Enum Creator User Manual

Overview

Main Purpose: EnumCreator Pro is designed to make it easier for designers and developers to create their own enums directly from Unity. The tool simplifies enum creation by automating the complex parts while giving you full control over the enum content.

What You Need to Know: There are only two main things you need to understand to use this tool effectively:

1. **Create an Enum Definition** - This is where you define your enum values, names, and properties
2. **Know About the Settings** - Configure how enums are generated (powers of two, default flags, etc.)

Everything else is automatically generated for you. The system handles the Unity integration, asset creation, and synchronization behind the scenes.

Code Side Enum Creation: EnumCreator supports creating enum files directly in the generated enums folder. When you place a properly formatted enum file in the generated enums folder, the system will automatically create a corresponding EnumDefinition asset if one doesn't already exist. This allows engineers to work directly with C# code while maintaining full integration with Unity's enum system.

> ***Important**: The tool requires **one enum per file/script**. You cannot put multiple enums in a single C# file. Each enum must be in its own separate file, which allows the system to properly track and manage each enum definition individually.*

How to Create Standalone Enum Files

There are **two main methods** for creating enum files with EnumCreator:

Method 1: Right-Click in Project Window

1. Right-click on any folder in the Project window
2. Navigate to `Create > Enum Creator > Enum File`
3. If you're not in the Generated Enums folder, the system will ask if you want to use the Generated Enums folder
4. Enter a name for your enum in the dialog
5. The system creates an empty enum file and automatically creates an EnumDefinition when you save
6. Edit the file to add your enum values as needed

Method 2: Create Manually

1. Navigate to your generated enums folder (default: `Assets/GeneratedEnums`)
2. Create a new `.cs` file with your enum name (e.g., `MyEnum.cs`)
3. Write your enum following the supported format (see below)
4. Save the file - the system will automatically detect it and create the corresponding EnumDefinition

Supported Enum File Format

Your enum file should follow this format:

```
namespace YourNamespace
{
    [System.Flags]  // Optional - only include if you want a flags enum
    public enum YourEnumName
    {
        [UnityEngine.Tooltip("Description of this value")]  // Optional tooltip
        Value1 = 1,
        [System.Obsolete("This value is deprecated")]
        OldValue = 2,
        Value3 = 4,
    }
}
```

Format Requirements:

- **Namespace**: Required - use any valid C# namespace
- **Enum Name**: Must match the filename (without .cs extension)
- **Values**: Each value should have an explicit numeric value
- **Tooltips**: Optional - use `[UnityEngine.Tooltip("text")]` above values
- **Obsolete Values**: Optional - use `[System.Obsolete("message")]` for deprecated values
- **Flags**: Optional - use `[System.Flags]` attribute for bitwise enums

What Happens When You Save

1. **File Detection**: The system watches the generated enums folder for changes
2. **Parsing**: Your enum file is parsed to extract:
    - Enum name and namespace
    - All enum values with their numeric values (if any)
    - Tooltips and obsolete attributes
    - Flags attribute
3. **Definition Creation**: If no EnumDefinition exists for this enum:
    - A new EnumDefinition asset is created in `Assets/EnumCreator/Definitions/`
    - The asset is named exactly like your enum (e.g., `MyEnum.asset`)
    - The asset is populated with data from your enum file
    - **Empty enums are supported** - you can start with an empty enum and add values later through the inspector
    - The system respects your EnumCreator settings (powers of two, default flags, etc.)
4. **Synchronization**: Future changes to your enum file will update the EnumDefinition
5. **Empty Template**: When using the right-click menu, an empty enum template is created that you can populate with your own values

Benefits

- **Direct Code Editing**: Edit enums directly in C# files instead of using the inspector
- **Version Control Friendly**: Enum files are easy to track and merge in version control
- **IDE Support**: Full IntelliSense and syntax highlighting for enum values

- **Automatic Sync**: Changes are automatically synchronized with Unity's enum system
- **Tooltip Support**: Add documentation directly in your enum files
- **Obsolete Support**: Mark deprecated values with proper obsolete attributes

