available regardless of whether or not your mod has a UI.

3. Click on “Elements”.
    a) Drill down into the game’s UI structure. This may help you determine where your mod may need to make changes to the game’s UI.
    b) Hover your cursor over an element to highlight the element in the game. This is helpful to make sure you are finding exactly the right element.
    c) You can change an existing element’s attributes to see in real time what the change looks like in the game. For example, change an element’s size, position, color, or other attributes. This is helpful for fine tuning the UI for your own mod instead of continually making changes to the UI code, recompiling, and restarting the game to see the effect.
4. If your mod includes a UI:
    a) Click on “Console”.
    b) Verify your mod’s “Hello UI!” message is displayed. This console message was generated from your mod’s default UI code at “UI\src\index.tsx”, which includes “HelloWorldComponent”.
    c) You can write debug messages to the UI console from your UI code (using console.log method) and see the results on the “DevTools” page. Remember to remove debug logging later.
5. The “DevTools” page can be left active in your browser while exiting and restarting the game during development:
    a) After exiting the game, the “DevTools” page will show a warning that the connection was closed. Leave the warning message displayed!
    b) After restarting the game, click on the “Reconnect DevTools” button to reconnect to the game. Reconnecting also clears the console.

Continue Last Save

While developing your mod, you will likely be exiting and restarting the game often. Each time you start the game to test your mod, you need to start a new game or load a saved game. To save yourself the step of loading a saved game, add the following game launch option to instruct the game to automatically load the last saved game:

```
-continuelastsave
```

See section “Developer Modes” for the steps to change the game’s launch options. For unknown reasons, the continue last save feature does not always work. In that case, just load a saved game manually.

Visual Studio Debugging

Much debugging and analysis can be accomplished by just logging messages from your mod. This section describes the steps to use Visual Studio for more in depth debugging. These steps are adapted from Paradox Wiki:
https://cs2.paradoxwikis.com/Debugging

Add the Unity workload to Visual Studio:

1. Close Visual Studio if it is open.
2. In Windows, use “Apps” to modify your Visual Studio installation. The “Visual Studio Installer” screen is displayed and within that screen the “Modifying” window is displayed defaulted to the “Workloads” tab.
3. On the “Workloads” tab, under the “Gaming” heading, find the entry for “Game development with Unity”. If it is already checked, you already have the Unity workload installed. If it is not checked:
    a) Check “Game development with Unity”. The installation details on the right will be updated.
    b) Click on “Modify” to install the workload.
    c) Wait for the installation to complete.
4. Close the “Modifying” window and the “Visual Studio Installer” screen.

Configure Unity debugging:

1. In Windows, use File Explorer go to the Unity editor directory. If you installed Unity in a different directory, adjust as needed:
   C:\Program Files\Unity <version>\Editor\Data\PlaybackEngines\windowsstandalonesupport\Variations\win64_player_development_mono
   where <version> is the Unity version.
2. Copy the "UnityPlayer.dll" file.
3. Go to the CS2 game directory:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II
4. Rename the existing "UnityPlayer.dll" file to "UnityPlayerOriginal.dll". This is so you can easily restore the original file later.
5. Paste the "UnityPlayer.dll" file in the CS2 game directory.
6. Go to the CS2 data directory:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II\Cities2_Data
7. Edit the "boot.config" file in a text editor (e.g. Notepad).
8. Add the following line to the end of the file:
   ```
   player-connection-debug=1
   ```
9. Save and close the "boot.config" file.

Attach Unity debugger in Visual Studio:

1. Start the game normally via Steam. The Paradox Wiki notes that starting from other than Steam might prevent the debugger from working.
2. Verify that the game screen shows "Development Build" in the lower right corner. This is due to the "UnityPlayer.dll" file that was copied.
3. Open your project in Visual Studio.
4. On the Visual Studio menus, select "Debug" -> "Attach Unity Debugger". The "Select Unity Instance" window is displayed.
5. On the "Select Unity Instance" window, click on the "Cities: Skylines II" entry to highlight it (hopefully it is the only entry). Click on "OK".
6. You can now set breakpoints and do other debugging tasks on your mod in Visual Studio.
7. To stop debugging, click on "Stop debugging" on the toolbar or in the Visual Studio menus click on "Debug" -> "Stop debugging".
8. If you exit the game to desktop with the debugger attached, the debugger will stop automatically.

