



# 11주차 (2023.12.11)

## Inventory:

### ▼ 펼치기

- 우선 Inventory 관련 클래스들의 형태를 잡아보자

파일 생성:

이름	수정한 날짜	유형	크기
HakInventoryItemInstance.h	2023-12-09 오후 5:54	C Header 원본 파일	OKB
HakInventoryItemInstance.cpp	2023-12-09 오후 5:54	C++ 원본 파일	OKB
HakInventoryItemDefinition.h	2023-12-09 오후 5:54	C Header 원본 파일	OKB
HakInventoryItemDefinition.cpp	2023-12-09 오후 5:54	C++ 원본 파일	OKB
HakInventoryManagerComponent.h	2023-12-09 오후 5:55	C Header 원본 파일	OKB
HakInventoryManagerComponent.cpp	2023-12-09 오후 5:55	C++ 원본 파일	OKB
HakInventoryFragment_EquipableItem.h	2023-12-09 오후 5:55	C Header 원본 파일	OKB
HakInventoryFragment_EquipableItem.cpp	2023-12-09 오후 5:55	C++ 원본 파일	OKB

- HakInventoryItemInstance.h/.cpp
- HakInventoryItemDefinition.h/.cpp
- HakInventoryManagerComponent.h/.cpp
- HakInventoryFragment\_EquipableItem.h/.cpp

HakInventoryItemDefinition:

```

1 #pragma once
2
3 #include "HakInventoryItemDefinition.generated.h"
4
5 /**
6 * Inventory에 대한 Fragment은 확 외단지 않을 수 있다:
7 * - Lyra에서 사용하는 예시를 통해 이해해보자:
8 *   - ULyraInventoryFragment_EquipableItem은 EquipmentItemDefinition을 가지고 있으며, 장착 가능한 아이템을 의미한다
9 *   - ULyraInventoryFramgent_SetStats는 아이템에 대한 정보를 가지고 있다
10 *   - Rifle에 대한 SetStats으로 총알(Ammo)에 대한 장착 최대치와 현재 남은 잔탄 수를 예시로 들 수 있다
11 *   - 등등...
12 */
13 UCLASS(Abstract, EditInlineNew)
14 class UHakInventoryItemFragment : public UObject
15 {
16     GENERATED_BODY()
17 public:
18 };
19
20 UCLASS(Blueprintable)
21 class UHakInventoryItemDefinition : public UObject
22 {
23     GENERATED_BODY()
24 public:
25     UHakInventoryItemDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
26
27     /** Inventory Item 정의(메타) 이름 */
28     UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category=Display)
29     FText DisplayName;
30
31     /** Inventory Item의 Component를 Fragment로 인식하면 된다 */
32     UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category=Display)
33     TArray<TObjectPtr<UHakInventoryItemFragment>> Fragments;
34 };

```

```

1 #include "HakInventoryItemDefinition.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryItemDefinition)
3
4 UHakInventoryItemDefinition::UHakInventoryItemDefinition(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6 {
7 }
8

```

- 현재 우리는 BP의 Property 속성값으로 Instanced를 쓰고 있지 않다:
  - 후일 에셋을 만들어가며, 해당 속성값이 어떤 문제를 일으키는지 확인해보자.
- InventoryItem과 Fragement의 관계에 대한 설명을 주석 설명으로 진행해보자
- HakInventoryItemInstance:

```

#pragma once

#include "UObject/Object.h"
#include "Templates/SubclassOf.h"

#include "HakInventoryItemInstance.generated.h"

/** forward declarations */
class UHakInventoryItemDefinition;

/**
 * 해당 클래스는 Inventory Item의 인스턴스로 볼 수 있다
 */
UCLASS(BlueprintType)
class UHakInventoryItemInstance : public UObject
{
    GENERATED_BODY()

public:
    UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Inventory Item의 인스턴스에는 무엇으로 정의되었는지 메타 클래스인 HakInventoryItemDefinition을 들고 있다 */
    UPROPERTY()
    TSubclassOf<UHakInventoryItemDefinition> ItemDef;
};

```

```

#include "HakInventoryItemInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryItemInstance)

UHakInventoryItemInstance::UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

```

## □ HakInventoryManagerComponent:

```

#pragma once

#include "Components/ActorComponent.h"
#include "HakInventoryManagerComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;

/** Inventory Item 단위 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryEntry
{
    GENERATED_BODY()

    UPROPERTY()
    TObjectPtr<UHakInventoryItemInstance> Instance = nullptr;
};

/** Inventory Item 관리 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryList
{
    GENERATED_BODY()

    FHakInventoryList(UActorComponent* InOwnerComponent = nullptr)
        : OwnerComponent(InOwnerComponent)
    {}

    UPROPERTY()
    TArray<FHakInventoryEntry> Entries;

    UPROPERTY()
    TObjectPtr<UActorComponent> OwnerComponent;
};

/**
 * PlayerController의 Component로서 Inventory를 관리한다
 * - 사실 UActorComponent 상속이 아닌 UControllerComponent를 상속받아도 될거 같은데... 일단 Lyra 기준으로 UActorComponent를 상속받고 있다
 */
UCLASS(BlueprintType)
class UHakInventoryManagerComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    UHakInventoryManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UPROPERTY()
    FHakInventoryList InventoryList;
};

```

```

#include "HakInventoryManagerComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryManagerComponent)

UHakInventoryManagerComponent::UHakInventoryManagerComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
    , InventoryList(this)
{
}

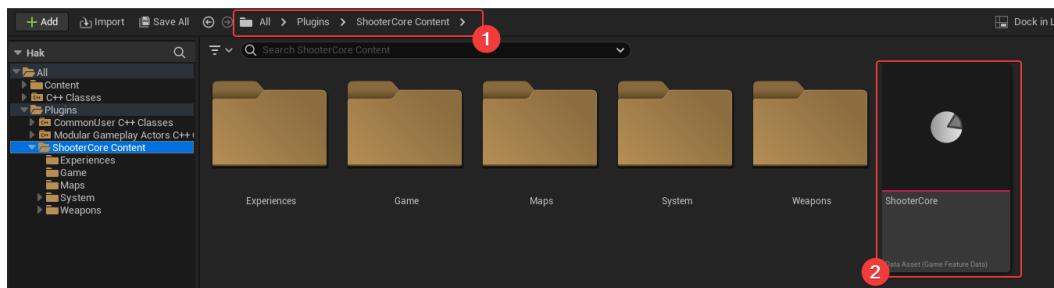
```

HakInventoryFragment\_EquippableItem:

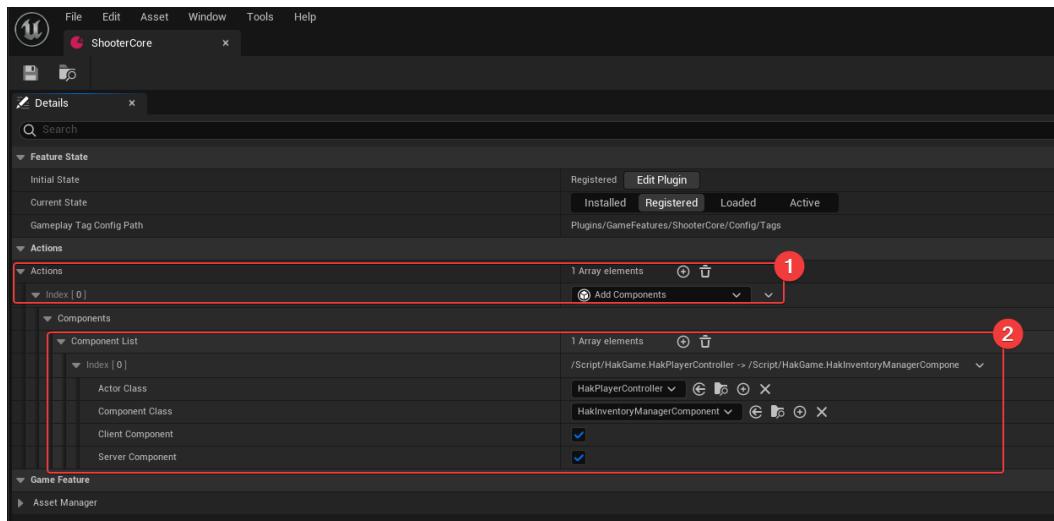
해당 클래스는 Equipment 관련 클래스 정의 이후, 진행하자.

HakInventoryManagerComponent를 Controller에 ShooterCore에 활성화되었을 경우, Inject(이식)될 수 있도록 하자:

아래의 같이 ShooterCore의 GameFeatureData를 열어주자:



AddComponent로 HakInventoryManagerComponent를 Controller에 추가하자:

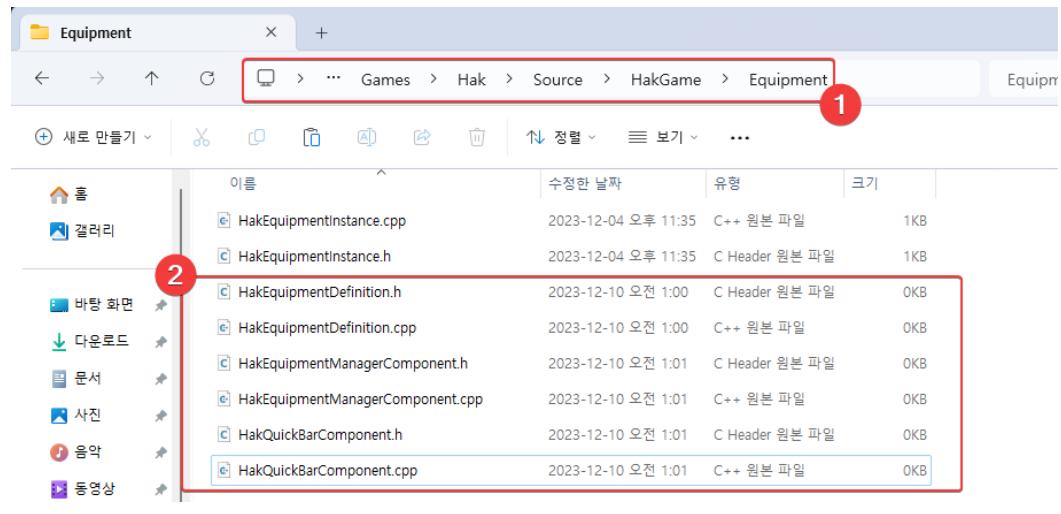


# Equipment:

## ▼ 펼치기

이전 시간에 B\_Pistol을 추가하기 위해, EquipmentInstance 클래스 정의를 위해 이미 폴더를 생성하였다:

여기에 추가적인 클래스 정의를 위해 파일을 추가해주자:



HakEquipmentDefinition:

```

#pragma once

#include "Containers/Array.h"
#include "Math/Transform.h"
#include "Templates/SubclassOf.h"
#include "UObject/NameTypes.h"
#include "UObject/Object.h"
#include "HakEquipmentDefinition.generated.h"

/** Forward declarations */
class UHakEquipmentInstance;

USTRUCT()
struct FHakEquipmentActorToSpawn
{
    GENERATED_BODY()

    /** Spawn할 대상 Actor 클래스 (== Actor를 상속받은 Asset으로 생각해도 될) */
    UPROPERTY(EditAnywhere, Category=Equipment)
    TSubclassOf<AAActor> ActorToSpawn;

    /** 어느 Bone Socket에 붙일지 결정한다 */
    UPROPERTY(EditAnywhere, Category=Equipment)
    FName AttachSocket;

    /** Socket에서 어느정도 Transformation을 더할것인지 결정: (Rotation, Position, Scale) */
    UPROPERTY(EditAnywhere, Category=Equipment)
    FTransform AttachTransform;
};

/***
 * BlueprintType과 Blueprintable은 각각 무엇일까?
 * - Blueprintable은 해당 클래스를 상속받는 모든 클래스는 Blueprint로 정의할 수 있다
 * - BlueprintType은 BP에서 해당 클래스를 변수**로서 사용 가능하다
 */

/***
 * HakEquipmentDefinition은 장착 아이템에 대한 정의 클래스(메타 데이터)이다
 */
UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentDefinition : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 해당 메타 데이터를 사용하면, 어떤 인스턴스를 Spawn할지 결정하는 클래스 */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TSubclassOf<UHakEquipmentInstance> InstanceType;

    /** 해당 장착 아이템을 사용하면, 어떤 Actor가 Spawn이 되는지 정보를 담고 있다 */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TArray<FHakEquipmentActorToSpawn> ActorsToSpawn;
};

```

```

#include "HakEquipmentDefinition.h"
#include "HakEquipmentInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakEquipmentDefinition)

UHakEquipmentDefinition::UHakEquipmentDefinition(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    // 기본값으로, HakEquipmentInstance로 설정
    InstanceType = UHakEquipmentInstance::StaticClass();
}

```

- BlueprintType과 Blueprintable을 이해하기
- 주석에 대한 설명을 진행하기
- 앞서 정의한 HakEquipmentInstance의 멤버 변수 정의:

```

1 #pragma once
2
3 #include "UObject/Object.h"
4 #include "UObject/UObjectGlobals.h"
5 #include "Containers/Array.h"
6 #include "HakEquipmentInstance.generated.h"
7
8 UCLASS(BlueprintType, Blueprintable)
9 class UHakEquipmentInstance : public UObject
10 {
11     GENERATED_BODY()
12 public:
13     UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
14
15     /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후, QuickBarComponent에서 보게 될것이다) */
16     UPROPERTY()
17     TObjectPtr<UObject> Instigator;
18
19     /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance를 */
20     UPROPERTY()
21     TArray< TObjectPtr< AActor >> SpawnerActors;
22 };

```

HakEquipmentManagerComponent:

HakEquipmentManagerComponent 정의:

```

# pragma once
#include "Components/PawnComponent.h"
#include "HakEquipmentManagerComponent.generated.h"

/** forward declarations */

/**
 */
UCLASS(BlueprintType)
class UHakEquipmentManagerComponent : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};

