UGESystem 커스텀 커맨드 추가 가이드

> **◉ [매우 중요] 패키지 업데이트 시 초기화 경고**
>
> 본 가이드는 `Assets/UGESystem/` 폴더 내의 **핵심 스크립트(`CommandType.cs`, `UGEGameEventController.cs` 등)를 직접 수정**하거나 패키지 폴더 내에 **새로운 파일을 생성**하는 방법을 설명합니다.
>
> **UGESystem 패키지를 업데이트할 경우, 이 폴더의 모든 변경 사항은 초기화(삭제/덮어쓰기)됩니다.**
>
> - **새로 만든 파일**: 업데이트 전 안전한 곳으로 **백업**하거나, 처음부터 패키지 폴더 밖(예: `Assets/MyGame/Scripts/`)에 만드세요.
> - **코어 코드 수정**: 업데이트 후 **반드시 다시 적용**해야 합니다.

이 문서는 UGESystem 구매자가 프로젝트 요구사항에 맞춰 새로운 명령어(커맨드)를 추가하는 방법을 설명합니다.

> **📢 웹 스토리 메이커(Web Story Maker) 관련 안내**
>
> - 웹 스토리 메이커는 개발자가 편의를 위해 서버에서 **무상으로 제공하는 부가 서비스**입니다.
> - 사용자는 웹 툴의 소스 코드를 수정하거나 기능을 변경할 수 없습니다.
> - 따라서, 이 가이드를 통해 추가한 **커스텀 커맨드는 웹 툴에 나타나지 않으며 편집할 수 없습니다.**
> - 커스텀 커맨드는 오직 **Unity 에디터의 인스펙터(Inspector) 창에서만** 설정 및 편집이 가능합니다.

목표: 무엇을 만드나요?

예제로 **"게임 실행 중에 콘솔창에 메시지를 띄우는 기능 (`DebugLogCommand`)"**을 만들어 보겠습니다.

1단계: 명령어 타입 등록 (CommandType)

시스템이 새로운 명령어를 인식할 수 있도록 고유한 이름을 등록합니다.

1. **파일 열기**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Enums/CommandType.cs`
2. **방법**: `CommandType` 열거형의 마지막 항목 뒤에 콤마(`,`)를 확인하고, 새 이름을 추가합니다.

```
public enum CommandType
{
    // ... 기존 목록 ...
    TriggerEvent, // 콤마가 없다면 꼭 붙여주세요!

    // [추가] 새 명령어 이름
    DebugLog
}
```

2단계: 데이터 구조 정의 (Command 클래스)

명령어에 필요한 설정값(예: 출력할 메시지 내용)을 담을 스크립트를 만듭니다.

1. **폴더 이동**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Data/Commands/` (또는 여러분의 커스텀 폴더)
2. **파일 생성**: `DebugLogCommand.cs` (C# 스크립트 생성)
3. **코드 작성**: 아래 코드를 복사해서 붙여넣으세요.

```
using UnityEngine;
using System;

namespace UGESystem
{
    [Serializable] // 이 줄이 있어야 저장/로드가 됩니다.
    public class DebugLogCommand : EventCommand
    {
        // 1. Unity 인스펙터에서 편집할 데이터 필드
        [SerializeField] public string LogMessage;

        public DebugLogCommand()
        {
            // 2. 1단계에서 등록한 이름을 연결합니다.
            CommandType = CommandType.DebugLog;
        }

        // 웹 툴은 커스텀 커맨드를 지원하지 않으므로 null을 반환합니다.
        public override IEventCommandDto ToDto()
        {
            return null;
        }
    }
}
```

3단계: 실행 로직 구현 (Handler 클래스)

이 명령어가 실행될 때 **실제로 어떤 행동**을 할지 정의합니다.

1. **폴더 이동**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Managers/Runners/Handlers/` (또는 여러분의 커스텀 폴더)
2. **파일 생성**: `DebugLogCommandHandler.cs` (C# 스크립트 생성)
3. **코드 작성**: 아래 코드를 복사해서 붙여넣으세요.

```
using System.Collections;
using UnityEngine;

namespace UGESystem
{
    // 시스템이 인식하도록 ICommandHandler 인터페이스를 구현합니다.
    public class DebugLogCommandHandler : ICommandHandler
    {
        public IEnumerator Execute(IGameEventCommand command, UGEGameEventController controller)
        {
            // 1. 전달받은 일반 커맨드를 내 전용 타입(DebugLogCommand)으로 변환합니다.
            var cmd = command as DebugLogCommand;

            if (cmd != null)
            {
                // 2. 실제 수행할 동작을 여기에 작성합니다. (로그 출력)
                Debug.Log({{content}}quot;[UGESystem Log] {cmd.LogMessage}");
            }

            // 3. 완료 후 다음 커맨드로 넘어가기 위해 한 프레임 대기하거나 종료합니다.
            yield return null;
        }
    }
}
```

4단계: 시스템 컨트롤러에 등록

마지막으로, 새로 만든 데이터(Command)와 로직(Handler)을 시스템 본체(Controller)에 연결합니다. 기존 로직을 수정하는 것이 아니라, 목록에 추가만 하면 됩니다.

1. **파일 열기**: `Assets/UGESystem/Core/Scripts/UGESystem/GameEvents/Managers/UGEGameEventController.cs`
2. **코드 수정**: `InitializeCommandHandlers` 함수를 찾아 아래 **주석이 달린 두 부분**을 추가하세요.

```
private void InitializeCommandHandlers()
{
    // ... 기존 핸들러 생성 코드들 ...
    var triggerEventHandler = new TriggerEventCommandHandler();

    // [추가 1] 핸들러 인스턴스 생성하기
    var debugLogHandler = new DebugLogCommandHandler();

    _commandHandlers = new Dictionary<GameEventType, Dictionary<Type, ICommandHandler>>
    {
        {
            GameEventType.Dialogue, new Dictionary<Type, ICommandHandler>
            {
                // ... 기존 등록 코드들 ...
                { typeof(TriggerEventCommand), triggerEventHandler },

                // [추가 2] 데이터와 핸들러 연결하기
                // "DebugLogCommand 데이터가 오면 debugLogHandler가 처리해라"
                { typeof(DebugLogCommand), debugLogHandler },
            }
        },
        // 필요한 경우, 시네마틱 텍스트(CinematicText) 모드에도 똑같이 추가할 수 있습니다.
        {
            GameEventType.CinematicText, new Dictionary<Type, ICommandHandler>
            {
                // ... 기존 등록 코드들 ...
                // [선택 사항] 시네마틱 모드에서도 작동하게 하려면 추가
                { typeof(DebugLogCommand), debugLogHandler },
            }
        },
    };
}
```

5단계: 사용 및 확인

1. Unity 에디터로 돌아가 컴파일이 완료될 때까지 기다립니다.
2. 프로젝트 창에서 우클릭 > `Create` > `UGESystem` > `New Game Event`를 선택해 새 이벤트를 만듭니다.
3. 인스펙터 창의 `Commands` 리스트에서 **Add (+)** 버튼을 누르면 목록에 `DebugLog`가 나타납니다.
4. 메시지를 입력하고 게임을 실행하면 콘솔창에 로그가 출력되는 것을 확인할 수 있습니다.