T4-Permissions: Overview

By Taylor Love

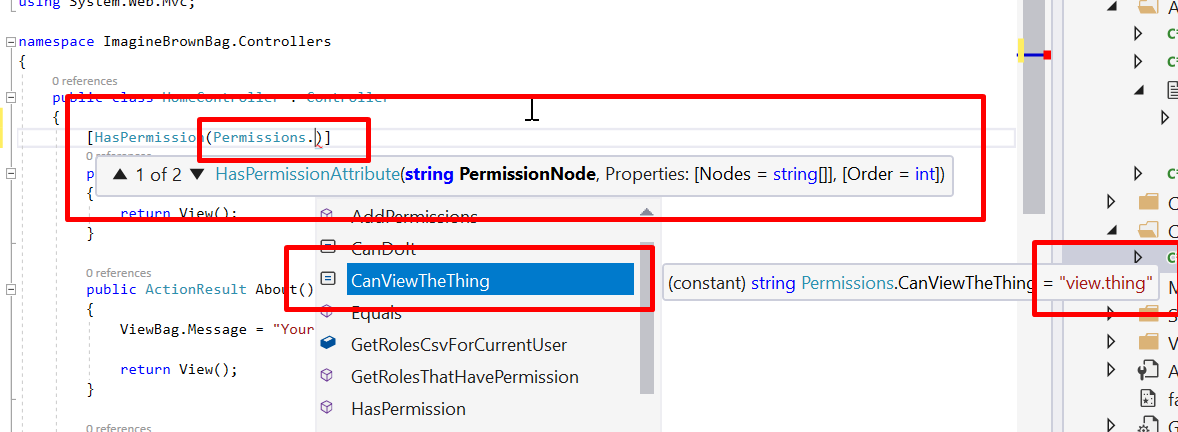
Introduction:

T4-Permissions is a permissions system that allows developers to manage expanding complexity with ease. It is a solution that was created to simplify role-based authentication and give more control over individual permissions. It makes it easy to add new roles, or see what each role is capable of; all from viewing and editing a single configuration file.

Requirements for this article are: an understanding of C#, and preferably a good understanding of ASP.NET and MVC as well.

Before we get started in-depth here is a short feature list for T4-Permissions:

* Central configuration file makes code easy to maintain and review
* T4 speeds up development time for updates and adds IDE ready auto-complete features based on the generated system.
* Generates filter attributes for ASP.NET controllers.
* Generates filter attributes for Web API controllers.
* Generates HtmlHelper extension method for use in cshtml view pages
* O(1) look-up time once compiled
* Uses an Enum pattern so developers can take advantage of autocomplete in their IDE’s.



Setup:

**Extension**s: [Devart T4 Editor](https://marketplace.visualstudio.com/items?itemName=DevartSoftware.DevartT4EditorforVisualStudio) will give syntax highlighting to your Visual Studio IDE.

**IDE:** [Visual Studio 2017](https://visualstudio.microsoft.com/downloads/?utm_medium=microsoft&utm_source=docs.microsoft.com&utm_campaign=button+cta&utm_content=download+vs2017) Community or better

