

언리얼 엔진 TObjectPtr 가이드

👉 TObjectPtr이란?

TObjectPtr은 언리얼 엔진 5에서 도입된 UObject 전용 포인터 시스템입니다. 템플릿 기반의 64비트 포인터로, 에디터 빌드에서는 추가 기능을 제공하고 에디터가 아닌 빌드에서는 일반 포인터처럼 동작합니다.

핵심 특징

- 에디터 전용 기능:** 동적 해석(dynamic resolution)과 액세스 추적(access tracking)
- Non-Editor 빌드:** Raw 포인터와 완전히 동일하게 동작 (성능 영향 없음)
- 자동 변환:** 함수 파라미터로 전달하거나 로컬 변수에 저장할 때 자동으로 Raw 포인터로 변환
- 옵셔널:** 선택적으로 사용 가능하지만 권장됨

⚠ 중요: 스마트 포인터가 아닙니다!

TObjectPtr은 스마트 포인터(TSharedPtr, TUniquePtr 등)와는 다른 시스템입니다.

| 참고: 이 비교는 이해를 돋기 위해 추가한 것으로 공식 문서에는 명시되어 있지 않습니다.

	스마트 포인터	TObjectPtr
대상	일반 C++ 클래스	UObject만
메모리 관리	참조 카운팅	가비지 컬렉션
UPROPERTY	✗ 사용 불가	✓ 사용 가능
Non-Editor 빌드 오버헤드	있음	없음 (Raw 포인터로 변환)
에디터 기능	없음	동적 해석, 액세스 추적

VS UE4 vs UE5 비교

UE4 방식 (Raw Pointer)

cpp

```

UCLASS()
class AMyActor : public AActor
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    AActor* OtherActor; // Raw Pointer

    UPROPERTY(EditAnywhere)
    USceneComponent* MyComponent;
};

```

UE5 방식 (TObjectPtr) - 권장!

```

cpp

UCLASS()
class AMyActor : public AActor
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    TObjectPtr<AActor> OtherActor; // TObjectPtr

    UPROPERTY(EditAnywhere)
    TObjectPtr<USceneComponent> MyComponent;
};

```

💡 왜 TObjectPtr을 사용해야 하나?

장점

에디터 빌드에서의 이점

- 동적 해석(dynamic resolution) 추가
- 액세스 추적(access tracking) 추가
- 해석된 객체 주소 캐싱으로 성능 향상 가능

Non-Editor 빌드에서

- Raw 포인터로 변환되어 성능 영향 없음
- 기존 코드와 완전히 동일하게 동작
- Development, Test, Shipping 빌드 모두 포함

개발 경험 개선

- 에디터 빌드에서 개발할 때 경험 향상 (원문: "may improve your experience when developing in editor builds")
- 많은 엔진 클래스가 이미 TObjectPtr 사용 중

언제 사용하나?

사용해야 하는 곳 (공식 권장):

- UCLASS의 UPROPERTY UObject 포인터 프로퍼티
- USTRUCT의 UPROPERTY UObject 포인터 프로퍼티
- 컨테이너 클래스 (TArray, TMap 등)

| 참고: 아래 코드 예시는 이해를 돋기 위해 추가한 것입니다.

cpp

```
UCLASS()
class AMyClass : public AActor
{
    GENERATED_BODY()

    //  이런 곳에 사용
    UPROPERTY()
    TObjectPtr<AActor> MyActor;

    UPROPERTY()
    TArray<TObjectPtr<UObject>> MyObjects;

    UPROPERTY()
    TMap<FName, TObjectPtr<AActor>> ActorMap;
};
```

사용하지 않아도 되는 곳:

- 로컬 변수 (.cpp 파일 내부의 지역 변수)
- 함수 파라미터
- 짧은 수명의 임시 변수

cpp

```

void AMyClass::MyFunction()
{
    // ❌ 로컬 변수에는 Raw Pointer 사용
    AActor* LocalActor = GetWorld()->SpawnActor<AActor>();

    // 함수 내부에서는 Raw Pointer 사용이 더 일반적
}

