# 유니버설 인스펙터 (Universal Custom Inspector) 사용 설명서

Development Team

2025년 12월 24일

## 소개

이 문서는 유니티 에디터 확장 시스템인 '유니버설 인스펙터'의 기능과 사용법을 다룹니다. 이 시스템은 별도의 커스텀 에디터 스크립트 작성 없이, **속성(Attribute)만 붙여서** 강력한 인스펙터를 구성할 수 있도록 설계되었습니다.

## 차 례

# 1 적용 방법 (두 가지 중 선택)

## 1.1 방법 A: 상속 (권장)

가장 쉽고 확실한 방법입니다. MonoBehaviour 대신 InspectorBase를 상속받으세요.

```
public class MyComponent : InspectorBase
{
    // ...   ...
}
```

## 1.2 방법 B: 속성 추가

이미 다른 클래스를 상속받고 있다면, 클래스 위에 [ExtendedInspector]를 붙이세요.

```
[ExtendedInspector]
public class MyComponent : MonoBehaviour
{
    // ...   ...
}
```

# 2 레이아웃 & 디자인 (Layout)

변수들을 깔끔하게 정리하여 가독성을 높입니다.

## 2.1 탭 (TabGroup)

변수가 많을 때 카테고리별로 탭을 나누어 보여줍니다.

```
[TabGroup(" ")] public int hp;
[TabGroup(" ")] public int mp;

[TabGroup(" ")] public float attackPower;
[TabGroup(" ")] public float defense;
```

## 2.2 박스 그룹 (BoxGroup)

관련된 변수들을 하나의 박스 안에 묶어서 보여줍니다.

```
[BoxGroup("   ")]
public float moveSpeed = 5f;

[BoxGroup("   ")]
public float jumpPower = 10f;
```

## 2.3 제목 & 구분선 (Title)

섹션의 시작을 알리는 굵은 제목과 구분선을 그립니다.

```
[Title("   ")]
public string id;

[Title("   ", true)] // true =
public bool isAdvanced;
```

### 2.4 단위 표시 (Suffix)

변수 옆에 단위(라벨)를 붙여 숫자의 의미를 명확히 합니다.

```
[Suffix("m/s")]
public float maxSpeed;

[Suffix(" ")]
public float duration;
```

## 3 생산성 도구 (Productivity) - 강력 추천

반복 작업을 자동화하여 개발 속도를 획기적으로 높여주는 기능들입니다.

### 3.1 자식 자동 연결 (FindChild)

일일이 드래그할 필요 없이, **이름이 같은 자식 오브젝트**를 찾아서 연결해줍니다.

- `[FindChild]`: 변수 이름과 같은 자식을 찾습니다.
- `[FindChild("Name")]`: 특정 이름의 자식을 찾습니다.

```
[FindChild]
public Text TitleText; //     "TitleText "

[FindChild("Btn_Close")]
public Button closeButton; //     "Btn_Close "
```

### 3.2 에셋 일괄 로드 (AssetList)

**폴더를 선택하면**, 해당 폴더에 있는 모든 파일(스프라이트, 오디오 등)을 배열에 채워줍니다.

```
[AssetList]
public Sprite[] runAnimation; //           !
```

### 3.3 색상 프리셋 (ColorPreset)

자주 쓰는 색상을 버튼으로 만들어, 클릭 한 번으로 적용합니다.

```
// "Red"       , "Blue"
[ColorPreset("Red", 1,0,0,  "Blue", 0,0,1)]
public Color teamColor;
```

### 3.4 폴더 경로 선택 (FolderPath)

탐색기를 열어 폴더를 선택하고, 그 경로(문자열)를 저장합니다.

```
[FolderPath]
public string screenshotSavePath;
```

### 3.5 즉시 반응 (OnValueChanged)

인스펙터에서 값이 바뀌는 즉시, 지정한 함수를 실행합니다. (게임 실행 전 테스트용)

```
[OnValueChanged("UpdateVisual")]
[Range(0, 1)]
public float opacity;

private void UpdateVisual()
{
    GetComponent<CanvasGroup>().alpha = opacity;
}
```

## 4 실수 방지 & 입력 보조 (Validation)

오타(String)나 잘못된 연결을 방지합니다.