To see how a breakpoint works with the debugger:

1. Exit the game to desktop.
2. In your project in Visual Studio, edit "Mod.cs" to add the following class after the "Mod" class:
   ```
   partial class TestSystem : Game.UI.UISystemBase
   {
       protected override void OnUpdate()
       {
           base.OnUpdate();
       }
   }
   ```
3. Set a breakpoint on the "base.OnUpdate();" line.
4. In the "Mod" class, at the end of the "OnLoad" method, add the following:
   ```
   updateSystem.UpdateAt<TestSystem>(SystemUpdatePhase.UIUpdate);
   ```
5. Build the solution.
6. Start the game to the main menu.
7. In Visual Studio, attach the Unity debugger as described above.
8. Verify that the breakpoint is hit. The mod's TestSystem.OnUpdate runs even in the game's main menu.
9. In the game, notice that the UI is now frozen because of the break point.
10. In Visual Studio, click on "Continue". Notice that the breakpoint is immediately hit again because the OnUpdate method gets called every frame.
11. Remove the breakpoint and click on "Continue".

12. In the game, notice that the UI is now responsive again.
13. Stop the debugger and exit the game to desktop.
14. Remove the code from “Mod.cs” that was added for testing breakpoints.

If a new version of Unity is installed later, remember to get the corresponding “UnityPlayer.dll” for that version.

View Game Source Code

When creating a mod, it is often useful to view the game’s source code to see what the game is doing, where your mod needs to make changes, or to find some example code to adapt for use in your own mod. The ILSpy extension in Visual Studio provides a way to view the game’s source code.

To install the ILSpy extension in Visual Studio:

1. In Visual Studio, in the menus, click on “Extensions” -> “Manage Extensions…”. The “Manage Extensions” screen is displayed.
2. On the “Manage Extensions” screen, search for “ilspy”. The search results are displayed.
3. In the search results, click on “ILSpy 2022” or the latest version and click on “Download”. The download is scheduled to run later.
4. Close the “Manage Extensions” screen.
5. Close Visual Studio. The “VSIX Installer” screen is displayed.
6. On the “VSIX Installer” screen, click on “Modify” to install ILSpy. The extension gets installed.
7. Wait for the modifications to be complete. The “Modifications Complete” screen is displayed.
8. Close the “Modifications Complete” screen.

To view the game’s source code in ILSpy:

1. Open your mod project in Visual Studio.
2. Under the project, expand “Dependencies” -> “Assemblies”.
3. Right-click on “Game” and select “Open in ILSpy”. ILSpy opens pointing to the “Game.dll” file.
4. Use ILSpy to expand the Game dll or use the search capabilities in ILSpy.
5. Close ILSpy.

Get Game Source Code

Sometimes it is easier to see view or search for source code if the game’s source code is in Visual Studio.

To get the source code for the “Game.dll” file (assumes ILSpy extension is installed, see section “View Game Source Code”):

1. In Windows, either:
   a) Delete all existing folders and files from the folder where the files will be stored.
   b) Create a new folder to hold the files. Author uses:
      C:\Users\<name>\Documents\Visual Studio Projects\Cities Skylines 2 Mods\Base Game
2. View the game source code in ILSpy as described in section “View Game Source Code”.
3. In ILSpy:
   a) Right-click on the “Game” assembly and select “Save Code…”. A “Save As” dialog is opened.
   b) Select the folder where the files should be stored.
   c) Click on “Save”.
   d) Wait for the export to complete. The chosen folder should now have a “Game.csproj” file along with folders and source code files that were created from the “Game.dll” file.
   e) Close ILSpy.
4. Configure the game source code project:
   a) Open the “Game.csproj” file in Visual Studio.
   b) TBD More steps to create a solution and define dependencies.

Publish New Mod To Paradox Mods

After you make your mod do what your mod must do, you probably will want to publish your mod to Paradox Mods so that other users can subscribe to your awesome mod. The following sections describe the steps to publish your new mod to Paradox Mods for the first time.

Remove Unneeded Code

If settings code (with or without key bindings) was initially included but your mod does not need settings now, remove the settings code:

1. Delete the "ModSettings" folder from the project.
2. Edit the "Mod.cs" file:
   a) Remove all code related to mod settings.
   b) Remove unused "using" statements.
   c) Save and close the "Mod.cs" file.
3. Remove any other code related to settings from everywhere in your project.