```

자동 변환

TObjectPtr은 대부분의 경우 **자동으로 Raw 포인터로 변환됩니다.**

자동 변환되는 경우

중요: TObjectPtr 변수는 **함수에 전달되거나 로컬 변수에 저장될 때 자동으로 Raw 포인터 타입으로 변환됩니다.**

cpp

```

// 함수 파라미터 전달
void MyFunction(AActor* Actor);

TObjectPtr<AActor> MyActor = GetActor();
MyFunction(MyActor); // ✅ 자동 변환!

// 로컬 변수 저장 - 자동 변환!
AActor* LocalActor = MyActor; // ✅ 자동 변환!

// 비교 연산
if (MyActor == nullptr) // ✅ 자동 변환!
{
    // ...
}

// 역참조
MyActor->DoSomething(); // ✅ 자동 변환!

```

핵심: 대부분의 상황에서 TObjectPtr을 Raw 포인터처럼 사용할 수 있으며, 코드 수정이 거의 필요 없습니다.

프로그래밍 스타일 적응 방법

대부분의 경우 코드 수정이 필요 없지만, 몇 가지 특수한 경우가 있습니다.

1. 컨테이너의 Find 함수

UE4 방식:

cpp

```
TMap<FName, AActor*> ActorMap;  
AActor** FoundActor = ActorMap.Find(Key);
```

UE5 방식:

cpp

```
TMap<FName, TObjectPtr<AAActor>> ActorMap;  
TObjectPtr<AAActor>* FoundActor = ActorMap.Find(Key); // TObjectPtr<T>* 사용!
```

2. Range-based For 루프

UE4 방식:

cpp

```
TArray<AActor*> Actors;  
for (auto* Actor : Actors) // auto* 사용  
{  
    ... Actor->DoSomething();  
}
```

UE5 방식 (권장):

cpp

```
TArray<TObjectPtr<AAActor>> Actors;  
for (auto& Actor : Actors) // auto& 사용 권장!  
{  
    ... Actor->DoSomething();  
}  
  
// 또는  
for (const auto& Actor : Actors) // const auto& 도 좋음  
{  
    ... Actor->DoSomething();  
}
```

이유: TObjectPtr은 해석된 객체 주소를 캐싱할 수 있어서, `auto&` 또는 `const auto&`를 사용하면 향후 액세스 시도에서 시간을 절약할 수 있습니다.

3. 명시적 Raw 포인터 변환

암시적 변환이 불가능한 경우 `ToRawPtr()` 또는 `Get()`을 사용하세요.

삼항 연산자:

cpp

```
TObjectPtr<AActor> ActorA;
TObjectPtr<AActor> ActorB;

// ✗ 암시적 변환이 안 될 수 있음
AAActor* Result = bCondition ? ActorA : ActorB;

// ✅ 명시적 변환
AAActor* Result = bCondition ? ActorA.Get() : ActorB.Get();
// 또는
AAActor* Result = bCondition ? ToRawPtr(ActorA) : ToRawPtr(ActorB);
```

`const_cast` 내부:

cpp

```
TObjectPtr<const AActor> ConstActor;

// ✗ 암시적 변환 안 될 수 있음
AAActor* MutableActor = const_cast<AAActor*>(ConstActor);

// ✅ 명시적 변환
AAActor* MutableActor = const_cast<AAActor*>(ConstActor.Get());
```

4. 델리게이트 함수 파라미터

암시적 변환이 안 되는 드문 경우, pass-through 함수를 만들어야 합니다.

cpp

```

// 원본 함수 시그니처 (Raw Pointer 사용)
static bool MyFunction(UObject* FirstParameter);

// 대부분의 경우 위 함수를 그대로 사용

// 드물게 암시적 변환이 안 되는 경우:
// Pass-through 함수 (헤더 파일)
static bool MyFunction(TObjectPtr<UObject> FirstParameter);

// Pass-through 함수 구현 (소스 파일)
bool UMyClass::MyFunction(TObjectPtr<UObject> FirstParameter)
{
    return SomeOtherFunction(FirstParameter.Get());
}

```

엔진 클래스 멤버 변수의 변화

많은 엔진 클래스의 멤버 변수가 UPROPERTY에서 Raw 포인터에서 TObjectPtr로 변경되었습니다.