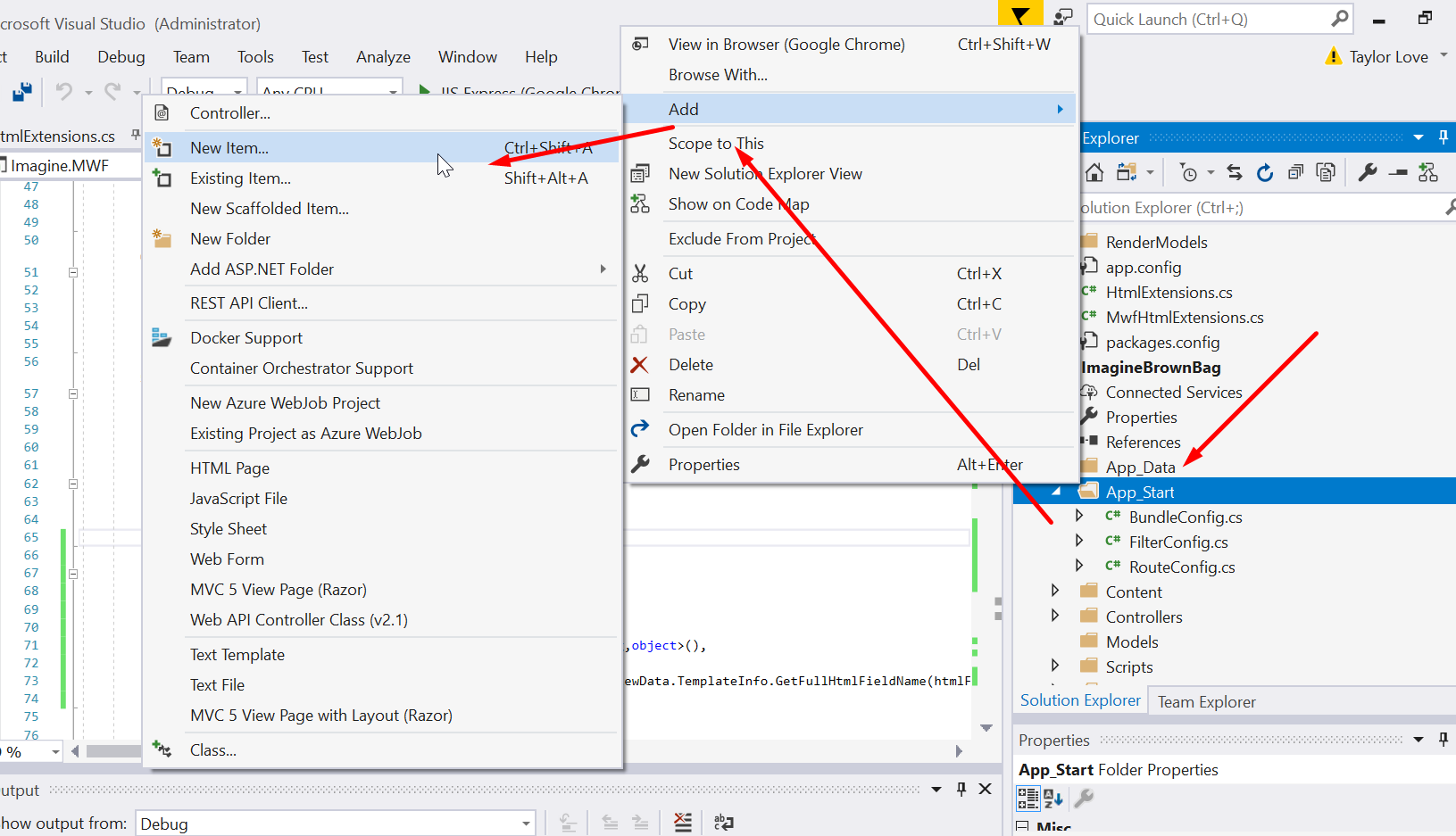
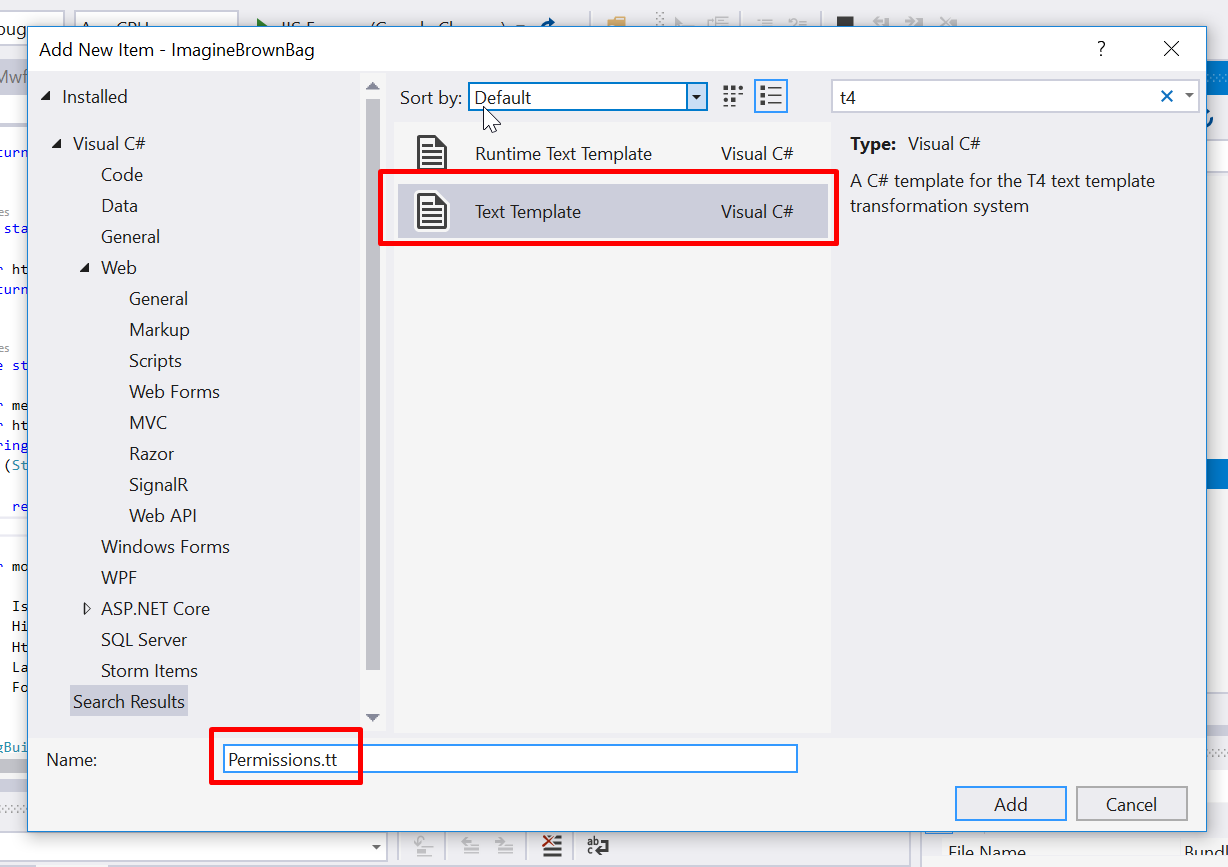
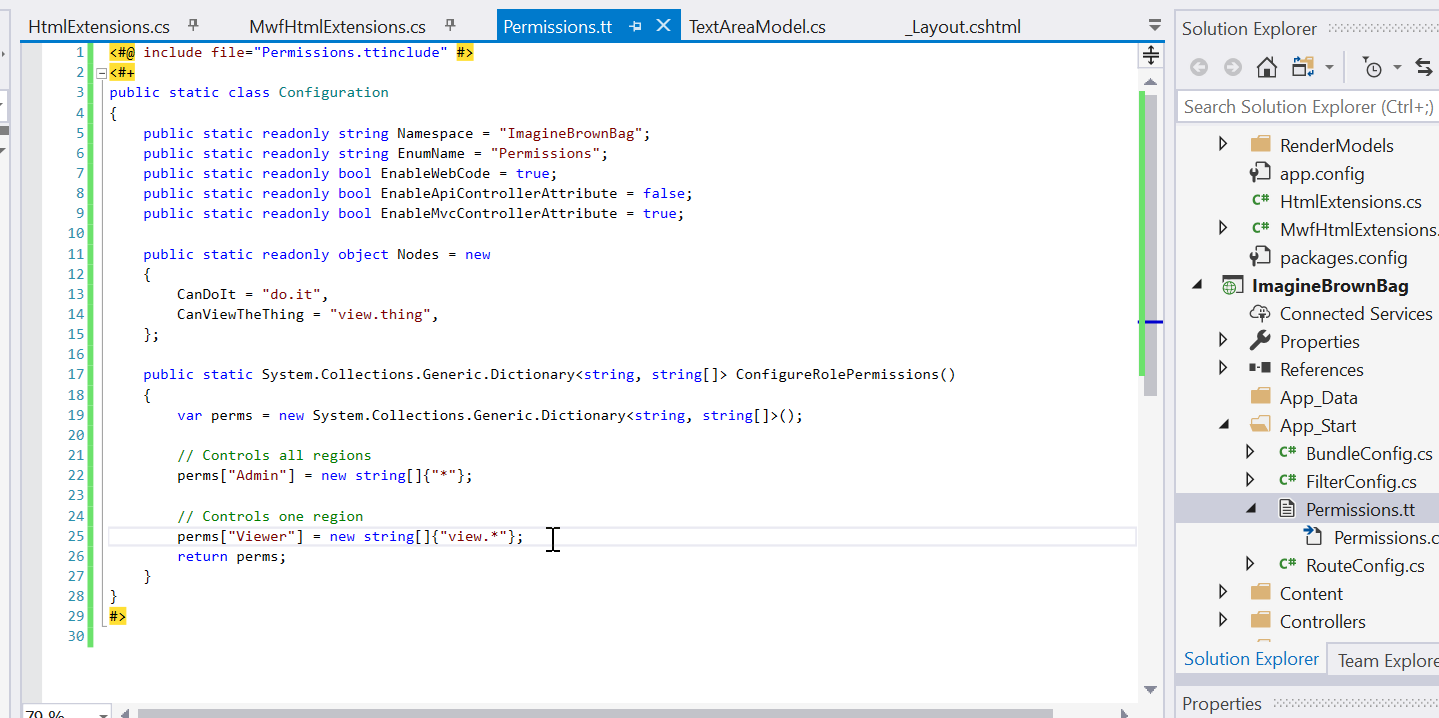