If UI code was included but your mod does not need a UI now, remove the UI code:

1. In the Visual Studio menu, click on "Project" -> "Edit Project File". The project file is opened.
2. At the end of the first "PropertyGroup", delete the following entry that was previously added. If you specified other items to exclude, then delete only the "UI\node_modules**;" portion and leave the rest.
   `<DefaultItemExcludes>UI\node_modules\**;$(DefaultItemExcludes)</DefaultItemExcludes>`
3. At the bottom of the file, delete the two sets of entries that were previously added for automatically building the UI with the project and for automatically building the "mod.json" file.
4. Delete the "UI" folder from the project.
5. Remove any other code related to UI from everywhere in your project.

If Harmony was included but your mod does not need Harmony now, remove Harmony:

1. In Visual Studio under the project, expand "Dependencies" and "Packages".
2. Right-click on the "Lib.Harmony (2.2.2)" package and click on "Remove".
3. Remove any other code related to Harmony from everywhere in your project.

Update Project Properties

Update project properties:

1. In the Visual Studio menu, click on "Project" -> "AssemblyName Properties". The project properties are opened.
2. Under "Package" -> "General", set "Package Version" for your initial release:
   a) This author prefers "1.0.0" for the initial release.
   b) Some other mod authors use version "0.m.p" (where "m" is minor version number and "p" is patch number) for their initial release and subsequent releases until they determine that the mod is no longer in testing or release candidate status.
3. Make sure the "Title", "Description", and "Copyright" properties are correct.
4. Save and close the project properties.

Prepare Publishing Files

The mod template includes a default thumbnail image in the "Properties" folder in the "Thumbnail.png" file. This default image should be replaced so your mod has its own unique thumbnail. Create/update a new thumbnail image:

1. Use your own image creation/editing tools to create a new thumbnail image.
2. Your thumbnail image should be 500 x 500 pixels even though the default thumbnail image is 950 x 500 pixels. A bit larger or smaller is okay, but it should be square.

3. If the game is played at 1920 x 1080 (i.e. the most common display resolution), then on the "Paradox Mods" screen of the game, the thumbnail will be scaled down to about 220 x 220 (i.e. about 44%) on the "Browse" tab and about 128 x 128 (i.e. about 26%) when viewing the individual mod. The thumbnail will be larger or smaller if the game is played at a higher or lower resolution. As you are creating your thumbnail image, you can try viewing it at these smaller sizes to verify that it looks okay.
4. Keep the same file name and image type "Thumbnail.png".
5. Replace the default file in the "Properties" folder with your own file.

Screenshots are optional but recommended to show potential users what your mod will look like in the game. A picture of your mod in action is worth a lot of words to describe it. To add screenshots to your published mod:

1. If you set up Visual Studio debugging, restore the original "UnityPlayer.dll" file so that the "Development Build" text is not captured in your screenshot:
   a) Exit the game to desktop.
   b) Go to the CS2 game directory:
   C:\Program Files (x86)\Steam\steamapps\common\Cities Skylines II
   c) Rename "UnityPlayer.dll" to "UnityPlayerDevelopment.dll".
   d) Rename "UnityPlayerOriginal.dll" to "UnityPlayer.dll".
   e) You can switch back and forth between original and development by renaming the files.
2. Under the project, in the "Properties" folder, create a new folder named "Screenshots".
3. Start the game to capture screenshots of your mod in action.
4. For each screenshot:
   a) The Windows Snipping tool can be used to take a screenshot. The Snipping tool can be activated with WindowsKey+Shift+S. The Snipping tool places the snip in the Windows clipboard.
   b) To make changes to the snip, use the Snipping tool's Snip & Sketch tool or paste the snip into an image editor of your choice.
   c) On Paradox Mods (both in game and web site), the screen shots are displayed in an area that has a width to height ratio of 1.778 (aka 16 x 9).
      i. If you want your screenshot to fill that area without any blank space on the top/bottom or left/right sides, then your screenshot must also have this ratio.
      ii. It is okay if your screenshot is not this ratio. Paradox Mods will automatically scale your image to fit in the area, but with blank space where needed.
      iii. Your screenshot does not need to be the whole game screen. Your screenshot can be only part of the game screen. If you want your partial screenshot to fill the display area on Paradox Mods, then it still needs to be this ratio.
      iv. Here some standard screen widths and the corresponding height needed to achieve this ratio:
      2560 x 1440
      1920 x 1080 (most common)
      1600 x 900
      1280 x 720
      1024 x 576
      800 x 450
   d) Save the screen shot to the new "Screenshots" folder:
      i. The file name should be a 2-digit number representing the screen shot order you want on Paradox Mods (e.g. "10", "20", etc.) followed by a descriptive name of the screenshot so you know what it is. Include no spaces in the file name. Counting by 10's now allows you to later easily add new screenshots in between without needing to rename existing files (e.g. "05", "15", "25", etc.). (TBD Need to confirm order on Paradox Mods is based on file name sorting.)
      ii. The file format should be png.
      iii. The file size should be less than 2.1MB. If a file is too large, resize it to a smaller resolution or capture only part of the screen. Note that a full window screenshot at 1920 x 1080 might be about 3.5MB to 4.0MB.