예시: AActor::RootComponent

UE4:

```

cpp

class AActor
{
    ... USceneComponent* RootComponent; // Raw Pointer
};

// 직접 접근
MyActor->RootComponent->SetRelativeLocation(NewLocation);

```

UE5:

```

cpp

```

```
class AActor
{
    ... TObjectPtr<USceneComponent> RootComponent; // TObjectPtr
};

// 대부분의 경우 자동 변환되므로 코드 변경 불필요
MyActor->RootComponent->SetRelativeLocation(NewLocation);

// Getter는 여전히 Raw Pointer 반환
USceneComponent* Root = MyActor->GetRootComponent(); // ✓ 변경 불필요
```

중요: 엔진 클래스 멤버 변수와의 직접적인 상호작용이 Raw 포인터 의미론에서 TObjectPtr 의미론으로 변경이 필요한 드문 경우가 있습니다. 하지만 `GetRootComponent()` 같은 Getter 함수는 여전히 Raw 포인터를 반환하므로 이런 호출들은 항상 기존 그대로 유지할 수 있습니다.

实践 예제

참고: 아래 예제들은 이해를 돋기 위해 추가한 것으로 원문에는 없습니다.

예제 1: 기본 사용법

cpp

```

UCLASS()
class AMyGameMode : public AGameModeBase
{
    GENERATED_BODY()

public:
    // UPROPERTY 멤버 변수에 TObjectPtr 사용 (권장)
    UPROPERTY(EditDefaultsOnly, Category = "Game")
    TObjectPtr<UClass> DefaultPawnClass;

    UPROPERTY()
    TObjectPtr<APlayerController> CurrentController;

    void SpawnPlayer()
    {
        // 자동 변환되므로 기존 함수 그대로 사용 가능
        if (DefaultPawnClass)
        {
            // 로컬 변수는 Raw Pointer 사용
            APawn* NewPawn = GetWorld()->SpawnActor<APawn>(DefaultPawnClass);

            if (NewPawn)
            {
                CurrentController->Possess(NewPawn); // 자동 변환
            }
        }
    }
};

```

예제 2: 컨테이너 사용

cpp

```

UCLASS()
class UInventoryComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    // UPROPERTY 컨테이너에 TObjectPtr 사용
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<TObjectPtr<UItemData>> Items;

    UPROPERTY()
    TMap<FName, TObjectPtr<UItemData>> ItemMap;

    void AddItem(UItemData* NewItem)
    {
        // 자동 변환
        Items.Add(NewItem);
        ItemMap.Add(NewItem->GetItemName(), NewItem);
    }

    void ProcessItems()
    {
        // Range-based for with auto& (권장)
        for (auto& Item : Items)
        {
            if (Item)
            {
                Item->Use();
            }
        }
    }

    // Find 사용 시 주의: TObjectPtr<T>* 사용
    TObjectPtr<UItemData>* FoundItem = ItemMap.Find(TEXT("HealthPotion"));
    if (FoundItem && *FoundItem)
    {
        (*FoundItem)->Use();
    }
}

```

예제 3: 함수 파라미터와 로컬 변수

cpp

```

UCLASS()
class AWeaponActor : public AActor
{
    GENERATED_BODY()

public:
    // UPROPERTY 멤버 변수는 TObjectPtr
    UPROPERTY(EditAnywhere)
    TObjectPtr<ACharacter> OwnerCharacter;

    // 함수 파라미터는 Raw Pointer 사용 (관례)
    void SetOwner(ACharacter* NewOwner)
    {
        // 자동 변환
        OwnerCharacter = NewOwner;
    }

    void Fire()
    {
        // 자동 변환
        if (OwnerCharacter)
        {
            ProcessFire(OwnerCharacter); // 자동 변환됨
        }
    }
};

private:
    void ProcessFire(ACharacter* Character)
    {
        // 로컬 함수에서는 Raw Pointer 사용
        Character->TakeDamage(10.0f, FDamageEvent(), nullptr, this);
    }
};

```

🔨 자동 변환 도구: UnrealObjectPtrTool

UE5에는 Raw 포인터를 TObjectPtr로 자동 변환해주는 도구가 포함되어 있습니다.

