Unity ScriptableObject

데이터 중심 설계를 위한 핵심 기술

삼성중공업 Unity 교육 과정

생산라인 장비 관리 시스템 구축

목차

1. ScriptableObject란?

1.1 정의

ScriptableObject는 Unity에서 제공하는 데이터 컨테이너 클래스입니다. MonoBehaviour와 달리 GameObject에 부착하지 않고 독립적인 에셋 파일(.asset)로 프로젝트에 저장됩니다.

1.2 MonoBehaviour vs ScriptableObject

구분	MonoBehaviour	ScriptableObject
부착 대상	GameObject 필수	독립 에셋 파일
라이프사이클	Start, Update 등	OnEnable, OnDisable만
메모리	인스턴스마다 복사	단일 에셋 참조 공유
주 용도	게임 로직, 동작	데이터 저장, 설정

2. 왜 ScriptableObject를 사용하는가?

2.1 메모리 효율성

100개의 적 캐릭터가 같은 스탯 데이터를 사용할 때, MonoBehaviour는 100개의 데이터 복사본을 만들지만, ScriptableObject는 1개의 에셋만 참조합니다.

2.2 데이터와 로직의 분리

데이터(ScriptableObject)와 동작(MonoBehaviour)을 분리하면 유지보수가 쉬워집니다. 디자이너는 코드 수정 없이 Inspector에서 값만 조정할 수 있습니다.

2.3 에디터에서 실시간 수정

Play 모드에서 수정한 값이 유지됩니다. MonoBehaviour는 Play 모드 종료 시 값이 초기화되지만, ScriptableObject 에셋은 변경사항이 저장됩니다.

> 💡 **실무 활용 예시**
>
> 컨베이어 속도를 10에서 20으로 변경할 때, ScriptableObject 에셋 하나만 수정하면 해당 에셋을 참조하는 모든 컨베이어에 즉시 적용됩니다.

3. 기본 구조와 문법

3.1 최소 구조

C#

```
using UnityEngine;

[CreateAssetMenu(fileName = "New Data", menuName = "MyGame/Data")]
public class MyData : ScriptableObject
{
    public string dataName;
    public int value;
}
```

3.2 핵심 요소 분석

요소	설명
`: ScriptableObject`	ScriptableObject 클래스를 상속 (필수)
`[CreateAssetMenu]`	Project 창 우클릭 메뉴에 생성 옵션 추가
`fileName`	생성 시 기본 파일 이름
`menuName`	Create 메뉴에 표시될 경로 (/ 로 하위 메뉴)

3.3 자주 사용하는 Attribute

C#

```
[Header("카테고리 제목")]
[Tooltip("마우스 오버 시 설명")]
public string description;

[Range(0f, 100f)]
public float percentage;

[TextArea(2, 5)]
public string longText;

[SerializeField]
private int privateValue;  // private도 Inspector에 노출

[HideInInspector]
public int hiddenValue;  // public이지만 Inspector에서 숨김
```

4. ScriptableObject 생성하기

4.1 스크립트 파일 생성

1. Project 창에서 Scripts 폴더 선택
2. 우클릭 → Create → C# Script
3. 스크립트 이름 지정 (예: ConveyorSectionData)
4. MonoBehaviour를 ScriptableObject로 변경

4.2 에셋 파일 생성

스크립트 작성 후:

1. Project 창에서 저장할 폴더 선택
2. 우클릭 → Create → [menuName에 지정한 경로]
3. .asset 파일이 생성됨

> ⚠️ 주의: 스크립트 파일(.cs)과 에셋 파일(.asset)은 다릅니다. 스크립트는 '설계도', 에셋은 '실제 데이터'입니다. 하나의 스크립트로 여러 에셋을 만들 수 있습니다.