e) In the properties for the screenshot file, verify “Build Action” is “None” and “Copy to Output Directory” is “Do not copy”. Set if needed.

5. Exit the game to desktop.
6. In the Visual Studio menu, click on “Project” -> “Edit Project File”. The project file is opened.
7. In the project file:
   a) Delete the “ItemGroup” for the “Screenshots” folder:
   ```
   <ItemGroup>
       <Folder Include="Properties\Screenshots\" />
   </ItemGroup>
   ```
   b) Save and close the project file.

The Read Me file will be used as the long description on Paradox Mods and will be displayed by default in GitHub. Create a Read Me file:

1. Under the project, add a new text file named “ReadMe.md”. The file is created and opened.
2. The file uses Markdown for formatting. See section “Working With Markdown”.
3. The standard based on looking at other mods seems to be to start with the mod title as a heading followed by the short description. Add the following to the “ReadMe.md” file:
   ```
   # <title>

   <description>
   ```
   where <title> and <description> are the assembly “Title” and “Description”.
4. Below that, add your full description of the mod.
5. In the menus, select: “Edit” -> “Advanced” -> “Set End of Line Sequence”. If “CRLF” is not checked, click on it now.
6. Save and close the “ReadMe.md” file.
7. In the properties for the “ReadMe.md” file, verify “Build Action” is “None” and “Copy to Output Directory” is “Do not copy”. Set if needed.

The Change Log file will be used as the change log on Paradox Mods. Create a Change Log file:

1. Under the project, add a new text file named “ChangeLog.md”. The file is created and opened.
2. The file uses Markdown for formatting. See section “Working With Markdown”.
3. Add one unordered list item to the file with the following text:
   ```
   - Initial release.
   ```
4. Paradox Mods will automatically display the change log with a heading that includes the mod version and release date, so there is no need to include those in your change log.
5. In the menus, select: “Edit” -> “Advanced” -> “Set End of Line Sequence”. If “CRLF” is not checked, click on it now.
6. Save and close the “ChangeLog.md” file.
7. In the properties for the “ChangeLog.md” file, verify “Build Action” is “None” and “Copy to Output Directory” is “Do not copy”. Set if needed.

The “PublishConfigutation.xml” file provides information about your mod to Paradox Mods. Automatically create the “PublishConfiguration.xml” file from assembly information:

1. Copy the “PublishConfiguration.csproj” file from GitHub https://github.com/rcav8tr/CS2-Modding
2. In Visual Studio, paste the file to the “Properties” folder.
3. Edit the “PublishConfiguration.csproj” file just added:
   a) If your mod requires a specific game version:
      i. Find the “GameVersion” entry.
      ii. Update the entry to your required game version. Use an asterisk (*) to accept any version in the position (e.g. “1.*”, “1.1.*”, “1.2.*”, etc).
   b) If using Harmony in the mod:

i. Find the “Dependency Id” entry.
ii. Delete the blank dependency entry.
iii. Add a new entry for the “DependencyPack” mod:
```
<Dependency Id="77209" />     <!-- DependencyPack (i.e. Harmony) -->
```
c) In the “ExternalLink” entry for GitHub:
i. Replace “<GitHubUsername>” with your GitHub username.
ii. The entry assumes your GitHub repository name is your AssemblyName with a “CS2Mod-” prefix. This is the author’s preference. Update as needed.
d) Add any other entries for external links. See the project file for the list of supported types.
e) Save and close the “PublishConfiguration.csproj” file.
4. In the Visual Studio menu, click on “Project” -> “Edit Project File”. The project file is opened.
5. Near the end of the file, find and delete the “ItemGroup” entry for the “PublishConfiguration.csproj” file that was automatically added by Visual Studio:
```
<None Include="Properties\PublishConfiguration.csproj" />
```
6. Add the following before the “</Project>” tag:
```
<!-- Automatically build PublishConfiguration.xml file. -->
<Import Project="Properties\PublishConfiguration.csproj" />
<ItemGroup>
    <None Include="Properties\PublishConfiguration.csproj" />
</ItemGroup>
```
7. Save the project file. As soon as the project file is saved, Visual Studio immediately builds the “PublishConfiguration.xml” file, overwriting the existing file.
8. Verify the “PublishConfiguration.xml” file:
a) Go to the “Properties” folder of the project:
b) Open the “PublishConfiguration.xml” file and verify its contents.
c) If you have screenshots, verify the screenshot file names are correct and in the correct order.
d) Close the file.

Prepare Solution

Rebuild the solution:
1. Change the build configuration from “Debug” to “Release”.
2. Rebuild the entire solution.
3. Test the mod one last time before publishing.

Publish To Paradox Mods

Publish the mod to Paradox Mods:
1. Mods are published from Visual Studio, not from within the game.
2. In Visual Studio, right-click on the project and select “Publish…”. The “Publish” screen is displayed.
3. On the “Publish” screen:
a) In the drop-down at the top, select “PublishNewMod.pubxml”.
b) Click on “Publish”. Publishing is started, including a rebuild of the solution. Wait for publishing to complete.
c) Verify the “Publish succeeded …” message is displayed. For any error, see Visual Studio’s Output log, correct the error, and try the publish again.
d) Close the “Publish” screen.

Verify Published Mod

Remove the local copy of the mod so it does not interfere with the published mod:
1. In Windows, go to the game’s local mods folder:
C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\Mods
2. Delete the folder for your mod.

Verify the mod on Paradox Mods:

1. Start the game to the main menu.
2. On the main menu, click on “Paradox Mods”. The “Paradox Mods” screen is displayed.
3. On the “Paradox Mods” screen:
   a) Click on “Playsets” at the top. The “Playsets” screen is displayed.
   b) On the “Playsets” screen, create a new Playset or activate an existing Playset to which your mod will be added for testing.
   c) Click on “Browse” at the top. The “Browse” screen is displayed.
4. On the “Browse” screen:
   a) Under “Filter” and “Tags”, check the “Code Mod” tag. This avoids finding maps and other assets.
   b) In the “Search mods…” box, search for your mod by name. A list of matching mods is displayed.
   c) Locate your mod and click on it to display its properties. Your mod’s properties are displayed.
   d) On the screen for your mod’s properties:
      i. Verify all the screenshots are available and in the correct order.
      ii. Verify the long description is correct and is formatted as desired.
      iii. Verify the thumbnail image is correct.
      iv. Verify the mod title is correct.
      v. Verify the author is correct.
      vi. Verify the Tags are correct. At a minimum, the “Code Mod” tag should be present.
      vii. Under the “More About This Mod” section, verify the external links (except GitHub) are correct by clicking on each one. The GitHub link won’t work until the solution is saved to GitHub below.
      viii. There will be no “ChangeLog” section for an initial release.
      ix. If any of the above are incorrect, then after completing these steps for a new mod, see section “Update Published Configuration To Paradox Mods”.
   e) Under the “Mod Info” section, make a note of the mod’s ID number. It will used in steps below.
   f) Click on “Add to Active Playset”. Wait for the installation to complete.
   g) Close the Paradox Mods screen.
   h) Exit the game to desktop.
5. Verify the mod is present in the game’s folder of subscribed mods:
   C:\Users\<name>\AppData\LocalLow\Colossal Order\Cities Skylines II\.cache\Mods\mods_subscribed\<ModID>
   where <ModID> is the mod’s ID obtained above followed by an underscore and another number. This second number seems to be the number of times the mod was updated on Paradox Mods.
6. Start the game to the main menu.
7. Start a new game or load a saved game.
8. Verify the subscribed mod works as expected.
9. If the mod does not work as expected, then after completing these steps for a new mod, see section “Publish New Version To Paradox Mods”.
10. Exit the game to desktop.