위치

- 솔루션 계층: [UE5/Programs/UnrealObjectPtrTool/](#)
- 소스 코드: [Engine/Source/Programs/UnrealObjectPtrTool/](#)
- 실행 파일: [Engine/Binaries/Win64/UnrealObjectPtrTool.exe](#)

- (OS에 따라 **Engine/Binaries/[OS]/** 폴더에 위치)

도구의 기능과 제한사항

✓ 자동으로 해주는 것:

- 헤더 파일(.h)의 UPROPERTY 변수를 TObjectPtr로 변환

⚠ 수동으로 해야 하는 것:

- 소스 파일(.cpp)의 필요한 코드 수정
- 컴파일 확인 및 오류 수정

중요: 도구는 헤더 파일의 클래스와 구조체 정의 내의 UPROPERTY 변수만 업데이트합니다. 위에서 설명한 모든 필요한 변경사항을 소스 코드에 적용하지는 않으므로, 여전히 수동으로 조정하고 UnrealObjectPtrTool을 사용하기 전에 프로젝트가 컴파일되는지 확인해야 합니다.

UnrealObjectPtrTool 사용 방법

⚠ 사전 준비 (필수!)

도구를 실행하기 전에 프로젝트가 반드시 컴파일되어야 합니다!

1단계: UHT 설정 수정

Engine\Programs\UnrealHeaderTool\Config\DefaultEngine.ini 파일을 열고 다음 설정을 수정:

ini

NonEngineNativePointerMemberBehavior=AllowAndLog

2단계: 프로젝트 리빌드

모든 코드가 UHT(Unreal Header Tool)에 의해 파싱되도록 프로젝트를 리빌드합니다.

3단계: UHT 로그 파일 위치 확인

컴파일 방식에 따라 로그 파일(**Log.txt**) 또는 (**UnrealHeaderTool.log**)이 다음 위치 중 하나에 생성됩니다:

- **C:\Users\USERNAME\AppData\Local\UnrealBuildTool\Log.txt**
- **C:\Users\USERNAME\AppData\Local\UnrealHeaderTool\Saved\Logs\UnrealHeaderTool.log**
- **Engine\Programs\UnrealBuildTool\Log.txt**

4단계: UnrealObjectPtrTool 컴파일 (소스 빌드만)

Epic Games Launcher에서 설치한 경우: 이미 컴파일되어 있으므로 이 단계는 건너뛰세요.

소스에서 엔진을 실행하는 경우: Visual Studio 솔루션에서 UnrealObjectPtrTool을 컴파일합니다.

5단계: 도구 실행

bash

```
UnrealObjectPtrTool.exe <UHT_LOG_PATH> -SCCCommand="p4 edit -c UPGRADE_CL {filenames}"
```

옵션 파라미터:

- `-PREVIEW` 또는 `-n`: 실제 변경하지 않고 미리보기만 표시

예시:

bash

```
# 미리보기 (실제 변경하지 않음)
UnrealObjectPtrTool.exe C:\Users\MyName\AppData\Local\UnrealHeaderTool\Saved\Logs\UnrealHeaderTool.log

# 실제 변환 (Perforce 사용)
UnrealObjectPtrTool.exe C:\Users\MyName\AppData\Local\UnrealHeaderTool\Saved\Logs\UnrealHeaderTool.log
```

6단계: 수동 수정 및 컴파일 확인 (필수!)