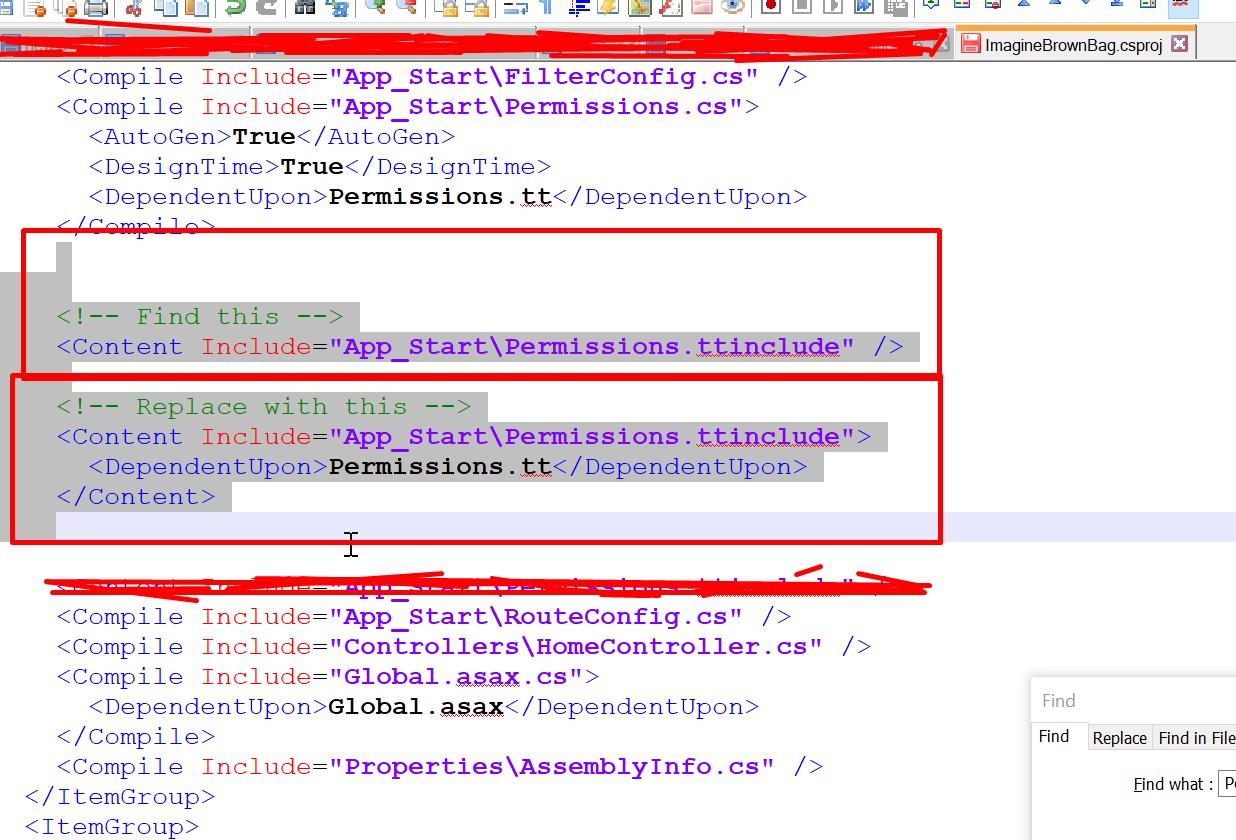
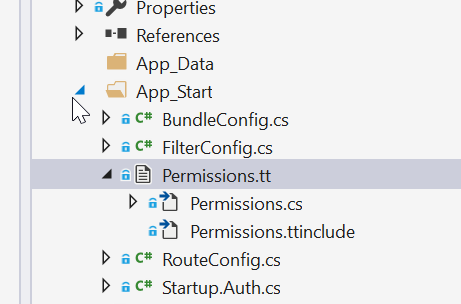
**Framework**: [ASP.NET 4.6.2](https://www.microsoft.com/en-us/download/details.aspx?id=53345). You can probably use other versions, but the example created will be for 4.6.2.