Update Mod ID

Update the mod ID in the publish configuration:

1. The mod ID is required to publish any future updates of the mod to Paradox Mods.
2. Edit the “PublishConfiguration.csproj” file.
3. Set the “ModId” value inside the quotes to the mod ID.
4. Save and close the project file. As soon as the project file is saved, Visual Studio immediately builds the “PublishConfiguration.xml” file, overwriting the existing file.
5. In the PublishConfiguration.xml file, verify the mod ID value is correct.

Save to GitHub

For more information on creating a Git repository from Visual Studio, see:
https://learn.microsoft.com/en-us/visualstudio/version-control/git-create-repository?view=vs-2022

Save the solution to GitHub:

1. On the Visual Studio menu, click on “Git” -> “Create Git Repository...”. The “Create a Git repository” screen is displayed.
2. On the “Create a Git repository” screen:
   a) Under “Push to a new remote”, click on “GitHub” to select it.
   b) Under “Initialize a local Git repository”:
      i. The “Local path” defaults to your local project folder. Leave as is.
      ii. The “.gitignore template” defaults to “Default (VisualStudio)”. Leave as is.
      iii. The “License template” defaults to “None”. Change it if desired. The “MIT License” is simple and is used by some CS2 mod authors. For more information on licenses, see: https://choosealicense.com/
   c) Under “Create a new GitHub repository”:
      i. The “Account” should default to your GitHub account. If not, click “Sign in” and select “GitHub Account”. A GitHub authorization web page is displayed. Click “Authorize github”. After the authorization completes, return to Visual Studio.
      ii. The “Owner” should default to your GitHub username. Leave as is.
      iii. The “Repository name” should default to your AssemblyName. Prefix your AssemblyName with “CS2Mod-” (author preference). This prefix quickly and easily identifies this repository as a CS2 mod among any other repositories you might have in your GitHub account.
      iv. The “Description” defaults to blank. Set this to “Cities Skylines 2 mod - ” followed by your mod’s assembly description.
      v. The “Private repository” defaults to checked. Uncheck it make the repository public.
   d) Under “Push your code to GitHub”, verify the URL is correct.
   e) Click on “Create and Push”.
      i. If notified that “Your solution contains files outside the solution folder …”, click “Yes” to continue anyway.
      ii. Wait for the repository to be created. It will take just a few seconds.
3. Return to the Solution Explorer and verify that all your source files indicate locked (i.e. not changed).
4. If a license was chosen, verify a “LICENSE.txt” file was added as a source file.

Update the GitHub repository:

1. In a browser, go to github.com and sign in into your account.
2. Verify the new repository for your mod is present.
3. Click on the repository for your mod. The repository is displayed defaulted to “<>Code”.
4. On the right side, to the right of “About”, click on the gear icon. The “Edit repository details” screen is displayed.
5. On the “Edit repository details” screen:
   a) Under “Topics”, add all the following entries:
      “cities-skylines2”
      “cities-skylines-2”
      “cities-skylines-2-mod”
   b) Under “Include in the home page”, leave all settings checked (i.e. the default).
   c) Click “Save changes”. GitHub returns to the “<>Code” page.
6. Verify all your source files are present.
7. Verify the contents of the “ReadMe.md” file are displayed below your source files with MarkDown formatting applied.
8. Under “master”, click on “Tags” and then “View All Tags”. An empty tags list is displayed.

9. Click on “Create a new release”. The “Create a new release” screen is displayed.
10. On the “Create a new release” screen:
    a) Click on “Choose a tag”. In the drop down, enter “v” followed by “M.m.p” where “M” is major version number, “m” is minor version number, and “p” is patch number (e.g. “v1.0.0”). Click “Create new tag: ...”.
    b) For “Release title”, enter “Original”.
    c) Enter a description of the release if desired. This author leaves description blank because the title alone is usually sufficient to describe the release.
    d) There are no binaries to attach.
    e) Uncheck “Set as a pre-release”.
    f) Click on “Publish release”. The release and tag are created.
11. Verify the release and tag were created.
12. Sign out of GitHub.