도구 실행 후 반드시:

1. 소스 파일(.cpp)의 필요한 변경사항을 수동으로 수정
2. 프로젝트를 컴파일하여 모든 것이 정상 작동하는지 확인
3. 오류가 있다면 위의 "프로그래밍 스타일 적응 방법" 섹션을 참고하여 수정

💡 모범 사례

✓ 권장사항

1. **새 프로젝트:** 처음부터 TObjectPtr 사용
2. **기존 프로젝트:** 점진적으로 변환 (필수는 아님)
3. **UPROPERTY 멤버 변수:** UCLASS와 USTRUCT의 UObject 포인터 프로퍼티에 TObjectPtr 사용
4. **소스 파일 내부:** 로컬 변수와 파라미터는 Raw 포인터 사용
5. **Range-based for:** `auto&` 또는 `const auto&` 사용
6. **캐싱 활용:** TObjectPtr의 객체 주소 캐싱 이점 활용

✖ 피해야 할 것

1. 불필요한 사용: 로컬 변수나 짧은 수명 변수에 TObjectPtr 사용
2. 과도한 명시적 변환: 대부분 자동 변환되므로 불필요한 `Get()` 호출 지양
3. 성능 걱정: Non-Editor 빌드에서는 Raw 포인터와 동일하므로 걱정 불필요

🎓 언제 무엇을 사용할까?

포인터가 필요한 상황

```
|  
|--- UObject 파생 클래스인가?  
|...|  
|...|--- YES → UObject 포인터 시스템  
|...|...|  
|...|...|--- UCLASS/USTRUCT의 UPROPERTY 멤버 변수?  
|...|...|...|--- YES → TObjectPtr<T> 사용 (UE5 권장)  
|...|...|  
|...|...|--- 로컬 변수나 할수 파라미터?  
|...|...|...|--- YES → T* (Raw Pointer) 사용  
|...|...|  
|...|...|--- 약한 참조 필요?  
|...|...|...|--- YES → TWeakObjectPtr<T>  
|...|...|  
|...|...|--- 에셋 지역 로드?  
|...|...|...|--- YES → TSoftObjectPtr<T>  
|...|  
|...|--- NO → 일반 C++ 클래스  
|...|...|  
|...|...|--- 스마트 포인터 사용 (TSharedPtr, TUniquePtr 등)
```

📊 비교 요약표

참고: 이 표는 이해를 돋기 위해 추가한 것입니다.

항목	Raw Pointer (UE4)	TObjectPtr (UE5)
타입 안전성	보통	향상됨
에디터 기능	없음	동적 해석, 액세스 추적
Non-Editor 빌드 성능	기준	동일 (변환됨)
컴파일 시간	기준	잠재적으로 개선
디버깅	보통	향상됨
코드 마이그레이션	-	대부분 자동 변환
필수 여부	-	선택적 (권장)

❶ 핵심 요약

1. **TObjectPtr ≠ Smart Pointer** - UObject 전용 포인터 래퍼
2. **에디터 전용 기능** - Non-Editor 빌드(Development, Test, Shipping)에서는 Raw 포인터로 변환 (성능 영향 없음)
3. **UPROPERTY에만 사용** - UCLASS/USTRUCT의 멤버 변수에 권장, 로컬 변수나 파라미터는 Raw 포인터 사용
4. **자동 변환** - 함수 전달이나 로컬 변수 저장 시 자동 변환되므로 대부분 코드 수정 불필요
5. **Range-based for는 auto& 사용** - 캐싱 이점 활용
6. **Find 함수 주의** - `TObjectPtr<T>*` 사용
7. **자동 변환 도구 제공** - UnrealObjectPtrTool 활용 가능하지만 수동 수정 필요
8. **도구 사용 전 컴파일 필수** - 프로젝트가 컴파일되는 상태에서 도구 실행
9. **선택적이지만 권장** - 새 프로젝트는 처음부터 사용 권장

❷ 추가 학습 자료

- [UE5 Migration Guide - TObjectPtr](#)
- [TObjectPtr API Reference](#)

팁: 새로운 UE5 프로젝트를 시작한다면 UCLASS와 USTRUCT의 UPROPERTY 멤버 변수에 TObjectPtr을 사용하세요. 기존 UE4 프로젝트는 점진적으로 변환하되, 필수는 아닙니다!