UGESystem Custom Archetype Guide

🛑 **[CRITICAL WARNING] Data Loss on Package Update**

This guide involves **modifying core scripts** (e.g., `GameEventArchetype.cs`, `UGEGameEventController.cs`) inside `Assets/UGESystem/`.

Updating the UGESystem package will OVERWRITE these files, causing your custom changes to be LOST (DELETED).

You MUST remember to **re-apply these changes** manually after every package update.

This guide explains how to add new **Game Event Archetypes** to the UGESystem.

📢 **IMPORTANT NOTICE: Web Story Maker**

- Custom Archetypes are **NOT supported in the Web Story Maker.**
- They can ONLY be created and configured within the **Unity Editor.**

Goal: What are we building?

As an example, we will create an **"Exploration"** mode. In this mode, we will restrict the environment to show only simple monologue or observation text, without complex dialogue choices.

Step 1: Register Archetype Name (Enums)

We need to register the new mode name in two separate files so the system recognizes it.

1. **Open File 1**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Enums/GameEventArchetype.cs`
2. **Add Code**:

```
public enum GameEventArchetype
{
    Generic,
    Dialogue,
    CinematicText,

    // [Add] New Archetype Name
    Exploration
}
```

3. **Open File 2**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Enums/GameEventType.cs`
4. **Add Code**: (Ensure the spelling matches exactly)

```
public enum GameEventType
{
    Dialogue,
    CinematicText,

    // [Add]
    Exploration
}
```

Step 2: Define Execution Strategy (Controller)

This is the most critical step. You define **how commands behave** when running in this new `Exploration` mode.

1. **Open File**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Managers/UGEGameEventController.cs`
2. **Edit Code**: Add a new entry to the `_commandHandlers` dictionary inside the `InitializeCommandHandlers` function.

```
private void InitializeCommandHandlers()
{
    // ... existing handler creation ...

    _commandHandlers = new Dictionary<GameEventType, Dictionary<Type, ICommandHandler>>
    {
        // ... existing Dialogue and CinematicText settings ...

        // [Add] Define Exploration Mode
        {
            GameEventType.Exploration, new Dictionary<Type, ICommandHandler>
            {
                // Register ONLY the commands allowed in this mode.

                { typeof(DialogueCommand), dialogueHandler },
                { typeof(CharacterCommand), characterHandler },
                { typeof(PlaySoundCommand), playSoundHandler },
                { typeof(EndCommand), endHandler }
                // Add more as needed.
            }
        }
    };
}
```

Step 3: Verify and Use

1. Return to the Unity Editor and wait for compilation.
2. In the Project window, select `Create` > `UGESystem` > `New Game Event`.
3. Click the **Archetype** dropdown in the Inspector window. You should see **Exploration** listed.
4. Select it, and the event will now behave according to the rules (the list of commands) you defined in Step 2.

Advanced: Hide Specific Commands in Editor (Optional)

What if you want to prevent a specific command from even appearing in the **Add Command** menu when this archetype is selected?

1. **Open File**: The command script you want to restrict (e.g., `ChoiceCommand.cs`).
2. **Add Attribute**: Use the `[AvailableIn]` attribute above the class definition.

```
// Example: I want ChoiceCommand to appear ONLY in Dialogue mode, not Exploration.
[AvailableIn(GameEventArchetype.Dialogue)]
public class ChoiceCommand : EventCommand { ... }
```