### 4.1 드롭다운 선택 시리즈

문자열을 직접 치지 않고 목록에서 선택합니다.

```
[SceneName] public string nextScene;       //     (Build Settings)
[Tag] public string targetTag;             //
[Layer] public int targetLayer;            //    (Int   )
[SortingLayer] public int sortLayerId;     // 2D      (ID)
[InputAxis] public string moveAxis;        // Input Manager
[AnimatorParam("animator")] public string animState; //
```

### 4.2 필수값 체크 (Required)

값이 비어있으면(Null) 빨간색 경고 박스를 띄워 게임 실행 전 실수를 알려줍니다.

```
[Required]
public GameObject coreSystem; //      !

[Required("       .")]
public Transform spawnPoint;
```

### 4.3 에셋/씬 제한 (AssetsOnly / SceneObjectsOnly)

잘못된 유형의 오브젝트 연결을 막습니다.

```
[AssetsOnly]
public GameObject enemyPrefab; //            (    X)

[SceneObjectsOnly]
public Transform patrolPoint; //            ( X)
```

### 4.4 읽기 전용 (ReadOnly)

값은 보여주되, 수정은 못 하게 막습니다. (디버깅용)

```
[ReadOnly]
public int currentComboCount;
```

### 4.5 조건부 표시 (ShowIf)

특정 조건(bool 변수)이 참일 때만 필드를 보여줍니다.

```
public bool hasWeapon;

[ShowIf("hasWeapon")]
public float weaponDamage; // hasWeapon true
```

## 5 시각화 (Visualization)

데이터를 직관적으로 확인합니다.

### 5.1 미디어 뷰어 (Viewer)

이미지, 오디오, 프리팹 등의 미리보기를 표시합니다.

```
[Viewer]
public Sprite icon;

[Viewer(200, 200)] //
public GameObject characterModel;
```

### 5.2 진행바 (ProgressBar)

숫자를 게이지 형태로 보여줍니다.

```
[ProgressBar(100)] //  100
public float health;

[ProgressBar("maxExp")] //  (maxExp)
public float currentExp;
public float maxExp = 1000;
```

### 5.3 상세 툴팁 (ExtendedTooltip)

변수 아래에 접었다 폈다 할 수 있는 설명창을 추가합니다.

```
[ExtendedTooltip("  ", "  .\ n  !")]
public float atk;
```

## 6 액션 버튼 (Button)

함수를 실행하는 버튼을 인스펙터에 추가합니다.

```
[InspectorButton("  ")]
public void TestRun()
{
    Debug.Log("Test Run!");
}
```

# 7 종합 예제 코드

아래 코드를 복사하여 스크립트를 생성하면 모든 기능을 한눈에 테스트할 수 있습니다.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;

public class UniversalInspectorTest : InspectorBase
{
    [TabGroup(" ")] [Title("    ")]
    [Suffix("HP")] public float health = 100;
    [ProgressBar("health")] public float currentHealth = 70;

    [TabGroup(" ")] [BoxGroup(" ")]
    [Suffix("m/s")] public float speed = 5f;
    [TabGroup(" ")] [BoxGroup(" ")]
    [InputAxis] public string horizontalAxis;

    [TabGroup(" ")] [Title("      ")]
    [FindChild("TitleText")] public Text title;
    [AssetList] public Sprite[] icons;
    [FolderPath] public string savePath;

    [TabGroup(" ")] [Title("   ")]
    [SceneName] public string lobbyScene;
    [Tag] public string enemyTag;
    [Layer] public int groundLayer;
    [ColorPreset] public Color uiColor;

    [TabGroup("  ")]
    [ReadOnly] public int frameCount;

    [InspectorButton("   ")]
    public void ResetHealth()
    {
        currentHealth = health;
        Debug.Log("       .");
    }
}
```