UGESystem Custom Start Condition Guide

> 🔘 **[CRITICAL WARNING] Data Loss on Package Update**
>
> This guide instructs you to create files within the `Assets/UGESystem/` folder for convenience.
> **Updating the UGESystem package may RESET (DELETE) all changes within this folder.**
>
> For safety, it is strongly recommended to create custom scripts **OUTSIDE the package folder (e.g., `Assets/MyGame/Scripts/`)**,
> or **BACK UP your work manually** before performing any package updates.

This guide explains how to extend UGESystem's storyboard trigger logic by creating custom Start Conditions.

> 📢 **IMPORTANT NOTICE: Web Story Maker**
>
> - Custom Start Conditions are **NOT supported in the Web Story Maker.**
> - They can ONLY be added and configured within the **Unity Editor's Storyboard Editor window.**

Goal: What are we building?

As an example, we will create a **`HealthLowCondition`** that automatically triggers an event when the player's health drops below 30%.

Step 1: Define Data Structure (DTO Class)

Create a Data Transfer Object (DTO) class to handle saving and loading the condition data.

1. **Folder**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Data/Storyboard/Conditions/` (or your custom folder)
2. **Create File**: `HealthLowConditionDto.cs` (C# Script)
3. **Write Code**: Inherit from `BaseEventConditionDto`.

```
using Newtonsoft.Json;

namespace UGESystem
{
    // Data class responsible for file I/O
    public class HealthLowConditionDto : BaseEventConditionDto
    {
        // Stores the health percentage threshold
        [JsonProperty] public float ThresholdPercentage { get; set; }

        public override AbstractEventCondition ToCondition()
        {
            // Converts this DTO back to the runtime Logic class
            return new HealthLowCondition(this);
        }
    }
}
```

Step 2: Implement Logic (Condition Class)

Create the core logic that monitors the game state and satisfies the condition. **You will need to connect this to your own project's health system.**

1. **Folder**: Same as above (or your custom folder)
2. **Create File**: `HealthLowCondition.cs` (C# Script)
3. **Write Code**: Inherit from `AbstractEventCondition`.

```
using UnityEngine;

namespace UGESystem
```

```
{
    [System.Serializable]
    public class HealthLowCondition : AbstractEventCondition
    {
        // Threshold value to set in Inspector (0.0 ~ 1.0)
        [SerializeField] private float _thresholdPercentage;

        // 1. Default Constructor (Called when adding new in Editor)
        public HealthLowCondition() : base("Player health is low")
        {
            _thresholdPercentage = 0.3f; // Default 30%
        }

        // 2. Load Constructor (Called when loading from file)
        public HealthLowCondition(HealthLowConditionDto dto) : base(dto)
        {
            _thresholdPercentage = dto.ThresholdPercentage;
        }

        // 3. Save Method (Called when saving to file)
        public override BaseEventConditionDto ToDto()
        {
            return new HealthLowConditionDto
            {
                Description = Description,
                ThresholdPercentage = _thresholdPercentage
            };
        }

        // 4. [Core] Start Monitoring (Subscribe)
        public override void Subscribe(System.Action onStateChanged)
        {
            base.Subscribe(onStateChanged);

            // [USER IMPLEMENTATION REQUIRED]
            // Connect to your specific player health script's event here.
            // Example: PlayerHealth.Instance.OnHealthChanged += CheckHealth;
        }

        // 5. Stop Monitoring (Unsubscribe) - Prevent Memory Leaks
        public override void Unsubscribe()
        {
            base.Unsubscribe();

            // [USER IMPLEMENTATION REQUIRED]
            // You MUST unsubscribe from the event here.
            // Example: PlayerHealth.Instance.OnHealthChanged -= CheckHealth;
        }

        // 6. [USER IMPLEMENTATION REQUIRED] Logic to check health
        private void CheckHealth(float currentHealth, float maxHealth)
        {
            // If already met, return to prevent duplicate execution
            if (IsMet) return;

            float percentage = currentHealth / maxHealth;

            // Check if current health is below the threshold
            if (percentage <= _thresholdPercentage)
            {
                IsMet = true; // Condition Met!

                // [OPTIONAL] Gameplay Pause/Control
                // You might want to disable input or make the player invincible here
                // immediately before the event starts.

                _onStateChanged?.Invoke(); // Notify system to "Start the Event!"
            }
        }
```

```
    }
}
```

Step 3: Verify

Unlike Commands, Conditions do not need manual registration in a controller. The system automatically detects them using Reflection.

1. Return to the Unity Editor and wait for compilation.
2. Open `Window` > `UGESystem` > `Storyboard Editor`.
3. Select any node in the graph.
4. In the Inspector window, find the `Start Conditions` section and click the `+` button.
5. Check if **HealthLowCondition** appears in the dropdown list and select it.
6. Adjust the `Threshold Percentage` value.

Now, when the player's health drops below the set value, the event for that node will automatically execute.