5. 실전 예제: 컨베이어 데이터

5.1 전체 코드

C#

```
using UnityEngine;

namespace SmartFactory.Data
{
    [CreateAssetMenu(
        fileName = "New Conveyor Section",
        menuName = "SmartFactory/Conveyor Section Data")]
    public class ConveyorSectionData : ScriptableObject
    {
        [Header("기본 정보")]
        [Tooltip("구간 고유 ID")]
        public string sectionId;

        [Tooltip("구간 이름")]
        public string sectionName;

        [Tooltip("컨베이어 타입")]
        public ConveyorType conveyorType = ConveyorType.Roller;

        [TextArea(2, 4)]
        public string description;

        [Header("물리적 스펙")]
        [Range(0.5f, 50f)]
        public float length = 5f;

        [Range(0.3f, 3f)]
        public float width = 0.6f;

        [Header("운영 설정")]
        [Range(1f, 60f)]
        public float speed = 10f;

        public bool isReversed = false;

        [Header("벨트 물리 설정")]
        public Vector3 beltDirection = Vector3.forward;

        [Header("연결 정보")]
        public ConveyorSectionData previousSection;
        public ConveyorSectionData nextSection;

        [Header("상태")]
        public EquipmentStatus status = EquipmentStatus.Idle;

        // 계산 메서드
        public float GetTransportTime()
        {
            if (speed <= 0) return 0;
            return (length / speed) * 60f;
        }
    }
}
```

5.2 코드 분석

► namespace SmartFactory.Data

관련 클래스들을 논리적으로 그룹화합니다. 대규모 프로젝트에서 이름 충돌을 방지하고 코드 구조를 명확하게 합니다.

► [Header("카테고리")]

Inspector에서 필드들을 시각적으로 구분합니다. 관련 있는 필드들을 그룹으로 묶어 가독성을 높입니다.

► [Range(min, max)]

숫자 필드에 슬라이더를 표시하고 값 범위를 제한합니다. 잘못된 값 입력을 방지합니다.

► Enum 타입 (ConveyorType, EquipmentStatus)

Inspector에서 드롭다운으로 표시됩니다. 정해진 옵션 중에서만 선택하도록 강제합니다.

► 자기 참조 (previousSection, nextSection)

같은 타입의 ScriptableObject를 참조할 수 있습니다. 연결 리스트, 트리 구조 등을 구현할 때 유용합니다.

► 계산 메서드

ScriptableObject에도 메서드를 추가할 수 있습니다. 데이터 기반 계산 로직을 캡슐화합니다.

6. 런타임 컴포넌트와 연동

6.1 연동 패턴

C#

```
public class ConveyorBelt : MonoBehaviour
{
    [Header("Data Reference")]
    [SerializeField] private ConveyorSectionData conveyorData;

    [Header("Runtime Status")]
    [SerializeField] private bool isRunning = true;
    [SerializeField] private float currentSpeed = 10f;

    private void Start()
    {
        LoadSettings();
    }

    private void LoadSettings()
    {
        if (conveyorData != null)
        {
            currentSpeed = conveyorData.speed;
            isRunning = conveyorData.status == EquipmentStatus.Running;
        }
    }

    public void RefreshSettings()
    {
        LoadSettings();  // 런타임에 SO 변경 시 호출
    }
}
```

6.2 핵심 포인트

> 💡 **SerializeField 사용**
>
> private 필드에 [SerializeField]를 붙이면 Inspector에서 할당 가능하면서도 외부 접근은 차단됩니다.
> 캡슐화를 유지하면서 Unity 직렬화를 활용하는 방법입니다.

> 💡 **null 체크 필수**
>
> ScriptableObject 참조가 없을 수 있으므로 항상 null 체크를 하고, 기본값 로직을 준비해두세요.

7. 데이터베이스 패턴

여러 ScriptableObject를 하나의 데이터베이스 SO로 관리하면 참조가 편리해집니다.

C#

```
[CreateAssetMenu(menuName = "SmartFactory/Production Line Database")]
public class ProductionLineDatabase : ScriptableObject
{
    [Header("장비 데이터")]
    public List<RobotCellData> robots = new List<RobotCellData>();
    public List<ConveyorSectionData> conveyors = new List<ConveyorSectionData>();
    public List<WorkStationData> stations = new List<WorkStationData>();

    // ID로 검색
    public ConveyorSectionData GetConveyorById(string id)
    {
        return conveyors.Find(c => c.sectionId == id);
    }

    // 상태별 필터
    public List<RobotCellData> GetRobotsByStatus(EquipmentStatus status)
    {
        return robots.FindAll(r => r.status == status);
    }

    // 통계
    public int GetTotalEquipmentCount()
    {
        return robots.Count + conveyors.Count + stations.Count;
    }
}
```

7.1 장점

• 단일 참조점: 하나의 DB만 참조하면 모든 데이터 접근 가능

• 검색 로직 캡슐화: ID 검색, 필터링 등을 DB 내부에 구현

• 통계/집계 용이: 전체 데이터에 대한 계산을 한 곳에서 처리

8. Custom Inspector 기초

Custom Inspector를 통해 ScriptableObject의 편집 UX를 크게 개선할 수 있습니다.