```

```

#include "HakEquipmentManagerComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakEquipmentManagerComponent)

UHakEquipmentManagerComponent::UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

```

HakEquipmentManagerComponent 멤버 변수 정의:

```

#pragma once

#include "Components/PawnComponent.h"
#include "HakEquipmentManagerComponent.generated.h"

/** forward declarations */
class UHakEquipmentDefinition;
class UHakEquipmentInstance;

USTRUCT(BlueprintType)
struct FHakAppliedEquipmentEntry
{
    GENERATED_BODY()

    /** 장착물에 대한 메타 데이터 */
    UPROPERTY()
    TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition;

    /** EquipmentDefinition을 통해 생성된 인스턴스 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> Instance = nullptr;
};

/** 참고로 EquipmentInstance의 인스턴스를 Entry에서 관리하고 있다:
 * - HakEquipmentList는 생성된 객체를 관리한다고 보면 된다
 */
USTRUCT(BlueprintType)
struct FHakEquipmentList
{
    GENERATED_BODY()

    FHakEquipmentList(UActorComponent* InOwnerComponent = nullptr)
        : OwnerComponent(InOwnerComponent)
    {}

    /** 장착물에 대한 관리 리스트 */
    UPROPERTY()
    TArray<FHakAppliedEquipmentEntry> Entries;

    UPROPERTY()
    TObjectPtr<UActorComponent> OwnerComponent;
};

/** Pawn의 Component로서 장착물에 대한 관리를 담당한다
 */
UCLASS(BlueprintType)
class UHakEquipmentManagerComponent : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UPROPERTY()
    FHakEquipmentList EquipmentList;
};

```

```

#include "HakEquipmentManagerComponent.h"
#include "HakEquipmentDefinition.h" ①
#include "HakEquipmentInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakEquipmentManagerComponent)

UHakEquipmentManagerComponent::UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
    , EquipmentList(this) ②
{
}

```

□ HakQuickBarComponent:

□ HakQuickBarComponent 정의:

```

#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};

```

```

#include "HakQuickBarComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakQuickBarComponent)

UHakQuickBarComponent::UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

```

□ HakQuickBarComponent 멤버 변수 정의:

```

#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakEquipmentInstance;

/**
 * HUD의 QuickBar를 생각하면 된다:
 * - 흔히 MMORPG에서는 Shortcut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(Slate)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 Slate Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** HUD QuickBar Slot 갯수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

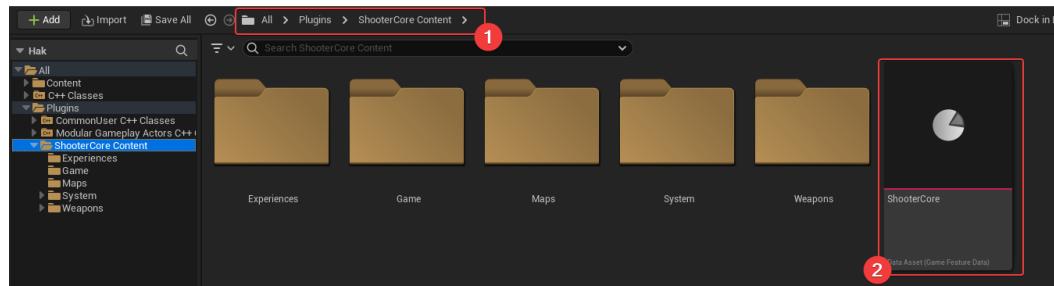
    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveSlotIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};

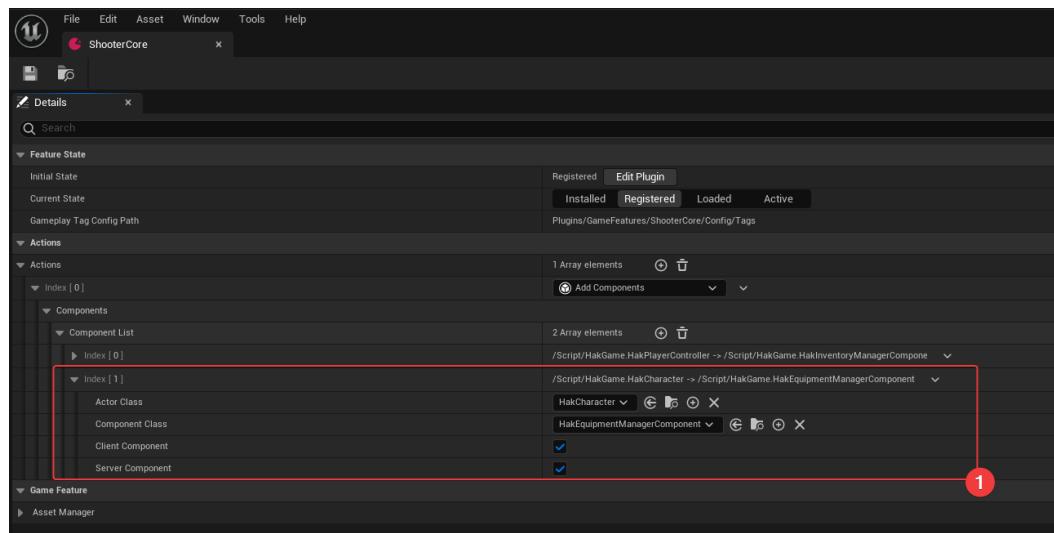
```

□ HakEquipmentManagerComponent를 GameFeatureAction 활성화하자:

□ 앞서 Inventory와 똑같이 ShooterCore의 GameFeatureData를 열어주자:

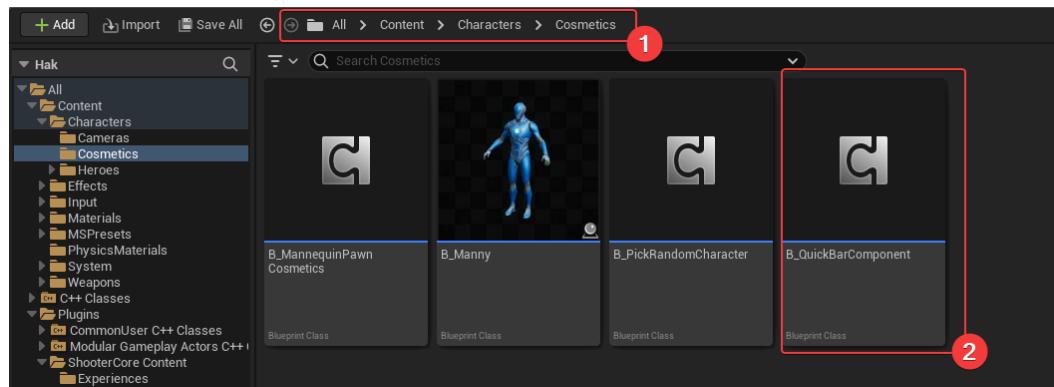
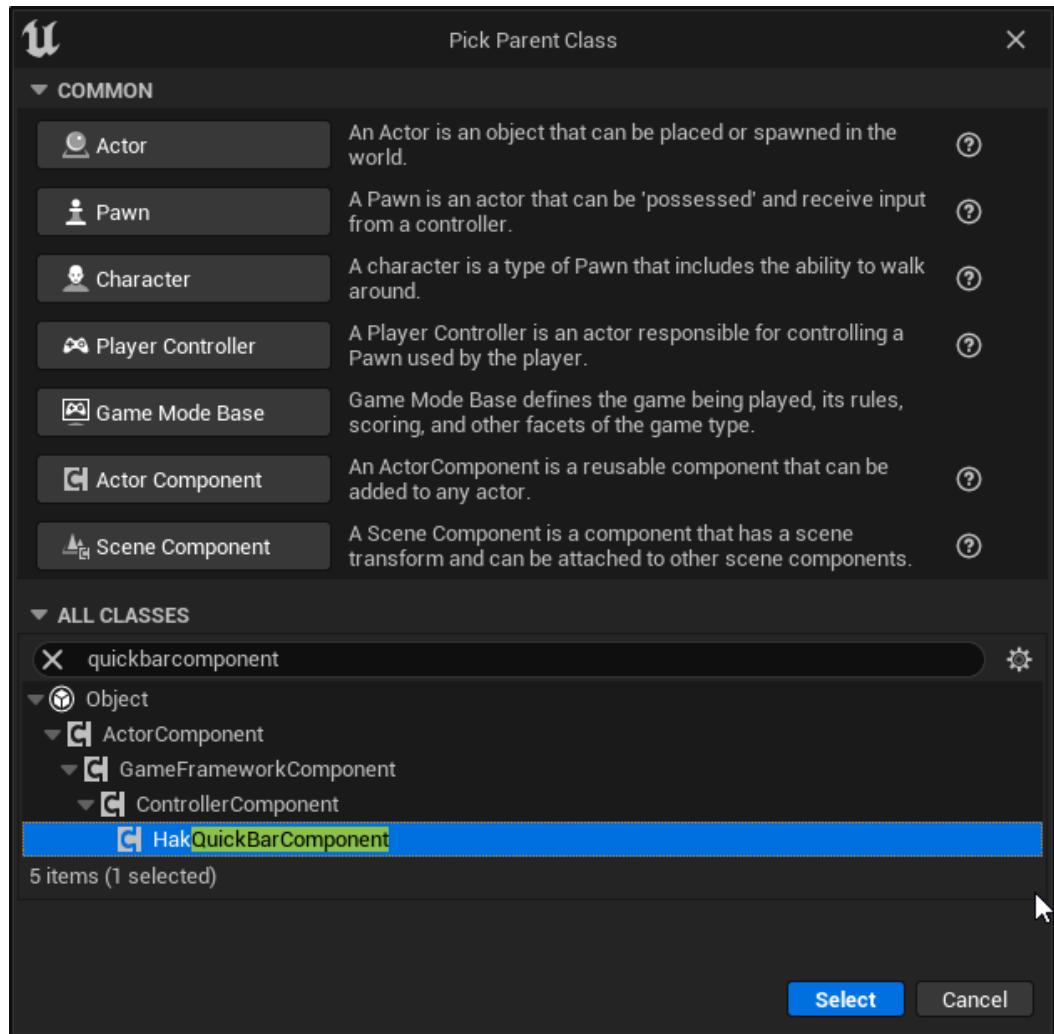


HakCharacter에 EquipmentManagerComponent를 붙여주자:



HakQuickBarComponent를 GameFeatureAction을 활성화하자:

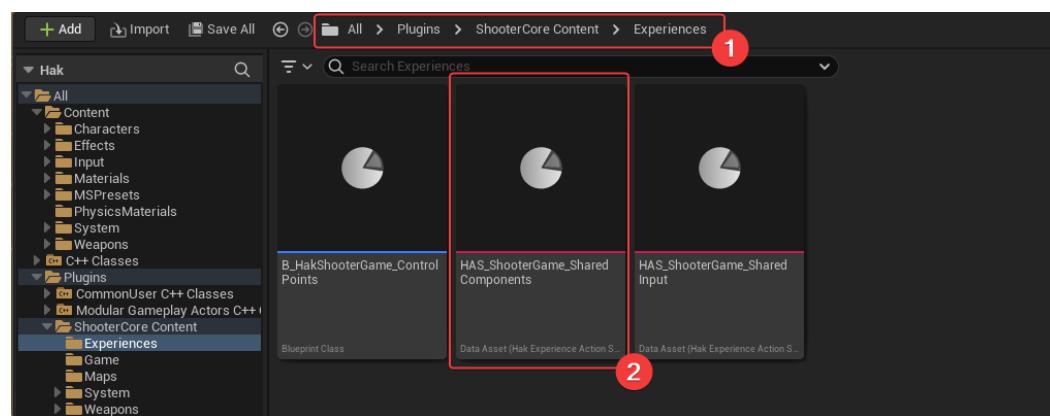
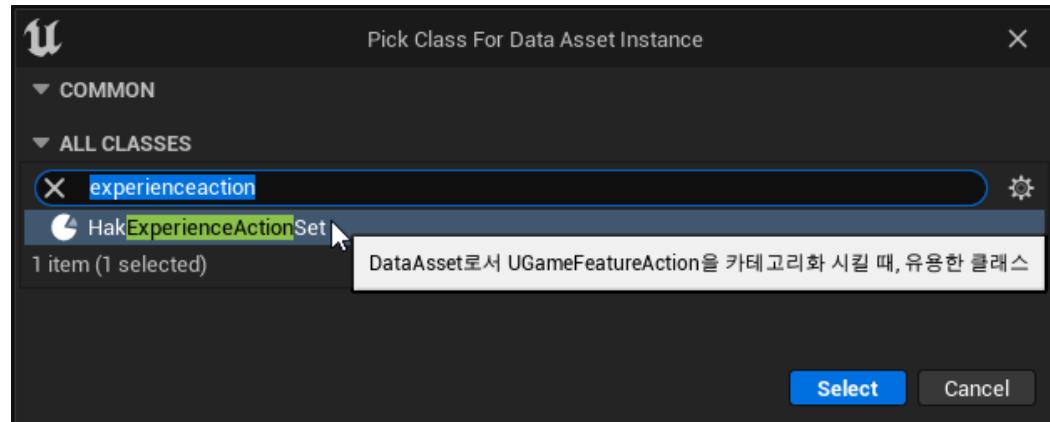
B\_QuickBarComponent BP를 생성하자:



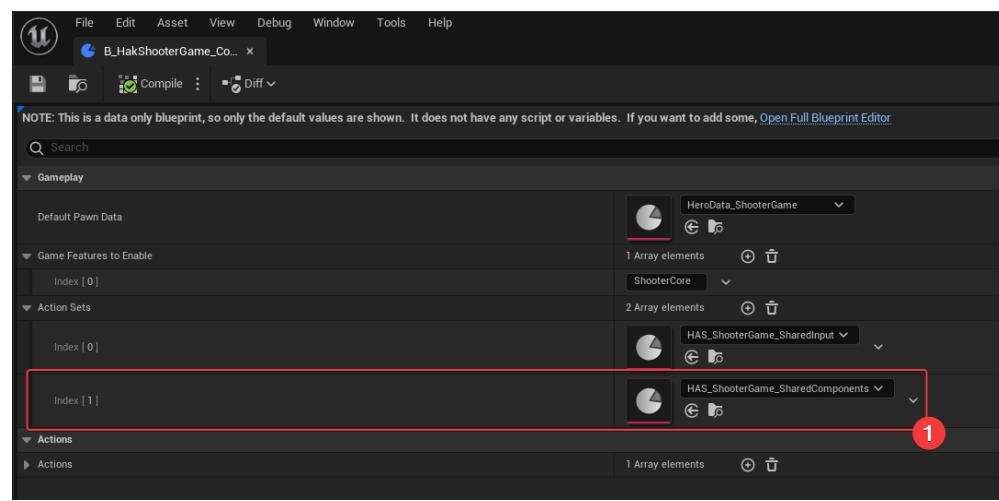
☐ QuickBarComponent는 B\_HakShooterGame\_ControlPoints의 Experience에 한해 활성화할 예정이다

- ShooterCore Plugin인 관점 GameFeature Action이 아니다!
- ShooterCore에 Elimination, ControlPoints 다양한 Experience가 있을 수 있으며, **특정 Experience에 대해 GameFeatureAction을 활성화할 경우는 Experience 내부에 있는 GameFeatureAction 활성화 멤버 변수를 활용하면 된다.**

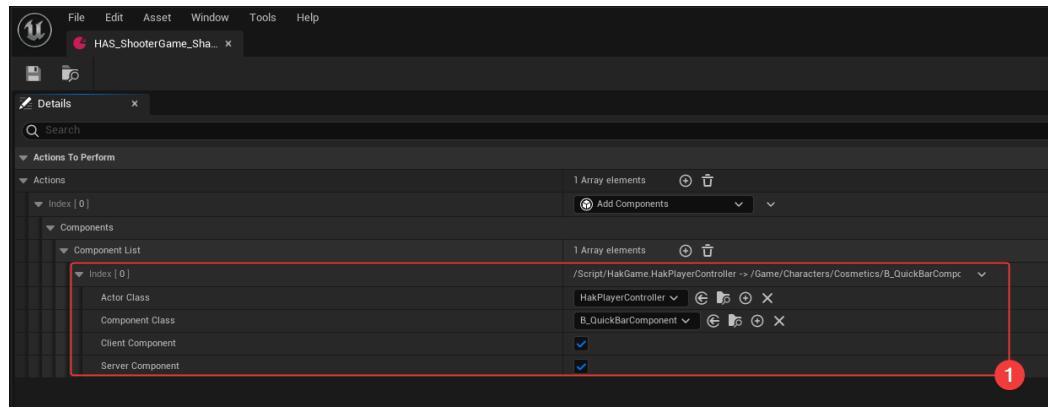
- 우선 Components 추가 용도로, ExperienceActionSet을 생성하자:



- 생성된 에셋을 ExperienceActionSet에 연결해놓자:



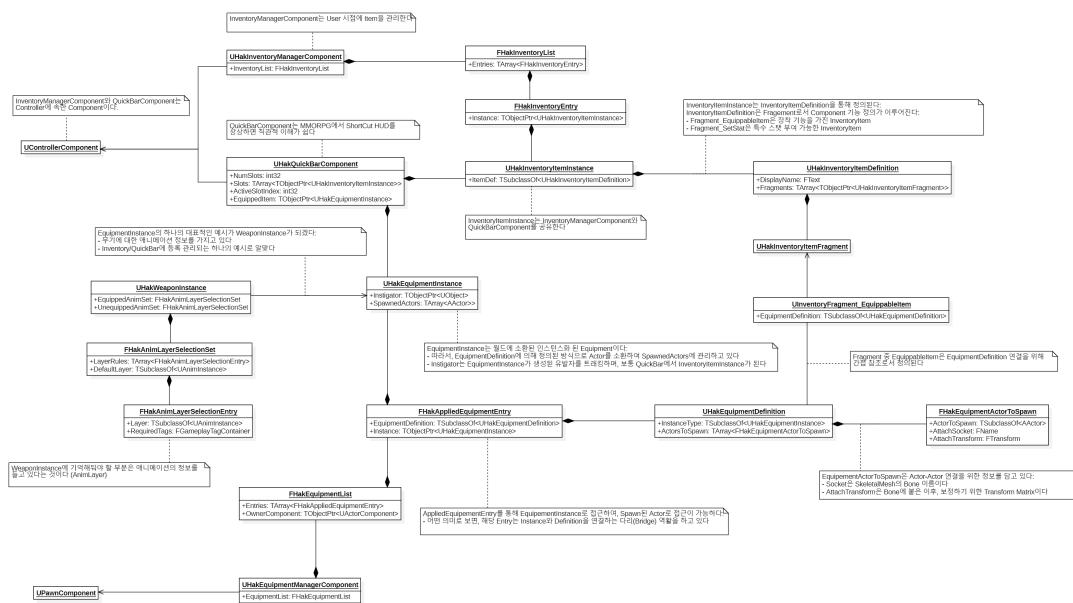
- HAS\_ShooterGame\_SharedComponents에 추가해주자:



## **Inventory/Equipment/QuickBar 클래스 관계:**

#### ▼ 펼치기

#### □ 클래스 다이어그램을 통해 복습해보기



## Initial Inventory

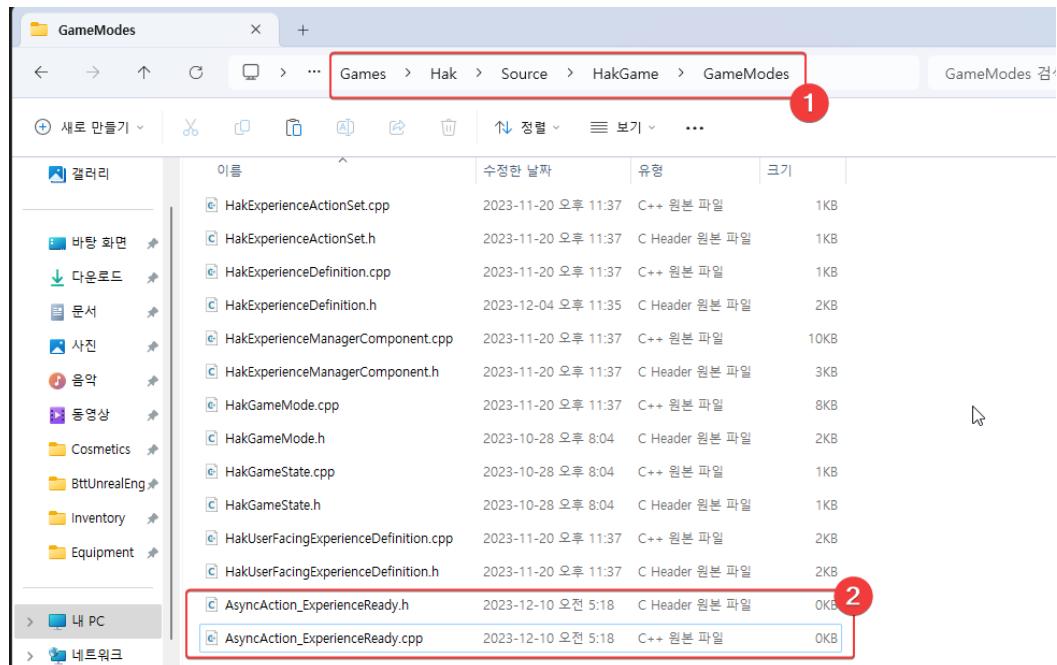
▼ 펼치기

- B\_Hero\_ShooterMannequin이 PlayerController에 의해 Possess가 되었을 경우, 인벤토리 초기화를 통해 Equipment/QuickBar로 기본적인 Pistol 무기 장착까지 이루어진다

**AsyncAction\_ExperienceReady:**

- 해당 기능은 PlayerController가 Pawn을 Possess했을 경우에도 아직 Experience가 준비가 되어 있지 않을 수 있고, 이를 기다리는 BP 이벤트 노드를 정의하는 클래스이다

□ AsyncAction\_ExperienceReady.h/.cpp를 GameModes 폴더에 생성해주자:



□ AsyncAction\_ExperienceReady 클래스 정의:

```
#pragma once

#include "Kismet/BlueprintAsyncActionBase.h"
#include "AsyncAction_ExperienceReady.generated.h"

UCLASS()
class UAAsyncAction_ExperienceReady : public UBlueprintAsyncActionBase
{
    GENERATED_BODY()
public:
    UAAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};

UAAsyncAction_ExperienceReady::UAAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{}
```

□ AsyncAction\_ExperienceReady 멤버 변수 정의:

```

#pragma once

#include "Kismet/BlueprintAsyncActionBase.h"
#include "Delegates/Delegate.h"
#include "AsyncAction_ExperienceReady.generated.h" 1

/** BP를 통한 UFunction으로 바인딩하는 Multicast Delegate: Dynamic! */
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FExperienceReadyAsyncDelegate);

UCLASS()
class UAsyncAction_ExperienceReady : public UBlueprintAsyncActionBase
{
    GENERATED_BODY()
public:
    UAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** BlueprintAssignable은 BP상에서 할당 가능한 변수로 정의한다 */
    UPROPERTY(BlueprintAssignable)
    FExperienceReadyAsyncDelegate OnReady;

    /** WorldPtr을 단순 로직상 캐싱하는 용도 */
    TWeakObjectPtr<UWorld> WorldPtr;
};

```

## □ AsyncAction\_ExperienceReady::Activate:

```

#pragma once

#include "Kismet/BlueprintAsyncActionBase.h"
#include "Delegates/Delegate.h"
#include "AsyncAction_ExperienceReady.generated.h"

/** BP를 통한 UFunction으로 바인딩하는 Multicast Delegate: Dynamic! */
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FExperienceReadyAsyncDelegate); 2

UCLASS()
class UAsyncAction_ExperienceReady : public UBlueprintAsyncActionBase
{
    GENERATED_BODY()
public:
    UAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 
     * UBlueprintAsyncActionBase interface
     */
    virtual void Activate() override; 1

    /** BlueprintAssignable은 BP상에서 할당 가능한 변수로 정의한다 */
    UPROPERTY(BlueprintAssignable)
    FExperienceReadyAsyncDelegate OnReady;

    /** WorldPtr을 단순 로직상 캐싱하는 용도 */
    TWeakObjectPtr<UWorld> WorldPtr;
};

```

```

#include "AsyncAction_ExperienceReady.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(AsyncAction_ExperienceReady)

UAAsyncAction_ExperienceReady::UAAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer) 1

void UAAsyncAction_ExperienceReady::Activate()
{
    if (UWorld* World = WorldPtr.Get())
    {
        // GameState가 이미 World에 준비되어 있으면, Step1을 스킵하고 바로 Step2를 진행한다
        if (AGameStateBase* GameState = World->GetGameState())
        {
            Step2_ListenToExperienceLoading(GameState);
        }
        else
        {
            // 아직 준비되어 있지 않으면, UWorld::GameStateSetEvent에 Step1_HandleGameStateSet을 Delegate로 바인딩시킨다
            // - Step1부터 시작할 것이다
            World->GameStateSetEvent.AddUObject(this, &ThisClass::Step1_HandleGameStateSet);
        }
    }
    else
    {
        // 현재 AsyncAction 대상이었던 World가 더이상 Valid하지 않으므로, 제거 진행
        // - 코드를 내려가보면, GameInstance에서 레퍼런스 대상으로 제거시킨다
        // - UAsyncAction의 캐싱 위치가 GameInstance임을 알 수 있다
        SetReadyToDelete();
    }
}

```

## □ 각 Step을 정의해보자:

```
#pragma once

#include "Kismet/BlueprintAsyncActionBase.h"
#include "Delegates/Delegate.h"
#include "AsyncAction_ExperienceReady.generated.h"

/** BP를 통한 UFunction으로 바인딩하는 Multicast Delegate: Dynamic! */
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FExperienceReadyAsyncDelegate);

/* forward declarations */
class AGameStateBase;
class UHakExperienceDefinition;

UCLASS()
class UAsyncAction_ExperienceReady : public UBlueprintAsyncActionBase
{
    GENERATED_BODY()

public:
    UAsyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * UBlueprintAsyncActionBase interface
     */
    virtual void Activate() override;

    /**
     * Step1 - Step4
     */
    void Step1_HandleGameStateSet(AGameStateBase* GameState);
    void Step2_ListenToExperienceLoading(AGameStateBase* GameState);
    void Step3_HandleExperienceLoaded(const UHakExperienceDefinition* CurrentExperience);
    void Step4_BroadcastReady();

    /** BlueprintAssignable은 BP상에서 할당 가능한 변수로 정의한다 */
    UPROPERTY(BlueprintAssignable)
    FExperienceReadyAsyncDelegate OnReady;

    /** WorldPtr을 단순 로직상 캐싱하는 용도 */
    TWeakObjectPtr<UWorld> WorldPtr;
};
```

## □ Step1\_HandleGameStateSet:

```
void UAsyncAction_ExperienceReady::Step1_HandleGameStateSet(AGameStateBase* GameState)
{
    if (UWorld* World = WorldPtr.Get())
    {
        World->GameStateSetEvent.RemoveAll(this);
    }

    Step2_ListenToExperienceLoading(GameState);
}
```