Tips

- Use meaningful names for your enum files - they become the enum name
- Consider using powers of 2 for flag enums (1, 2, 4, 8, 16...)
- Add tooltips to document what each enum value represents
- Use obsolete attributes to deprecate values instead of deleting them
- The system respects your numeric values - they won't be changed automatically

Troubleshooting

If your enum file isn't being detected:

1. Ensure the file is in the correct generated enums folder (default: `Assets/GeneratedEnums`)
2. Check that the enum name matches the filename (without .cs extension)
3. Verify the enum format is correct (see format requirements above)
4. Ensure the enum name is a valid C# identifier (starts with letter/underscore, contains only letters/digits/underscores)
5. Check the Console for any error messages
6. Try refreshing the project (Ctrl+R) to trigger file detection

If you get naming conflicts when using the right-click menu:

1. The dialog will warn you if a file with the same name already exists
2. Choose "Overwrite" to replace the existing file, or "Cancel" to choose a different name
3. The system validates enum names to prevent invalid C# identifiers

If the right-click context menu doesn't appear:

1. Make sure you're right-clicking on a folder or the project root
2. The menu item appears under `Create > Enum Creator > Enum File`
3. If you're not in the Generated Enums folder, the system will offer to create the file there instead

If your enum definition asset isn't being created:

1. Check that the enum file is properly formatted and compiles without errors
2. Ensure the file is saved and Unity has refreshed
3. Look for any error messages in the Console
4. Empty enums are supported - you can start with an empty enum and add values later

Tools Menu

The following Tools menu items are available for managing Enum Creator:

Tools > Enum Creator > Settings

- Opens the Enum Creator Settings window
- Configure default namespace, generated enums path, and other preferences
- Set whether new enums should use flags by default
- Configure powers of two numbering for unflagged enums

Tools > Enum Creator > Create Settings Asset

- Creates a new EnumCreatorSettings asset if one doesn't exist
- Automatically opens the Settings window after creation
- Useful for first-time setup or resetting settings

Tools > Enum Creator > About

- Shows information about the Enum Creator tool
- Displays version information and features

Asset Store Distribution

This feature is designed for professional Unity development workflows and is distributed through the Unity Asset Store. The standalone enum file system provides:

- **Professional Development Workflow**: Engineers can work with familiar C# code editing
- **Team Collaboration**: Enum files are easily shared and version controlled
- **IDE Integration**: Full IntelliSense and debugging support in your preferred IDE
- **Unity Integration**: Seamless integration with Unity's inspector and serialization system

Example Files

Empty Enum (Starting Point)

```
namespace Game.Enums
{
    public enum Weapons
    {
    }
}
```

This is what you get when using the right-click menu. You can add values later through the inspector or by editing the file directly.

Simple Enum

```
namespace Game.Enums
{
    public enum PlayerState
    {
        Idle = 0,
        Walking = 1,
        Running = 2,
        Jumping = 3,
    }
}
```

Flags Enum with Tooltips

```
namespace Game.Enums
{
    [System.Flags]
    public enum GameFlags
    {
        [UnityEngine.Tooltip("No special flags set")]
        None = 0,
        [UnityEngine.Tooltip("Player has completed tutorial")]
        TutorialCompleted = 1,
        [UnityEngine.Tooltip("Player has unlocked hard mode")]
        HardModeUnlocked = 2,
        [UnityEngine.Tooltip("Player is a premium member")]
        PremiumMember = 4,
        [System.Obsolete("This flag is no longer used")]
        OldFlag = 8,
    }
}
```

Complex Enum with Mixed Attributes

```
namespace Game.Enums
{
    [System.Flags]
    public enum WeaponType
    {
        [UnityEngine.Tooltip("No weapon equipped")]
        None = 0,
        [UnityEngine.Tooltip("Melee weapons for close combat")]
        Melee = 1,
        [UnityEngine.Tooltip("Ranged weapons for distance combat")]
        Ranged = 2,
        [UnityEngine.Tooltip("Explosive weapons with area damage")]
        Explosive = 4,
        [UnityEngine.Tooltip("Magical weapons with special effects")]
        Magical = 8,
        [System.Obsolete("Energy weapons have been removed from the game")]
        Energy = 16,
        [UnityEngine.Tooltip("Ancient weapons with special properties")]
        Ancient = 32,
    }
}
```