8.1 기본 구조

C#

```
using UnityEngine;
using UnityEditor;

[CustomEditor(typeof(ConveyorSectionData))]
public class ConveyorSectionDataEditor : Editor
{
    private ConveyorSectionData data;
    private SerializedProperty speedProp;

    private void OnEnable()
    {
        data = (ConveyorSectionData)target;
        speedProp = serializedObject.FindProperty("speed");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();

        // 커스텀 UI
        EditorGUILayout.LabelField("속도 설정", EditorStyles.boldLabel);
        EditorGUILayout.PropertyField(speedProp);

        // 버튼
        if (GUILayout.Button("속도 리셋"))
        {
            Undo.RecordObject(data, "Reset Speed");
            data.speed = 10f;
            EditorUtility.SetDirty(data);
        }

        serializedObject.ApplyModifiedProperties();
    }
}
```

8.2 핵심 요소

요소	설명
`[CustomEditor(typeof(...))]`	어떤 타입에 적용할지 지정
`serializedObject`	Undo/Redo 지원하는 직렬화된 객체
`SerializedProperty`	개별 필드 접근용 (Undo 자동 지원)

`Undo.RecordObject`	직접 값 변경 시 Undo 기록 필수
`EditorUtility.SetDirty`	변경사항 저장 플래그 설정

9. 베스트 프랙티스

9.1 폴더 구조

```
Folder
Assets/
├── _SmartFactory/
│   ├── Scripts/
│   │   ├── Runtime/
│   │   │   ├── Data/           ← ScriptableObject 스크립트
│   │   │   │   ├── ConveyorSectionData.cs
│   │   │   │   ├── RobotCellData.cs
│   │   │   │   └── Enums.cs
│   │   │   └── Controllers/    ← MonoBehaviour 스크립트
│   │   │       └── ConveyorBelt.cs
│   │   └── Editor/             ← Custom Inspector
│   │       └── ConveyorSectionDataEditor.cs
│   └── Data/                   ← ScriptableObject 에셋
│       ├── Conveyors/
│       │   ├── Conveyor_Main.asset
│       │   └── Conveyor_Sub.asset
│       └── Robots/
│           └── Robot_001.asset
```

9.2 네이밍 컨벤션

유형	패턴	예시
SO 스크립트	[대상]Data	`ConveyorSectionData`
SO 에셋	[타입]_[이름]	`Conveyor_Main.asset`
DB 에셋	[시스템]Database	`ProductionLineDB.asset`

10. 흔한 실수와 해결책

❌ 실수 1: Play 모드에서 SO 수정 후 저장 안 함

Play 모드에서 SO 값을 변경해도 자동 저장되지 않습니다. Ctrl+S로 명시적 저장 필요.

❌ 실수 2: 런타임에서 SO 원본 수정

빌드된 게임에서 SO 값을 수정하면 해당 세션에만 적용됩니다. 영구 저장이 필요하면 별도 저장 시스템(PlayerPrefs, JSON 등)을 사용하세요.

❌ 실수 3: Editor 스크립트를 Runtime 폴더에 생성

Custom Inspector 등 Editor 전용 스크립트는 반드시 'Editor' 폴더에 넣어야 합니다. 그렇지 않으면 빌드 에러가 발생합니다.

❌ 실수 4: null 참조 체크 누락

C#

```
// ❌ 위험한 코드
currentSpeed = conveyorData.speed;  // NullReferenceException!

// ✅ 안전한 코드
if (conveyorData != null)
{
    currentSpeed = conveyorData.speed;
}
else
{
    currentSpeed = 10f;  // 기본값
}
```

❌ 실수 5: Undo 지원 없이 값 변경

C#

```
// ❌ Undo 안됨
data.speed = 20f;

// ✅ Undo 지원
Undo.RecordObject(data, "Change Speed");
data.speed = 20f;
EditorUtility.SetDirty(data);
```

ScriptableObject를 잘 활용하면 데이터 관리가 체계적이고 효율적으로 변합니다. 작은 프로젝트부터 적용해보며 익숙해지세요!