## □ Step2\_ListenToExperienceLoading:

```
void UAsyncAction_ExperienceReady::Step2_ListenToExperienceLoading(AGameStateBase* GameState)
{
    check(GameState);
    // 여기서 왜 우리가 GameStateSetEvent에 Delegate로 바인딩시켰는지 이유가 나온다:
    // - Experience 르디 상태 정보가 GameState에 붙어있는 ExperienceManagerComponent에서 가져올 수 있기 때문이다!
    UHakExperienceManagerComponent* ExperienceManagerComponent = GameState->FindComponentByClass<UHakExperienceManagerComponent>();
    check(ExperienceManagerComponent);

    // 만약 이미 Experience가 준비되었다면, Step3를 스킵하고 Step4로 간다
    if (ExperienceManagerComponent->IsExperienceLoaded())
    {
        UWorld* World = GameState->GetWorld();
        check(World);

        // 이미 Experience가 준비되었다고 해도, 아직 준비되지 않은 상태가 있을 수도 있으니 (혹은 과정중이라면?) 그래서 그냥 다음 Tick에 실행되도록 Timer를 바인딩시킨다
        World->GetTimerManager().SetTimerForNextTick(FTimerDelegate::CreateUObject(this, &ThisClass::Step4_BroadcastReady));
    }
    else
    {
        // 준비가 안되었다면, OnExperienceLoaded에 바인딩시킨다 (후일 로딩 끝나면 Step3부터 실행이 될 것이다)
        ExperienceManagerComponent->CallOrRegister_OnExperienceLoaded(FOnHakExperienceLoaded::FDelegate::CreateUObject(this, &ThisClass::Step3_HandleExperienceLoaded));
    }
}
```

## □ Step3\_HandleExperienceLoaded

```

void UAyncAction_ExperienceReady::Step3_HandleExperienceLoaded(const UHakExperienceDefinition* CurrentExperience)
{
    // 현재 바로 ExperienceDefinition에 대해 처리할 로직은 없다
    Step4_BroadcastReady();
}

```

□ Step4\_BroadcastReady

```

void UAyncAction_ExperienceReady::Step4_BroadcastReady()
{
    // 바인딩된 BP 혹은 UFUNCTION을 호출해준다
    OnReady.Broadcast();
    SetReadyToDestroy();
}

```

□ UAyncAction\_ExperienceReady를 생성할 BP 함수를 만들어주자:

```

#pragma once

#include "Kismet/BlueprintAsyncActionBase.h"
#include "Delegates/Delegate.h"
#include "AsyncAction_ExperienceReady.generated.h"

/** BP를 통한 UFunction으로 바인딩하는 Multicast Delegate: Dynamic! */
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FExperienceReadyAsyncDelegate);

/** forward declarations */
class AGameStateBase;
class UHakExperienceDefinition;

UCLASS()
class UAyncAction_ExperienceReady : public UBlueprintAsyncActionBase
{
    GENERATED_BODY()
public:
    UAyncAction_ExperienceReady(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** UAyncAction_ExperienceReady를 생성하고 기다리는 BP 호출 */
    UFUNCTION(BlueprintCallable, meta=(WorldContext="WorldContextObject"))
    static UAyncAction_ExperienceReady* WaitForExperienceReady(UObject* WorldContextObject); 1

    /**
     * UBlueprintAsyncActionBase interface
     */
    virtual void Activate() override;

    /**
     * Step1 ~ Step4
     */
    void Step1_HandleGameStateSet(AGameStateBase* GameState);
    void Step2_ListenToExperienceLoading(AGameStateBase* GameState);
    void Step3_HandleExperienceLoaded(const UHakExperienceDefinition* CurrentExperience);
    void Step4_BroadcastReady();

    /** BlueprintAssignable은 BP상에서 할당 가능한 변수로 정의한다 */
    UPROPERTY(BlueprintAssignable)
    FExperienceReadyAsyncDelegate OnReady;

    /** WorldPtr을 단순 로직상 캐싱하는 용도 */
    TWeakObjectPtr<UWorld> WorldPtr;
};

```

```

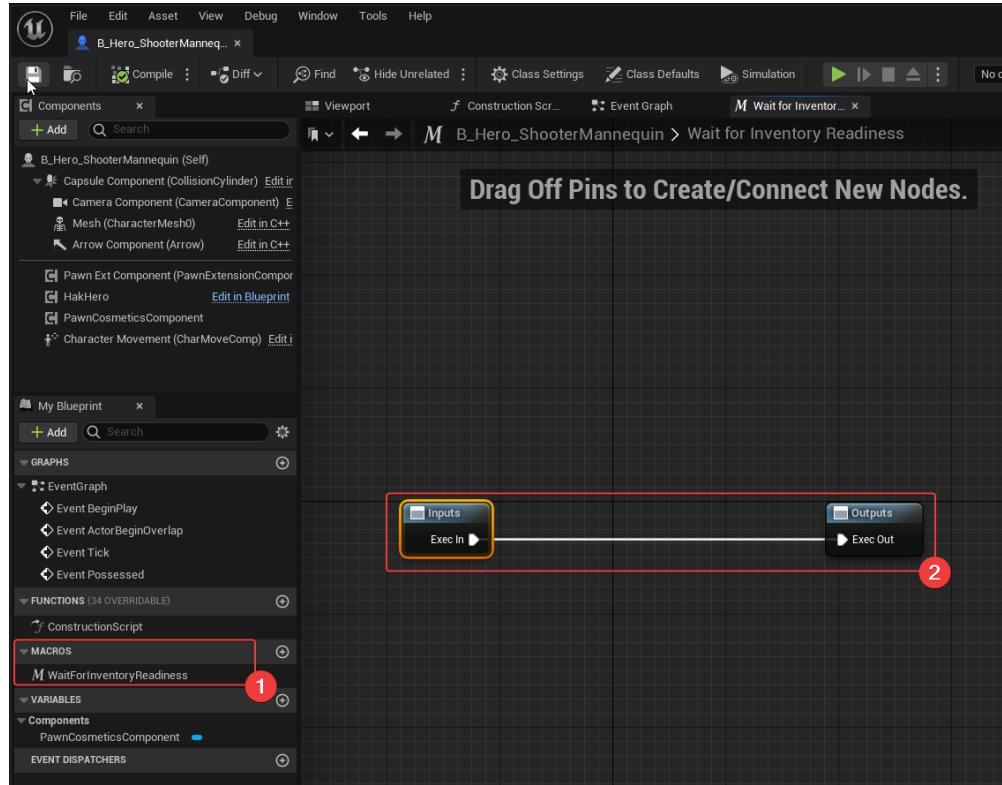
UAyncAction_ExperienceReady* UAyncAction_ExperienceReady::WaitForExperienceReady(UObject* WorldContextObject)
{
    UAyncAction_ExperienceReady* Action = nullptr;
    if (UWorld* World = GEngine->GetWorldFromContextObject(WorldContextObject, EGetWorldErrorMode::LogAndReturnNull))
    {
        Action = NewObject<UAyncAction_ExperienceReady>();
        Action->WorldPtr = World;
        Action->RegisterWithGameInstance(World);
    }
    return Action;
}

```

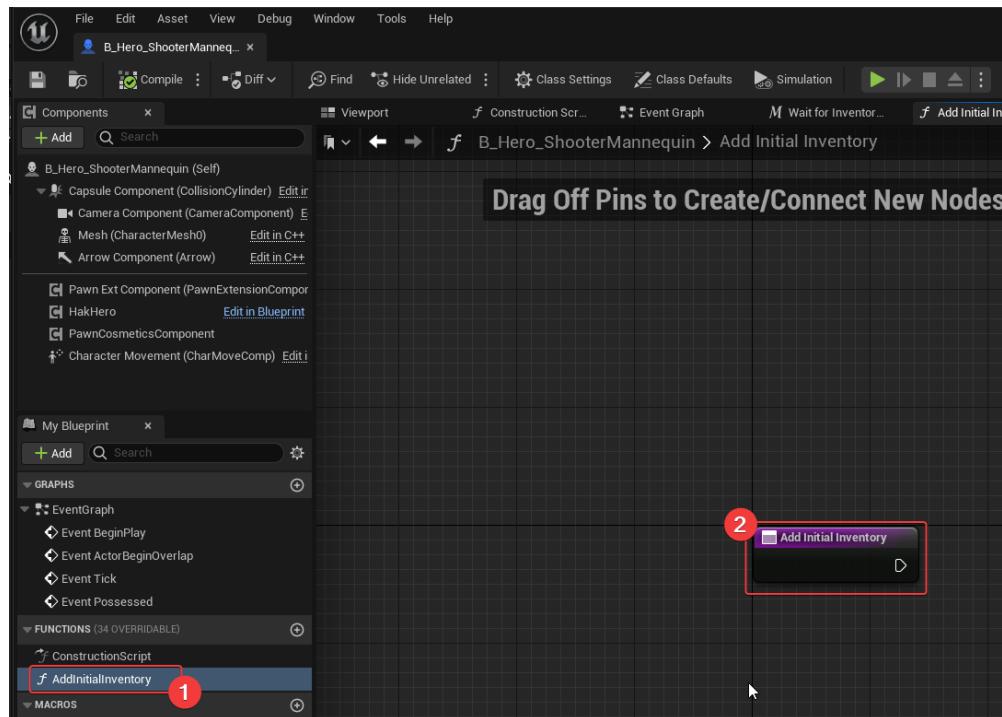
□ B\_Hero\_ShooterMannequin

□ Initial Inventory 로직 생성:

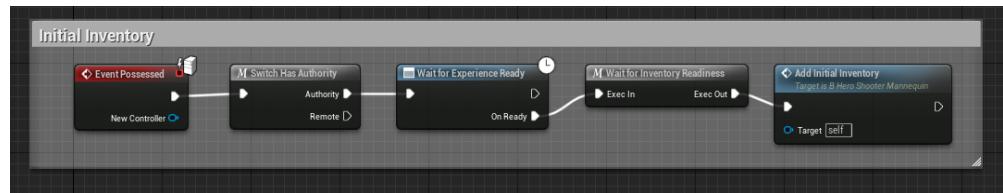
## □ WaitForInventoryReadiness Macro 더미 생성:



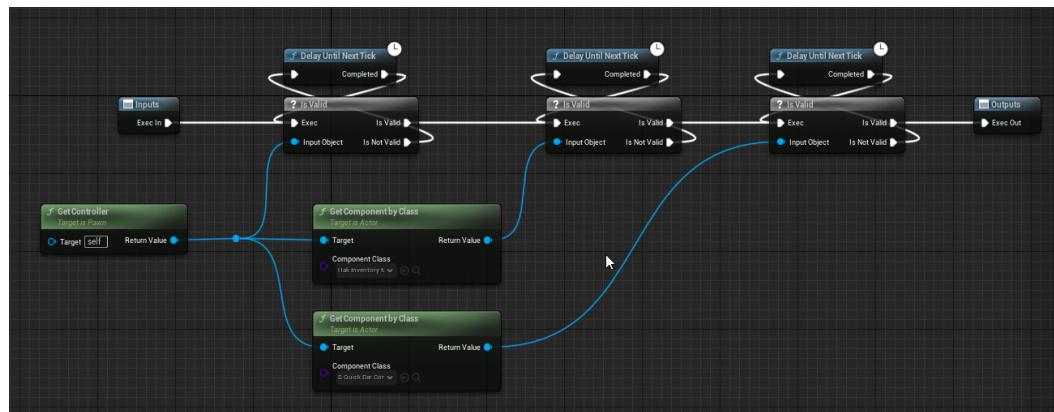
## □ AddInitialInventory 함수 생성:



- 아래와 같이 B\_HeroShooterMannequin이 Controller에 의해 Possess되었을 경우, 이벤트를 시작으로 Inventory 초기화를 실행하는 과정이다:



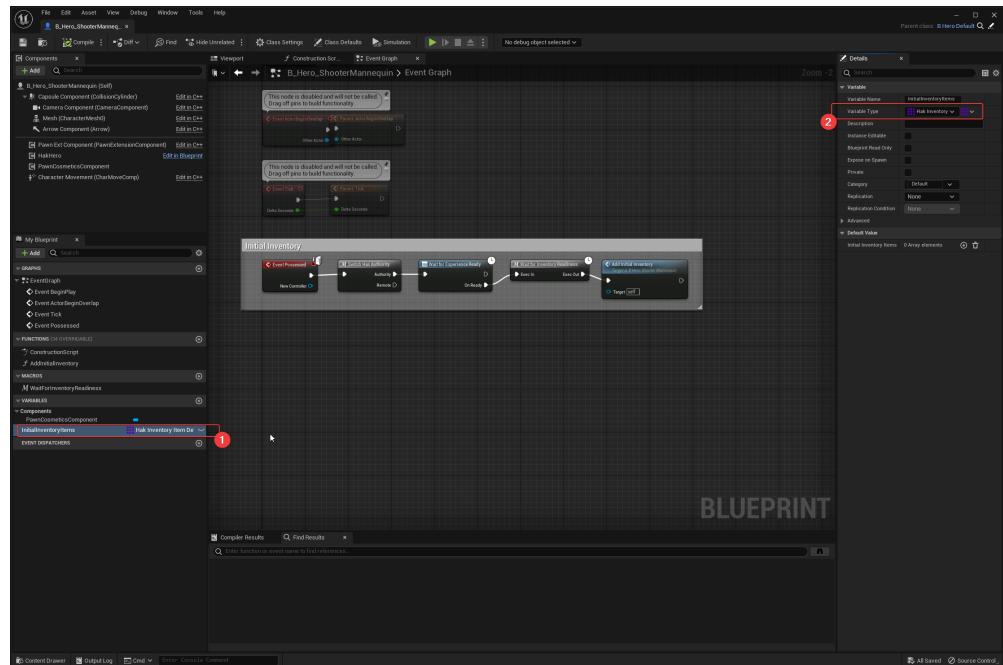
- WaitForInventoryReadiness Macro 구현하자:



- Controller/HakInventoryManagerComponent/B\_QuickBarComponent 가 준비될 때까지 기다리는 매크로이다