Verify the GitHub link on Paradox Mods:

1. Start the game to the main menu.
2. Click on “Paradox Mods”. The “Paradox Mods” screen is displayed.
3. Locate your mod and click on it to display its properties. Your mod’s properties are displayed.
4. Under the “More About This Mod” section, click on the GitHub link.
5. When asked to open a link in a web browser, click “Yes”.
6. Verify the link opens your GitHub repository in a browser.
7. Close the “Paradox Mods” screen.
8. Exit the game to desktop.

Mod Is Published

Your new mod is now published to Paradox Mods! Wait for CS2 users to find your awesome mod!

Be sure to monitor your mod’s comments for problems, errors, suggestions, etc. If you don’t want to monitor your mod’s comments from within the game, you can also access the Paradox Mods web site for CS2: https://mods.paradoxplaza.com/games/cities_skylines_2

Update Your Mod

Before starting to update your mod with added/changed/deleted features or bug fixes:

1. Change the project build configuration from “Release” back to “Debug”.
2. If using Visual Studio debugging, change to the development “UnityPlayer.dll” file.
3. Unsubscribe from your mod to avoid interference with the local mod during development:
    a) In the game, on the “Paradox Mods” screen, create a new empty Playset just for testing. You might name it something like “Empty” or “No Mods”. The new Playset is activated by default.
    b) Now you can make changes to your mod locally without the subscribed instance of your mod interfering.

Now go update your mod to make your mod do what your mod must do! When you are ready to publish your updated mod to Paradox Mods, continue with section “Publish New Version To Paradox Mods”.

Publish New Version To Paradox Mods

This section describes the steps to publish your mod to Paradox Mods after it’s functionality is changed. If only the publish configuration and/or code comments are changed, then see section “Update Published Configuration To Paradox Mods”.

Update Project Properties

Update project properties.

1. In the Visual Studio menu, click on “Project” -> “AssemblyName Properties”. The project properties are opened.
2. Under “Package” -> “General”, update “Package Version” as needed. Paradox Mods expects a new version number. Click on the question mark to get Microsoft best practices for “Package Version”.
3. Update the “Title”, “Description”, and “Copyright” properties if needed.
4. Save and close the project properties.

Update Publishing Files

Update thumbnail file:

1. If needed, update the “Properties\Thumbnail.png” file.

Update screenshots:

1. Update existing screenshot files where needed.
2. Delete the files for any screenshots no longer needed from the “Properties/Screenshots” folder.
3. Add any new screenshots using the procedure described above for a new mod.

Update the “ReadMe.md” file.

1. Edit the “ReadMe.md” file.
2. Verify the title and description at the top of the “ReadMe.md” file matches the title and description from your assembly info.
3. Make any other changes needed to the file.
4. Save and close the “ReadMe.md” file.

Update the “ChangeLog.md” file:

1. Edit the “ChangeLog.md” file.
2. The file lists only the changes since the previous version (i.e. not a running log of all changes):
   a) Delete the entire contents of the file.
   b) Add a bulleted list of changes where each bullet (markdown: -) is a short description of the change.
   c) Paradox Mods will automatically display the change log with a heading that includes the mod version and release date, so there is no need to include those in your change log.
3. Save and close the “ChangeLog.md” file.

Update the “PublishConfiguration.xml” file:

1. Delete the “PublishConfiguration.xml” file. Visual Studio will automatically rebuild the file.
2. Review the “PublishConfiguration.xml” file.
3. If any changes are needed for entries that are not automatically populated, edit the “PublishConfiguration.csproj” file and make the changes there.

Review Changes

Review all changes before publishing:

1. In Visual Studio, on the menu, click “View” -> “Team Explorer”. The “Team Explorer” tab or screen is displayed.
2. In “Team Explorer”, click on “Git Changes”. The “Git Changes” tab or screen is displayed.
3. On “Git Changes”, review every change in every changed file to make sure nothing unexpected was changed.
4. Close “Team Explorer”.

