Standalone Enum Files Guide

Overview

EnumCreator Pro 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.

How to Create Standalone Enum Files

Method 1: Use the Tools Menu

1. Go to `Tools > Enum Creator > Create New Enum File`
2. Enter a name for your enum in the centered dialog that appears
3. The system will create a template enum file with your chosen name in the Generated Enums folder
4. Edit the file to add your enum values
5. Save the file - the system will automatically detect it and create an EnumDefinition

Method 2: 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 the enum file and automatically creates an EnumDefinition when you save

Method 3: 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

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
    - 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
    - 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. **Template Generation**: When using the menu items, templates are generated based on your current EnumCreator settings

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. Try using `Tools > Enum Creator > Utilities > Force Sync All Enum Files`
6. Check the Console for any error messages

If you get naming conflicts when using the menu items:

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

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

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,
    }
}
```