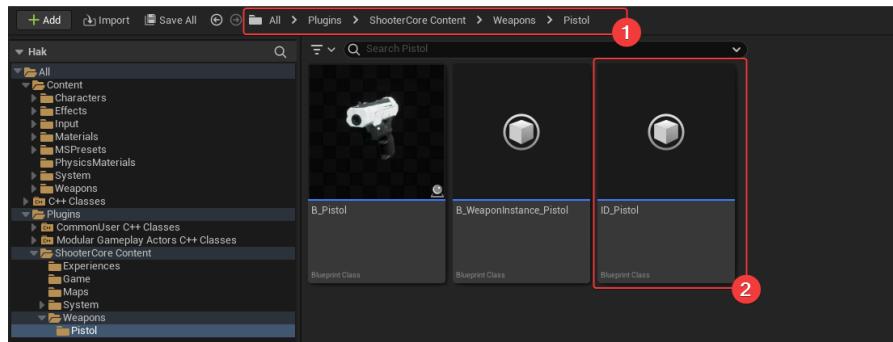
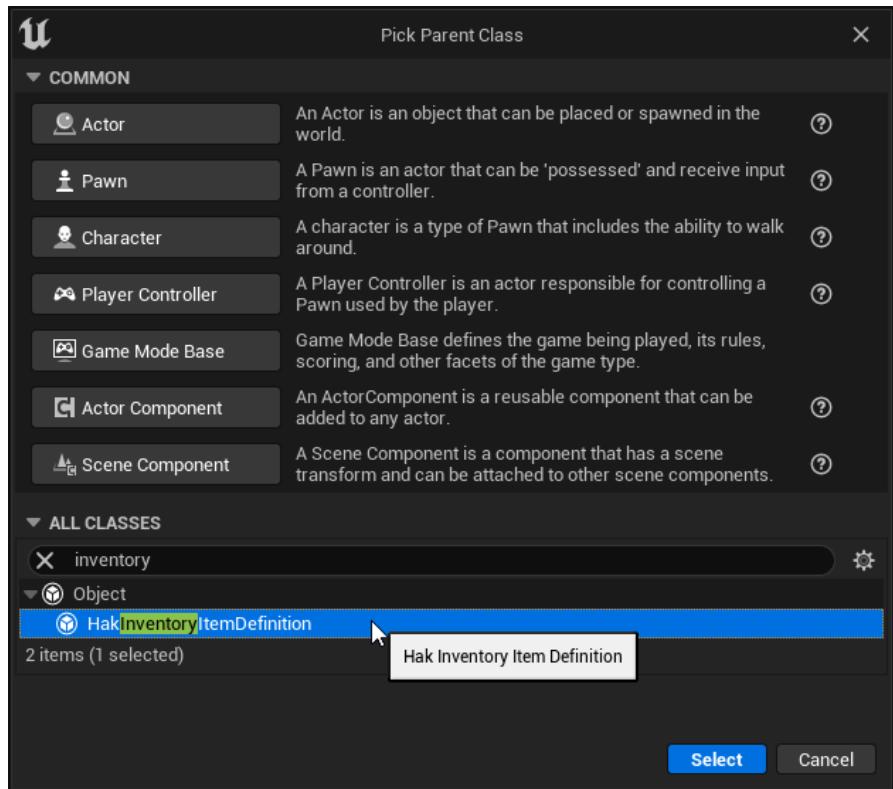
- AddInitialInventory 함수를 구현하자:

- AddInitialInventory의 대상이 되는 InitialInventoryItems의 멤버 변수를 추가하자:



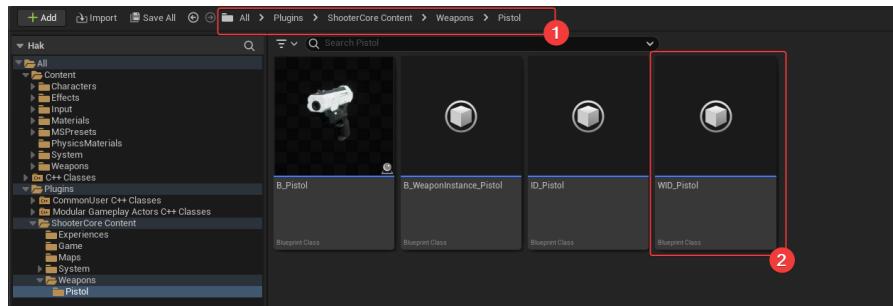
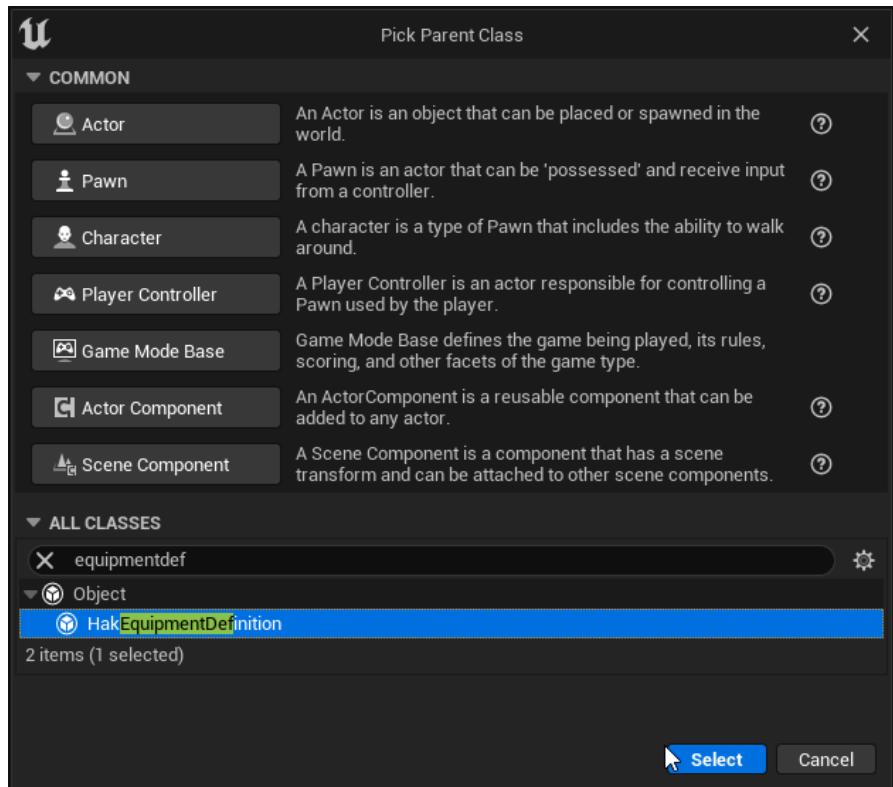
- 참고로, Class Reference와 Array로 설정하도록 한다:

- Object/Class의 구분은 색깔로 하면 된다~
- 그리고 여기에 Pistol 정보를 넣도록 하자
  - 우선 Inventory Definition을 Pistol에 대해 추가하는 에셋을 만들어야 한다:
    - 아래의 경로에 ID\_Pistol을 생성하자:

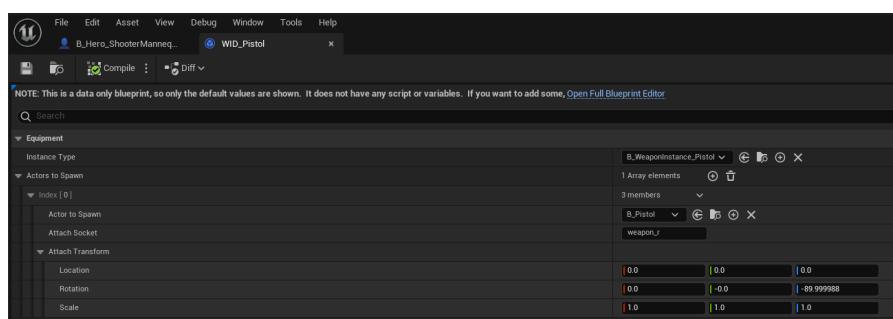


Equipment Definition을 추가하자:

- 아래의 경로에 WID\_Pistol을 생성하자:



- WID\_Pistol의 Actor 정보/Socket/AttachTransform을 알맞게 넣어 주자:



Inventory Definition에 Equipment Definition과 연결시킬 Fragment 해야 한다:

앞서 파일로 만들었던 HakInventoryFragment\_EquipableItem 정의하자:

```

#pragma once

#include "HakInventoryItemDefinition.h"
#include "Templates/SubclassOf.h"
#include "HakInventoryFragment_EquippableItem.generated.h"

/** forward declaration */
class UHakEquipmentDefinition;

UCLASS()
class UHakInventoryFragment_EquippableItem : public UHakInventoryItemFragment
{
    GENERATED_BODY()
public:
    UPROPERTY(EditAnywhere, Category=Hak)
    TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition;
};

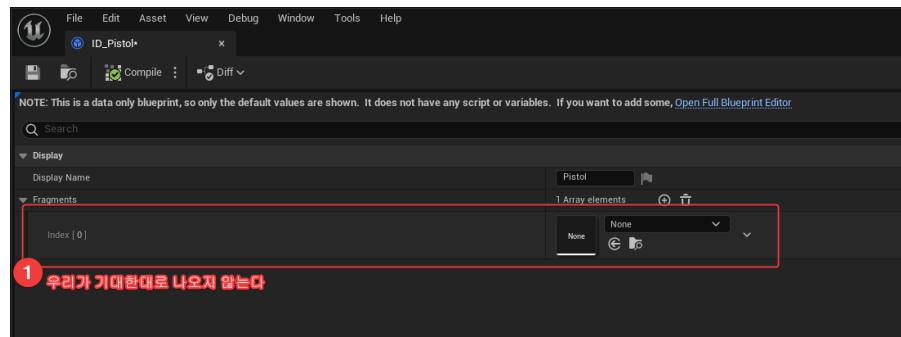
```

```

#include "HakInventoryFragment_EquippableItem.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryFragment_EquippableItem)

```

- ID\_Pistol에 Fragment를 추가하려고 했으나...



- HakInventoryDefintion과 HakInventoryItemFragment를 살펴보자:

```

#pragma once

#include "HakInventoryItemDefinition.generated.h"

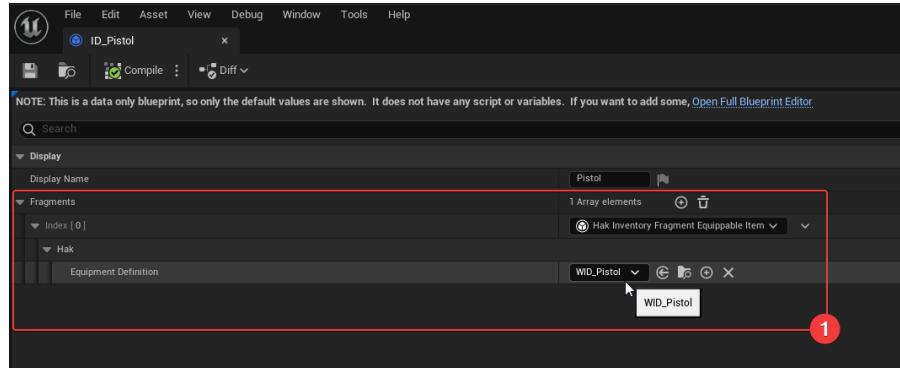
/**
 * Inventory에 대한 Fragment는 확 외단지 않을 수 있다:
 * - Lyra에서 사용하는 예시를 통해 이해해보자:
 *   - ULyraInventoryFragment_EquippableItem은 EquipmentItemDefinition을 가지고 있으며, 장착 가능한 아이템을 의미한다
 *   - ULyraInventoryFragment_SetStats는 아이템에 대한 정보를 가지고 있다
 *   - Rifle에 대한 SetStats으로 총알(Ammo)에 대한 장착 최대치와 현재 남은 잔탄 수를 예시로 들 수 있다
 *   - 등등...
 */
UCLASS(Abstract, DefaultToInstanced, EditInlineNew)
class UHakInventoryItemFragment : public UObject
{
    GENERATED_BODY()
public:
    UCLASS(Blueprintable)
class UHakInventoryItemDefinition : public UObject
{
    GENERATED_BODY()
public:
    UHakInventoryItemDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Inventory Item 정의(메타) 이름 */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category=Display)
    FText DisplayName;

    /** Inventory Item의 Component를 Fragment로 인식하면 된다 */
    UPROPERTY(EditDefaultsOnly, Instanced, BlueprintReadOnly, Category=Display)
    TArray<TObjectPtr<UHakInventoryItemFragment>> Fragments;
};

```

□ 아래와 같이 설정 가능하다:

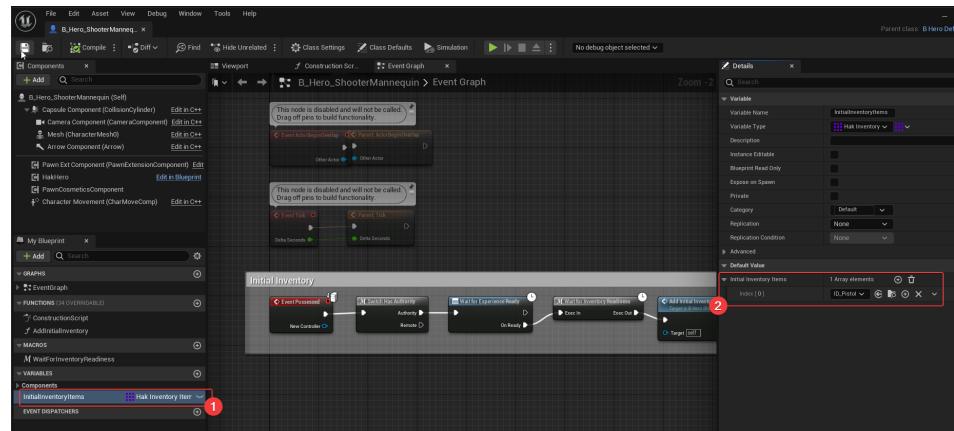


- DefaultToInstanced :

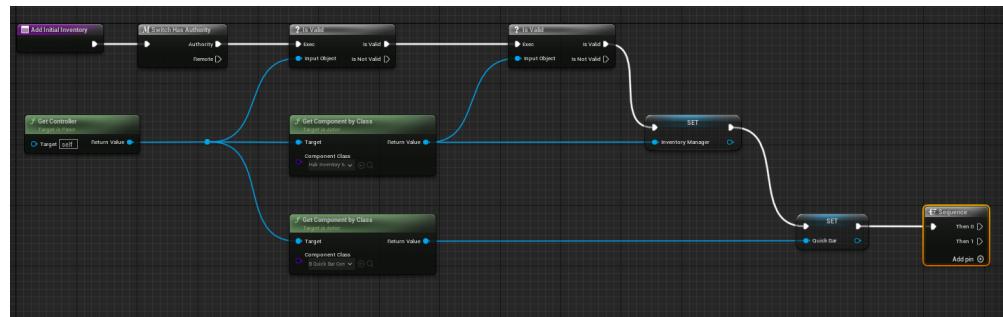
**DefaultToInstanced** All instances of this class are considered "instanced". Instanced classes (components) are duplicated upon construction. This Specifier is inherited by subclasses.