Prepare, Publish, Verify Mod

Prepare, publish, and verify the mod using the same steps as in sections “Prepare Solution”, “Publish To Paradox Mods”, and “Verify Published Mod” for a new mod, except:

1. On the “Publish” screen, in the drop-down at the top, select “PublishNewVersion.pubxml”.

Update GitHub

Update the solution on GitHub:

1. In Visual Studio, return to “Git Changes”:
   a) In the message box, enter “Version” followed by space and “M.m.p” where “M” is major version number, “m” is minor version number, and “p” is patch number (e.g. “Version 1.2.3”).
   b) Click on the drop down next to “Commit All” and select “Commit All and Push”. Wait for it to complete.
2. Return to the solution explorer and verify that all the files indicate locked (i.e. not changed).
3. In a browser, go to github.com:
   a) Sign in into your GitHub account.
   b) Click on the repository for your mod. The repository is displayed defaulted to “<>Code”. Verify your changed source files are updated.
   c) Under “master”, click on “Tags” and then “View All Tags”. The list of tags is displayed.
   d) Click on “Releases”. The list of releases is displayed.
   e) Click on “Draft a new release”. The “Draft a new release” screen is displayed.
   f) On the “Draft a new release” screen:
      i. Click on “Choose a tag”. In the drop down, enter “v” followed by the same “M.m.p” as above for Version (e.g. “v1.2.3”). Click “Create new tag: <tag> on publish”.
      ii. For “Release title”, enter a short description of the changes made to the mod in this version.
      iii. Enter a description of the release if desired. This author leaves description blank because the title alone is usually sufficient to describe the release.
      iv. There are no binaries to attach.
      v. Uncheck “Set as a pre-release”.
      vi. Check “Set as the latest release”.
      vii. Click on “Publish release”. The release and tag are created.
   g) Verify the release and tag were created.
   h) Sign out of GitHub.

Update Published Configuration To Paradox Mods

This section describes the steps when only your mod configuration on Paradox Mods needs to be updated and/or only comments were updated in your source code. Sometimes it is desirable to update only comments in the source code and not wait until the next functional release before updating GitHub. If any functionality or assembly information (e.g. title, description, copyright, version, etc.) has changed, then see section “Publish New Version To Paradox Mods”.

While it is possible to update your published configuration on the Paradox Mods web site, it is recommended to make all your updates in your Visual Studio source files and then publish from Visual Studio. This keeps your Visual Studio project, your GitHub repository, and your Paradox Mods published configuration all in sync with each other.

Update Publishing Files

Update publishing files using the same steps as in section “Update Publishing Files” for a new version.

Review Changes

Review all changes before publishing using the same steps as in section “Review Changes” for a new version. There should be changes only to publishing files and comments.

Prepare, Publish, Verify Mod

Prepare, publish, and verify the mod using the same steps as in sections “Prepare Solution”, “Publish To Paradox Mods”, and “Verify Published Mod” for a new mod, except:

1. On the “Publish” screen, in the drop-down at the top, select “UpdatePublishedConfiguration.pubxml”.

Update GitHub

Update the solution on GitHub using the same steps as in section “Update GitHub” for a new version, except:

1. Modify the “Version M.m.p” and “vM.m.p” release and tag labels (this is author’s preference):
   a) From GitHub for the repository, display all releases.
   b) Note the latest release version and whether or not the version has a trailing letter. The latest version might not be first in the list.
   c) At the end of the two version labels, append the next single letter a-z not previously used by the latest release on GitHub. For example, if the latest release is “v1.2.3”, then the two labels would be “Version 1.2.3a” and “v1.2.3a”. If the latest release is “v2.3.1c”, then the two labels would be “Version 2.3.1d” and “v2.3.1d”.
2. For “Release title” enter one of the following depending on what was changed:
   a) “Update Only Published Configuration”
   b) “Update Only Comments”
   c) “Update Only Published Configuration and Comments”

Working With Markdown

The ReadMe.md and ChangeLog.md files use Markdown (md) for text formatting.

A note on the Modding Toolchain web site indicates that only headings (#, ##, ###), bullet lists (-), and bold text (**text**) are supported. TBD But may want to try other markdown anyway.

Markdown references:

- https://www.markdownguide.org/cheat-sheet/
- https://commonmark.org/help/

Markdown testing tool where you can paste your markdown and see what it will look like when formatted:

- https://spec.commonmark.org/dingus/

***