**The Code:** The code should be available within the folder you’re now viewing.

**Instructions:** Open the project in Visual Studio 2017. Make sure VS2017 is run in **Administrator** mode.

Adding T4-Permissions:

In the example project, T4 permissions will have already been added for you. But to see how it can be added to a new project, follow this example.

1. Add a new item, preferably to the App\_Start folder of your web project. (Any folder will work though)  
    
2. The exact name of your tt file does not matter, but the recommended name is “Permissions.tt”.  
   
3. Copy and paste the content from an existing permissions.tt file into your new Permissions.tt file.  
   
4. Copy over the Permissions.ttinclude file as well. Same thing, just copy over the content for Permissions.ttinclude instead.
5. You will notice the Permissions.ttinclude file will not be under Permissions.tt. This looks lame, so we will go into our csproj file to fix it. Open up your <projectname>.csproj file and edit it as shown:  
   
6. That’s it. You have installed the permissions.tt system. Should look like this:  
   

Configuring the permissions:

Configuring is pretty simple. In this example, “CanDoIt” will become the permission’s variable name, and “do.it” will become the permission’s variable value. There are three main things to be configured.

1. Configure **Nodes** to name your permission nodes and define how the permissions can be grouped together.
2. Configure the ConfigureRolePermissions() method to define which groups get access to which permissions. To make things easier, we can take advantage of wild cards. If you have the permissions:   
   CanDoKungFoo = “do.fight.kungfoo”,   
   CanFightKarate = “do.fight.karate”,   
   CanCodeJava = “do.code.java”,   
   CanCodeCSharp = “do.code.c-sharp”,   
   Then you can give group “Foo” the permission node, “do.code.\*” to assign both CanCodeJava and CanCodeCSharp to that group. (During compile time the wild card will be expanded for o(1) look-up at run-time)
3. Assign a custom function to the Permissions.GetRolesCsvForCurrentUser variable at run time if you want roles to come from a different source than claims.

Best practices:

* Enums keys should start with **Is**, **Has**, **Can**, or other Boolean-ish names
* Enum values should use periods to separate different sections or hierarchies in the permission node system. For example, **allow.a** and **allow.b** are both separated after the common “**allow**” section.

|  |  |
| --- | --- |
| T4 (Config) | CS (Output) |
|  |  |

Using Permissions:

[HasPermission] attributes can be applied to controllers and to action routes. They work similar to how you would use an [[Authorize](https://docs.microsoft.com/en-us/previous-versions/aspnet/hh834194(v%3Dvs.118))] attribute, except instead of passing a comma separated list of roles, you pass in the permissions you want to allow.

|  |  |
| --- | --- |
| Controller attribute example | |
| Before | After |
|  |  |

|  |
| --- |
| In-code example. (You can call this from anywhere) |
| Before + After |
|  |

|  |
| --- |
| In-view example |
| <h2>Index</h2>  The tempurature is currently X degrees F.  @if(Permissions.HasPermission(Permissions.CanAdjustTempurature))  {  @using (Html.BeginForm("SetTempurature", "Tempurature", FormMethod.Post))  {  @Html.AntiForgeryToken()  // TextBox with inputs ...  // ....  // ....  <button type="submit">Submit Changes</button>  }  } |

You can start to see how a lack of a permissions system could become confusing as more and more logic and roles are added to the code. It isn’t sustainable in larger systems. So you need to switch to permissions. But that can be a lot of work. The nice thing about system is that it works well with small systems too. Configuration is pretty minimal and the footprint/ overhead of the core system itself is tiny.