- 해당 속성을 상속받은 클래스의 인스턴스는 초기화될 경우, 마치 CDO(Class Default Object)과 같이 멤버변수와 같은 속성값들이 복사된다.
- 그리고, 해당 클래스를 상속받는 모든 클래스는 DefaultToInstanced 속성을 상속받는다.

□ B\_Hero\_ShooterMannequin의 InitialInventoryItems에 ID\_Pistol을 추가해주자:



- 앞서 보았던 WaitForInventoryReadiness Macro와 비슷하게 Controller/InventoryManagerComponent/B\_QuickBarComponent가 미리 준비하여 로컬 변수로 캐싱해두자:

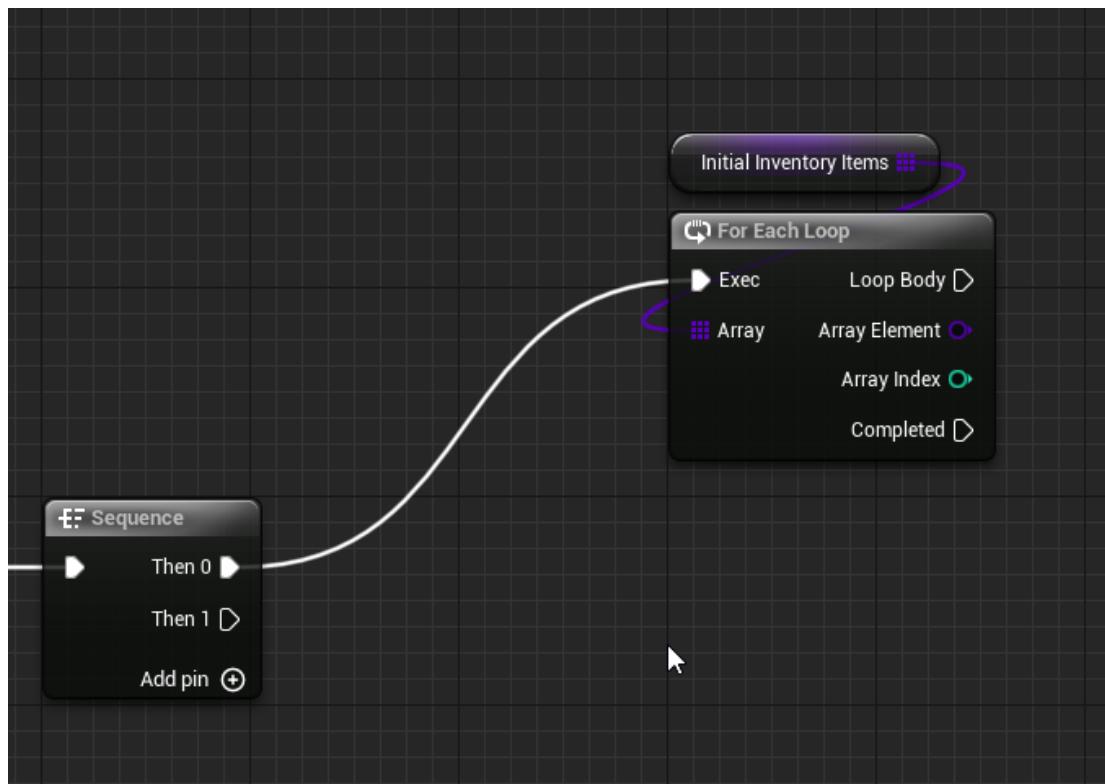


- 이제 각각 Sequence0과 Sequence1의 로직을 C++ 코드로 작성해가며, 구현해보자

## AddItemDefinition/AddItemToSlot

### ▼ 편집기

- B\_Hero\_ShooterMannequin의 AddInitialInventory Seq0을 구현하자:



- InitialInventoryItems을 순회하며, InventoryManagerComponent에 해당 아이템을 추가하고, QuickBarComponent에도 Slot에 추가해야 한다

- HakInventoryManagerComponent::AddItemDefinition:

```

#pragma once

#include "Components/ActorComponent.h"
#include "HakInventoryManagerComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakInventoryItemDefinition; 1

/** Inventory Item 단위 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryEntry
{
GENERATED_BODY()

UPROPERTY()
TObjectPtr<UHakInventoryItemInstance> Instance = nullptr;
};

/** Inventory Item 관리 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryList
{
GENERATED_BODY()

FHakInventoryList(UActorComponent* InOwnerComponent = nullptr)
: OwnerComponent(InOwnerComponent)
{}

UPROPERTY()
TArray<FHakInventoryEntry> Entries;

UPROPERTY()
TObjectPtr<UActorComponent> OwnerComponent;
};

/** 
 * PlayerController의 Component로서 Inventory를 관리한다
 * - 사실 UActorComponent 상속이 아닌 UControllerComponent를 상속받아도 될거 같은데... 일단 Lyra 기준으로 UActorComponent를 상속받고 있다
 */
UCLASS(BlueprintType)
class UHakInventoryManagerComponent : public UActorComponent
{
GENERATED_BODY()
public:
UHakInventoryManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

UFUNCTION(BlueprintCallable, Category=Inventory)
UHakInventoryItemInstance* AddItemDefinition(TSubclassOf<UHakInventoryItemDefinition> ItemDef); 2

UPROPERTY()
FHakInventoryList InventoryList;
};

```

```

UHakInventoryItemInstance* UHakInventoryManagerComponent::AddItemDefinition(TSubclassOf<UHakInventoryItemDefinition> ItemDef)
{
    UHakInventoryItemInstance* Result = nullptr;
    if (ItemDef)
    {
        Result = InventoryList.AddEntry(ItemDef);
    }
    return Result;
}

```

## □ FHakInventoryList::AddEntry

```

#pragma once

#include "Components/ActorComponent.h"
#include "HakInventoryManagerComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakInventoryItemDefinition;

/** Inventory Item 단위 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryEntry
{
GENERATED_BODY()

UPROPERTY()
TObjectPtr<UHakInventoryItemInstance> Instance = nullptr;
};

/** Inventory Item 관리 객체 */
USTRUCT(BlueprintType)
struct FHakInventoryList
{
GENERATED_BODY()

FHakInventoryList(UActorComponent* InOwnerComponent = nullptr)
: OwnerComponent(InOwnerComponent)
{}

UHakInventoryItemInstance* AddEntry(TSubclassOf<UHakInventoryItemDefinition> ItemDef);

UPROPERTY()
TArray<FHakInventoryEntry> Entries;

UPROPERTY()
TObjectPtr<UActorComponent> OwnerComponent;
};

/*
 * PlayerController의 Component로서 Inventory를 관리한다
 * - 사실 UActorComponent 상속이 아닌 UControllerComponent를 상속받아도 될거 같은데... 일단 Lyra 기준으로 UActorComponent를 상속받고 있다
 */
UCLASS(BlueprintType)
class UHakInventoryManagerComponent : public UActorComponent
{
GENERATED_BODY()

public:
    UHakInventoryManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** InventoryItemDefinition을 통해, InventoryList에 추가하여 관리하며, InventoryItemInstance를 반환한다 */
    UFUNCTION(BlueprintCallable, Category=Inventory)
    UHakInventoryItemInstance* AddItemDefinition(TSubclassOf<UHakInventoryItemDefinition> ItemDef);

    UPROPERTY()
    FHakInventoryList InventoryList;
};

```

```

UHakInventoryItemInstance* FHakInventoryList::AddEntry(TSubclassOf<UHakInventoryItemDefinition> ItemDef)
{
    UHakInventoryItemInstance* Result = nullptr;
    check(ItemDef);
    check(OwnerComponent);

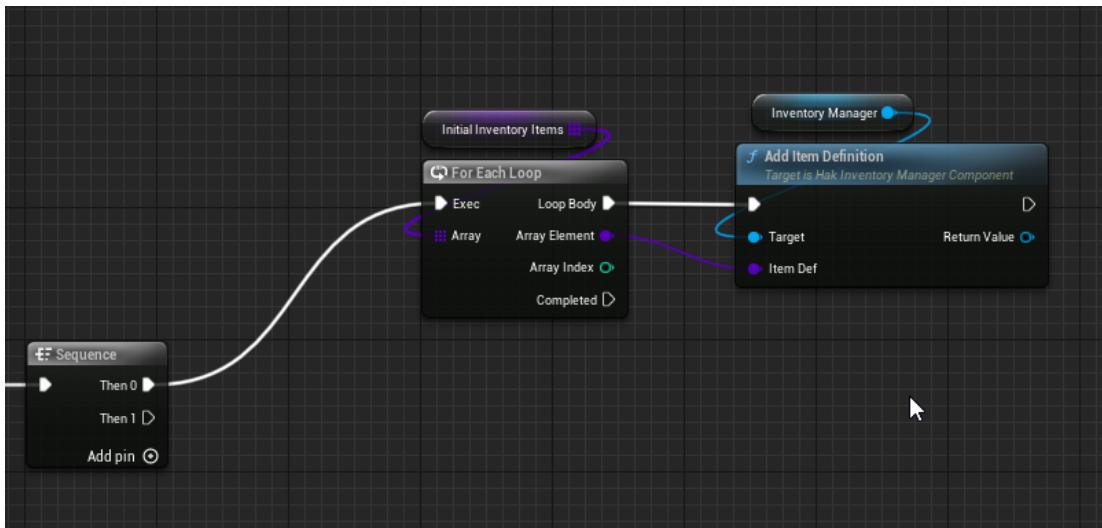
    AActor* OwningActor = OwnerComponent->GetOwner();
    check(OwningActor->HasAuthority());

    FHakInventoryEntry& NewEntry = Entries.AddDefaulted_GetRef();
    NewEntry.Instance = NewObject<UHakInventoryItemInstance>(OwningActor);
    NewEntry.Instance->ItemDef = ItemDef;

    Result = NewEntry.Instance;
    return Result;
}

```

AddItemDefinition을 BP 로직에 추가하자:



## □ HakQuickBarComponent::AddItemToSlot

```
#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakEquipmentInstance;

/**
 * HUD의 QuickBar를 생각하면 된다:
 * - 흔히 MMORPG에서는 ShortCut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(Slate)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 Slate Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable)
    void AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item); 1

    /** HUD QuickBar Slot 개수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveSlotIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};
```

```

void UHakQuickBarComponent::AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item)
{
    // 해당 로직을 보면, Slots는 Add로 동적 추가가 아닌, Index에 바로 넣는다:
    // - 그럼 미리 Pre-size 했다는 것인데 이는 BeginPlay()에서 진행한다
    if (Slots.IsValidIndex(SlotIndex) && (Item != nullptr))
    {
        if (Slots[SlotIndex] == nullptr)
        {
            Slots[SlotIndex] = Item;
        }
    }
}

```

## □ HakQuickBarComponent::BeginPlay

```

#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakEquipmentInstance;

/**
 * HUD의 QuickBar를 생각하면 된다:
 * - 흔히 MMORPG에서는 ShortCut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(State)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 Slate Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * ControllerComponent interface
     */
    virtual void BeginPlay() override; 1

    /**
     * member methods
     */
    UFUNCTION(BlueprintCallable)
    void AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item);

    /** HUD QuickBar Slot 갯수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveTabIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};

```

```

#include "HakQuickBarComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakQuickBarComponent)

UHakQuickBarComponent::UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{ }

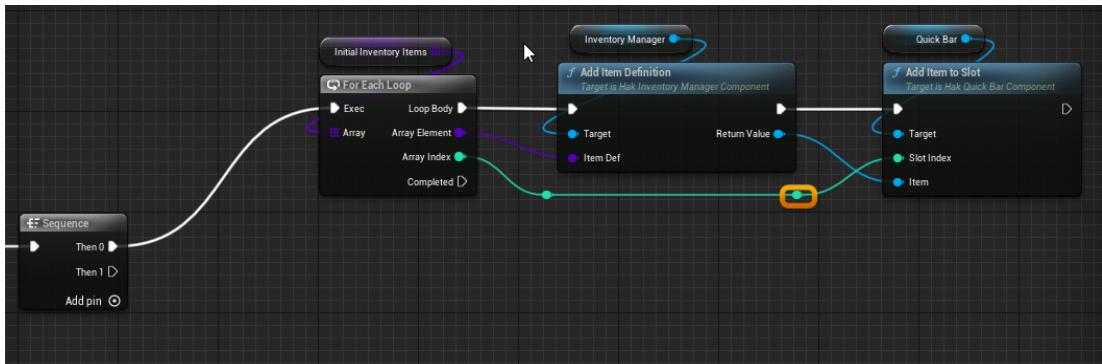
void UHakQuickBarComponent::BeginPlay()
{
    // NumSlots에 따라 미리 Slots을 할당한다
    if (Slots.Num() < NumSlots)
    {
        Slots.AddDefaulted(NumSlots - Slots.Num());
    }

    // 반드시 BeginPlay() 블러주는 것을 까먹지 말자!
    Super::BeginPlay();
}

void UHakQuickBarComponent::AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item)
{
    // 해당 로직을 보면, Slots는 Add로 등적 추가가 아닌, Index에 바로 넣는다:
    // - 그럼 미리 Pre-size 했다는 것인데 이는 BeginPlay()에서 진행한다
    if (Slots.IsValidIndex(SlotIndex) && (Item != nullptr))
    {
        if (Slots[SlotIndex] == nullptr)
        {
            Slots[SlotIndex] = Item;
        }
    }
}

```

- AddItemToSlot을 넣어주자:



## SetActiveSlotIndex

### ▼ 펼치기

- HakQuickBarComponent::SetActiveSlotIndex 더미 만들기:

```

#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakEquipmentInstance;

/**
 * HUD의 QuickBar를 생각하면 된다:
 * - 흔히 MMORPG에서는 ShortCut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(Slate)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 State Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * ControllerComponent interface
     */
    virtual void BeginPlay() override;

    /**
     * member methods
     */
    UFUNCTION(BlueprintCallable)
    void AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item);

    UFUNCTION(BlueprintCallable, Category="Hak")
    void SetActiveSlotIndex(int32 NewIndex);

    /** HUD QuickBar Slot 개수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveSlotIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};

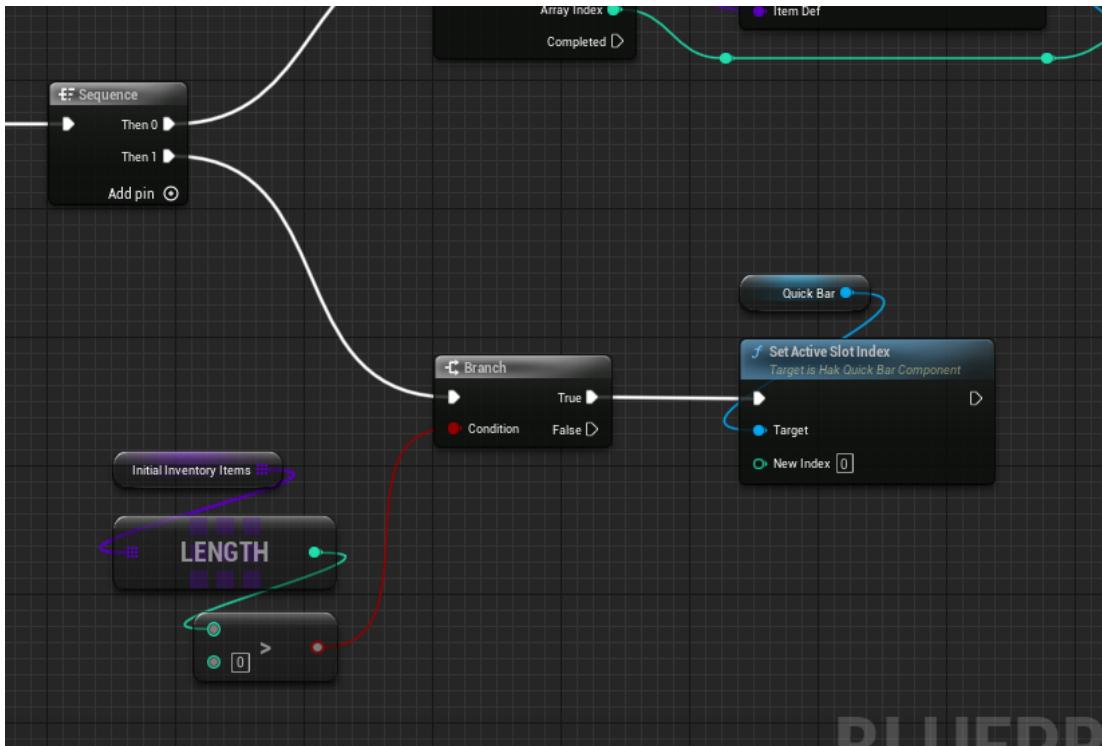
```

```

void UHakQuickBarComponent::SetActiveSlotIndex(int32 NewIndex)
{
    {
        :
    }
}

```

□ B\_Hero\_ShooterMannequin의 AddInitialInventory Seq1을 구현하자:



#### □ SetActiveSlotIndex 구현:

```

void UHakQuickBarComponent::SetActiveSlotIndex(int32 NewIndex)
{
    if (Slots.IsValidIndex(NewIndex) && (ActiveSlotIndex != NewIndex))
    {
        // UnequipItem/EquipItem을 통해, NewIndex를 통해 할당된 Item을 창자 및 업데이트를 진행한다
        UnequipItemInSlot();
        ActiveSlotIndex = NewIndex;
        EquipItemInSlot();
    }
}

```

#### □ UnequipItemInSlot 구현:

```


/***
 * HUD의 QuickBar를 생각하면 된다:
 * - 훗날 MMORPG에서는 ShortCut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(Slate)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 Slate Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * ControllerComponent interface
     */
    virtual void BeginPlay() override;

    /**
     * member methods
     */
    void UnequipItemInSlot(); 1
    UFUNCTION(BlueprintCallable)
    void AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item);

    UFUNCTION(BlueprintCallable, Category="Hak")
    void SetActiveSlotIndex(int32 NewIndex);

    /** HUD QuickBar Slot 갯수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveSlotIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};


```

```


void UHakQuickBarComponent::UnequipItemInSlot()
{
    // 참고로 QuickBar는 Controller에 붙어있는 Component이지만, EquipmentManagerComponent는 Controller가 소유(Possess)하고 있는 Pawn의 Component이다
    if (UHakEquipmentManagerComponent* EquipmentManager = FindEquipmentManager())
    {
        // 현재 장착된 Item에 있다면,
        if (EquippedItem)
        {
            // EquipmentManager를 통해, Pawn의 장비를 해제시킨다
            EquipmentManager->UnequipItem(EquippedItem);

            // 그리고 Controller에도 EquipItem의 상태를 없는 것으로 업데이트한다
            EquippedItem = nullptr;
        }
    }
}


```

## □ FindEquipmentManager 구현:

```

#pragma once

#include "Components/ControllerComponent.h"
#include "HakQuickBarComponent.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;
class UHakEquipmentInstance;
class UHakEquipmentManagerComponent; 1

/**
 * HUD의 QuickBar를 생각하면 된다:
 * - 흔히 MMORPG에서는 ShortCut HUD를 연상하면 된다
 *
 * 해당 Component는 ControllerComponent로서, PlayerController에 의해 조작계 중 하나로 생각해도 된다
 * - HUD(Slate)와 Inventory/Equipment(Gameplay)의 다리(Bridge) 역할하는 Component로 생각하자
 * - 해당 Component는 Lyra의 HUD 및 Slate Widget을 다루면서 다시 보게될 예정이다
 */
UCLASS(Blueprintable, meta = (BlueprintSpawnableComponent))
class UHakQuickBarComponent : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakQuickBarComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * ControllerComponent interface
     */
    virtual void BeginPlay() override;

    /**
     * member methods
     */
    UHakEquipmentManagerComponent* FindEquipmentManager() const; 2
    void UnequipItemInSlot();

    UFUNCTION(BlueprintCallable)
    void AddItemToSlot(int32 SlotIndex, UHakInventoryItemInstance* Item);

    UFUNCTION(BlueprintCallable, Category="Hak")
    void SetActiveSlotIndex(int32 NewIndex);

    /** HUD QuickBar Slot 갯수 */
    UPROPERTY()
    int32 NumSlots = 3;

    /** HUD QuickBar Slot 리스트 */
    UPROPERTY()
    TArray<TObjectPtr<UHakInventoryItemInstance>> Slots;

    /** 현재 활성화된 Slot Index (아마 Lyra는 딱 한개만 Slot이 활성화되는가보다?) */
    UPROPERTY()
    int32 ActiveSlotIndex = -1;

    /** 현재 장착한 장비 정보 */
    UPROPERTY()
    TObjectPtr<UHakEquipmentInstance> EquippedItem;
};

```

```

UHakEquipmentManagerComponent* UHakQuickBarComponent::FindEquipmentManager() const
{
    if (AController* OwnerController = Cast<AController>(GetOwner()))
    {
        if (APawn* Pawn = OwnerController->GetPawn())
        {
            return Pawn->FindComponentByClass<UHakEquipmentManagerComponent>();
        }
    }
    return nullptr;
}

```

- 아직 우리는 아래와 같이 UnequipItem에 대한 구현을 진행해야 한다:
  - HakQuickBarComponent의 Equip/Unequip 구현을 완성하고 넘어가도록 하자:

## □ HakQuickBarComponent::EquipItemInSlot

```

void UHakQuickBarComponent::EquipItemInSlot()
{
    check(Slots.IsValidIndex(ActiveSlotIndex));
    check(EquippedItem != nullptr);

    // 현재 활성화된 ActiveSlotIndex를 활용하여 활성화된 InventoryItemInstance를 찾는다
    if (UHakInventoryItemInstance* SlotItem = Slots[ActiveSlotIndex])
    {
        // Slot Item을 통해 (InventoryItemInstance) InventoryFragment_EquipableItem의 Fragment를 찾는다:
        // - 찾는 것이 실패했다면, 장착할 수 없는 Inventory Item임을 의미하다
        if (const UHakInventoryFragment_EquipableItem* EquipInfo = SlotItem->FindFragmentByClass<UHakInventoryFragment_EquipableItem>())
        {
            // EquipableItem에서 EquipmentDefinition을 찾는다:
            // - EquipmentDefinition이 있어야, 장착할 수 있다
            TSubclassOf<UHakEquipmentDefinition> EquipDef = EquipInfo->EquipmentDefinition;
            if (EquipDef)
            {
                // 아래는 Unequip이랑 비슷하게 EquipmentManager를 통해 장착한다
                if (UHakEquipmentManagerComponent* EquipmentManager = FindEquipmentManager())
                {
                    EquippedItem = EquipmentManager->EquipItem(EquipDef);
                    if (EquippedItem)
                    {
                        // EquippedItem에는 앞서 보았던 Instigator로 Slot을 대상으로 넣는다
                        EquippedItem->Instigator = SlotItem;
                    }
                }
            }
        }
    }
}

```

## □ HakInventoryItemInstance::FindFragmentByClass 구현:

```

#pragma once

#include "UObject/Object.h"
#include "Templates/SubclassOf.h"
#include "HakInventoryItemInstance.generated.h"

/** forward declarations */
class UHakInventoryItemDefinition;
class UHakInventoryItemFragment;

/**
 * 해당 클래스는 Inventory Item의 인스턴스로 볼 수 있다
 */
UCLASS(BlueprintType)
class UHakInventoryItemInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    const UHakInventoryItemFragment* FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const;

    template <typename ResultClass>
    const ResultClass* FindFragmentByClass() const
    {
        return (ResultClass*)FindFragmentByClass(ResultClass::StaticClass());
    }

    /** Inventory Item의 인스턴스에는 무엇으로 정의되었는지 메타 클래스인 HakInventoryItemDefinition을 들고 있다 */
    UPROPERTY()
    TSubclassOf<UHakInventoryItemDefinition> ItemDef;
};

```

```

#include "HakInventoryItemInstance.h"
#include "HakInventoryItemDefinition.h"
#include "UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryItemInstance)

UHakInventoryItemInstance::UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer)
: Super(ObjectInitializer)
{}

1 const UHakInventoryItemFragment* UHakInventoryItemInstance::FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const
{
    if ((ItemDef != nullptr) && (FragmentClass != nullptr))
    {
        // HakInventoryItemDefinition은 모든 멤버 변수가 EditDefaultsOnly로 선언되어 있으므로, GetDefault로 가져와도 무관하다
        // - Fragment 정보는 Instance가 아닌 Definition에 있다
        return GetDefault<UHakInventoryItemDefinition>(ItemDef)->FindFragmentByClass(FragmentClass);
    }
    return nullptr;
}

```

## □ HakInventoryItemDefinition::FindFragmentByClass 구현:

```
#pragma once
#include "HakInventoryItemDefinition.generated.h"

/* Inventory에 대한 Fragment는 확 와닫지 않을 수 있다:
 * - Lyra에서 사용하는 예시를 통해 이해해보자:
 *   - ULyraInventoryFragment_EquipableItem은 EquipmentItemDefinition을 가지고 있으며, 장착 가능한 아이템을 의미한다
 *   - ULyraInventoryFragment_SetStats는 아이템에 대한 정보를 가지고 있다
 *   - Rifle에 대한 SetStats으로 총알(Ammo)에 대한 장착 최대치와 현재 남은 잔탄 수를 예시로 들 수 있다
 *   - 등등...
 */
UCLASS(Abstract, DefaultToInstanced, EditInlineNew)
class UHakInventoryItemFragment : public UObject
{
GENERATED_BODY()
public:
};

UCLASS(Blueprintable)
class UHakInventoryItemDefinition : public UObject
{
GENERATED_BODY()
public:
    UHakInventoryItemDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
    const UHakInventoryItemFragment* FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const; 1
    /* Inventory Item 정의(메타) 이름 */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category=Display)
    FText DisplayName;

    /* Inventory Item의 Component를 Fragment로 인식하면 된다 */
    UPROPERTY(EditDefaultsOnly, Instanced, BlueprintReadOnly, Category=Display)
    TArray<TObjectPtr<UHakInventoryItemFragment>> Fragments;
};

const UHakInventoryItemFragment* UHakInventoryItemDefinition::FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const
{
    if (FragmentClass)
    {
        // Fragments를 순회하며, IsA()를 통해 해당 클래스를 가지고 있는지 확인한다:
        for (UHakInventoryItemFragment* Fragment : Fragments)
        {
            if (Fragment && Fragment->IsA(FragmentClass))
            {
                return Fragment;
            }
        }
    }
    return nullptr;
}
```

## □ EquipmentManagerComponent

### □ UnequipItem 구현하자:

- 앞서 QuickBarComponent의 UnequipItemInSlot()에서 호출하였다:

```
void UHakQuickBarComponent::UnequipItemInSlot()
{
    // 참고로 QuickBar는 Controller에 붙어있는 Component이지만, EquipmentManagerComponent는 Controller가 소유(Possess)하고 있는 Pawn의 Component이다
    if (UHakEquipmentManagerComponent* EquipmentManager = FindEquipmentManager())
    {
        // 현재 장착된 Item에 있다면,
        if (EquippedItem)
        {
            // EquipmentManager를 통해, Pawn의 장비를 해제시킨다
            EquipmentManager->UnequipItem(EquippedItem); 1
            // 그리고 Controller에도 EquipItem의 상태를 없는 것으로 업데이트한다
            EquippedItem = nullptr;
        }
    }
}
```

- 내용을 구현하자:

```


    /**
     * Pawn의 Component로서 장착물에 대한 관리를 담당한다
     */
    UCLASS(BlueprintType)
    class UHakEquipmentManagerComponent : public UPawnComponent
    {
        GENERATED_BODY()
    public:
        UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
        void UnequipItem(UHakEquipmentInstance* ItemInstance);
        UPROPERTY()
        FHakEquipmentList EquipmentList;
    };


```

1

```


void UHakEquipmentManagerComponent::UnequipItem(UHakEquipmentInstance* ItemInstance)
{
    if (ItemInstance)
    {
        // 해당 함수는 BP의 Event노드를 호출해준다 (자세한건 해당 함수 구현하면서 보자)
        ItemInstance->OnUnequipped();

        // EquipmentList에 제거해준다:
        // - 제거하는 과정을 통해 주가되었던 Actor Instance를 제거를 진행한다
        EquipmentList.RemoveEntry(ItemInstance);
    }
}


```

## □ HakEquipmentInstance::OnUnequipped:

```


#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"
#include "Containers/Array.h"
#include "HakEquipmentInstance.generated.h"

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 함수
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();
    /**
     * interfaces
     */
    virtual void OnUnequipped(); 2
    /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance들 */
    UPROPERTY()
    TArray< TObjectPtr< AActor >> SpawnedActors;
};


```

1

2

## □ FHakEquipmentList::RemoveEntry

```


    /**
     * 참고로 EquipmentInstance의 인스턴스를 Entry에서 관리하고 있다:
     * - HakEquipmentList는 생성된 객체를 관리한다고 보면 된다
     */
    USTRUCT(BlueprintType)
    struct FHakEquipmentList
    {
        GENERATED_BODY()

        FHakEquipmentList(UActorComponent* InOwnerComponent = nullptr)
            : OwnerComponent(InOwnerComponent)
        {}

        void RemoveEntry(UHakEquipmentInstance* Instance); 1

        /** 장착물에 대한 관리 리스트 */
        UPROPERTY()
        TArray<FHakAppliedEquipmentEntry> Entries;

        UPROPERTY()
        TObjectPtr<UActorComponent> OwnerComponent;
    };


```

```


void FHakEquipmentList::RemoveEntry(UHakEquipmentInstance* Instance)
{
    // 단순히 그냥 Entries를 순회하며, Instance를 찾아서
    for (auto EntryIt = Entries.CreateIterator(); EntryIt; ++EntryIt)
    {
        FHakAppliedEquipmentEntry& Entry = *EntryIt;
        if (Entry.Instance == Instance)
        {
            // Actor 제거 작업 및 iterator를 통한 안전하게 Array에서 제거 진행
            Instance->DestroyEquipmentActors();
            EntryIt.RemoveCurrent();
        }
    }
}


```

HakEquipmentInstance::DestroyEquipmentActors

```

#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"
#include "Containers/Array.h"
#include "HakEquipmentInstance.generated.h"

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 함수
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();

    void DestroyEquipmentActors(); 1

    /**
     * interfaces
     */
    virtual void OnUnequipped();
};

/** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후후, QuickBarComponent에서 보게 될것이다) */
UPROPERTY()
TObjectPtr<UObject> Instigator;

/** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance들 */
UPROPERTY()
TArray<TObjectPtr<AActor>> SpawnedActors;
};

```

```

#include "HakEquipmentInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakEquipmentInstance)

UHakEquipmentInstance::UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{ }

void UHakEquipmentInstance::DestroyEquipmentActors()
{
    // 참고로 장착물이 복수개의 Actor Mesh로 구성되어 있을 수도 있다
    // - 같은 Lv10이었지만, 상체와 하체로 같이 구성되어있을 수도 있으니깐?
    for (AActor* Actor : SpawnedActors)
    {
        if (Actor)
        {
            Actor->Destroy();
        }
    }
}

void UHakEquipmentInstance::OnUnequipped()
{
    K2_OnUnequipped();
}

```

## □ HakEquipmentManagerComponent::EquipItem

```


    /**
     * Pawn의 Component로서 장착물에 대한 관리를 담당한다
     */
    UCLASS(BlueprintType)
    class UHakEquipmentManagerComponent : public UPawnComponent
    {
        GENERATED_BODY()
    public:
        UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
        1 UHakEquipmentInstance* EquipItem(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition);
        void UnequipItem(UHakEquipmentInstance* ItemInstance);

        UPROPERTY()
        FHakEquipmentList EquipmentList;
    };


```

```


UHakEquipmentInstance* UHakEquipmentManagerComponent::EquipItem(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition)
{
    UHakEquipmentInstance* Result = nullptr;
    if (EquipmentDefinition)
    {
        Result = EquipmentList.AddEntry(EquipmentDefinition);
        if (Result)
        {
            Result->OnEquipped();
        }
    }
    return Result;
}


```

## □ FHakEquipmentList::AddEntry

```


    /**
     * 참고로 EquipmentInstance의 인스턴스를 Entry에서 관리하고 있다:
     * - HakEquipmentList는 생성된 객체를 관리한다고 보면 된다
     */
    USTRUCT(BlueprintType)
    struct FHakEquipmentList
    {
        GENERATED_BODY()

        FHakEquipmentList(UActorComponent* InOwnerComponent = nullptr)
            : OwnerComponent(InOwnerComponent)
        {}

        1 UHakEquipmentInstance* AddEntry(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition);
        void RemoveEntry(UHakEquipmentInstance* Instance);

        /** 장착물에 대한 관리 리스트 */
        UPROPERTY()
        TArray<FHakAppliedEquipmentEntry> Entries;

        UPROPERTY()
        TObjectPtr<UActorComponent> OwnerComponent;
    };


```

```

EUhakEquipmentInstance* FHakEquipmentList::AddEntry(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition)
{
    UHakEquipmentInstance* Result = nullptr;
    check(EquipmentDefinition != nullptr);
    check(OwnerComponent);
    check(OwnerComponent->GetOwner()>>HasAuthority());

    // EquipmentDefinition의 멤버 변수들은 EditDefaultsOnly로 정의되어 있어 GetDefault로 들고 와도 우리에게 필요한 것들이 모두 들어있다
    const UHakEquipmentDefinition* EquipmentCDO = GetDefault<UHakEquipmentDefinition>(EquipmentDefinition);

    TSubclassOf<UHakEquipmentInstance> InstanceType = EquipmentCDO->InstanceType;
    if (!InstanceType)
    {
        InstanceType = UHakEquipmentInstance::StaticClass();
    }

    // Entries에 추가해주자
    FHakAppliedEquipmentEntry& NewEntry = Entries.AddDefaulted_GetRef();
    NewEntry.EquipmentDefinition = EquipmentDefinition;
    NewEntry.Instance = NewObject<UHakEquipmentInstance>(OwnerComponent->GetOwner(), InstanceType);
    Result = NewEntry.Instance;

    // ActorsToSpawn을 통해, Actor들을 인스턴스화 해주자
    // - 어디에? EquipmentInstance에!
    Result->SpawnEquipmentActors(EquipmentCDO->ActorsToSpawn);

    return Result;
}

```

## □ HakEquipmentInstance::SpawnEquipmentActors

```

#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"
#include "Containers/Array.h"
#include "HakEquipmentInstance.generated.h"

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 함수
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_Unequipped();

    APawn* GetPawn() const;
    void SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn); 1
    void DestroyEquipmentActors();

    /**
     * interfaces
     */
    virtual void OnUnequipped();

    /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance들 */
    UPROPERTY()
    TArray< TObjectPtr<AActor>> SpawedActors;
};

```

```

EAPawn* UHakEquipmentInstance::GetPawn() const
{
    // 우리는 EquipmentInstance를 생성할 당시에, Outer를 Pawn으로 두었다!
    return Cast<APawn>(GetOuter());
}

void UHakEquipmentInstance::SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn)
{
    if (APawn* OwningPawn = GetPawn())
    {
        // 현재 Owner인 Pawn의 RootComponent를 AttachTarget 대상으로 한다
        USceneComponent* AttachTarget = OwningPawn->GetRootComponent();
        if (ACharacter* Char = Cast<ACharacter>(OwningPawn))
        {
            // 만약 캐릭터라면, SkeletalMeshComponent가 있으면 GetMesh로 반환하여, 여기에 블인다
            AttachTarget = Char->GetMesh();
        }

        for (const FHakEquipmentActorToSpawn& SpawnInfo : ActorsToSpawn)
        {
            // SpawnActorDeferred는 FinishSpawning을 호출해야 Spawn이 완성된다 (즉, 작성자에게 코드로서 Ownership이 있다는 의미)
            AActor* NewActor = GetWorld()->SpawnActorDeferred<AActor>(SpawnInfo.ActorToSpawn, FTransform::Identity, OwningPawn);
            NewActor->FinishSpawning(FTransform::Identity, /*bIsDefaultTransform=*/true);

            // Actor의 RelativeTransform을 AttachTransform으로 설정
            NewActor->SetActorRelativeTransform(SpawnInfo.AttachTransform);

            // AttachTarget에 블이자 (Actor -> Actor)
            NewActor->AttachToComponent(AttachTarget, FAttachmentTransformRules::KeepRelativeTransform, SpawnInfo.AttachSocket);

            SpawnerActors.Add(NewActor);
        }
    }
}

```

## □ HakEquipmentInstance::OnEquipped

```

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 핸들
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();

    APawn* GetPawn() const;
    void SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn);
    void DestroyEquipmentActors();

    /**
     * interfaces
     */
    virtual void OnEquipped();
    virtual void OnUnequipped();

    /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance를 */
    UPROPERTY()
    TArray<TObjectPtr<AActor>> SpawnerActors;
};

```

```

void UHakEquipmentInstance::OnEquipped()
{
    K2_OnEquipped();
}

```

## □ 실행해보면, 아래와 같이 총이 부탁되었음을 확인할 수 있다:

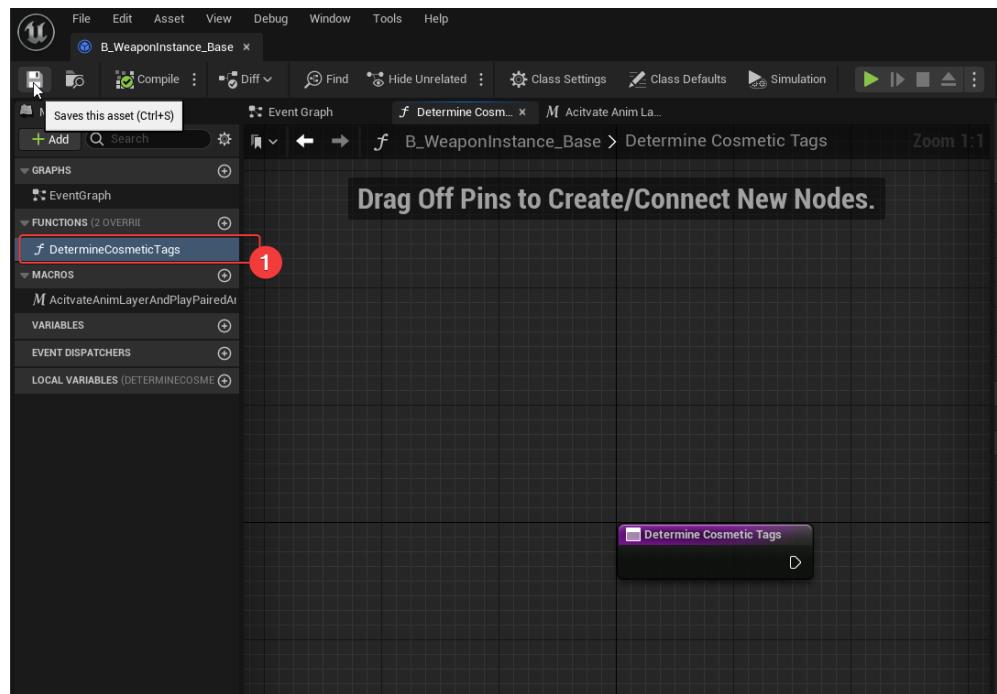


- 아직 우리는 무기의 애니메이션을 바인딩시키지 않았다:
  - 이를 위해 앞서 구현했던 OnEquipped의 Event를 활용해야 한다

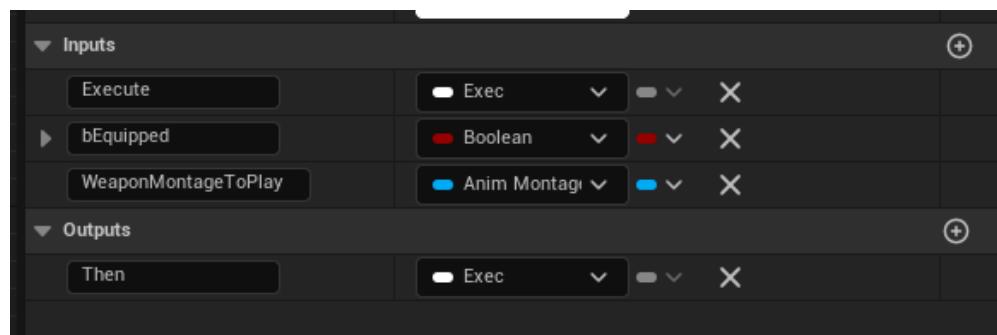
## B\_WeaponInstance\_Base

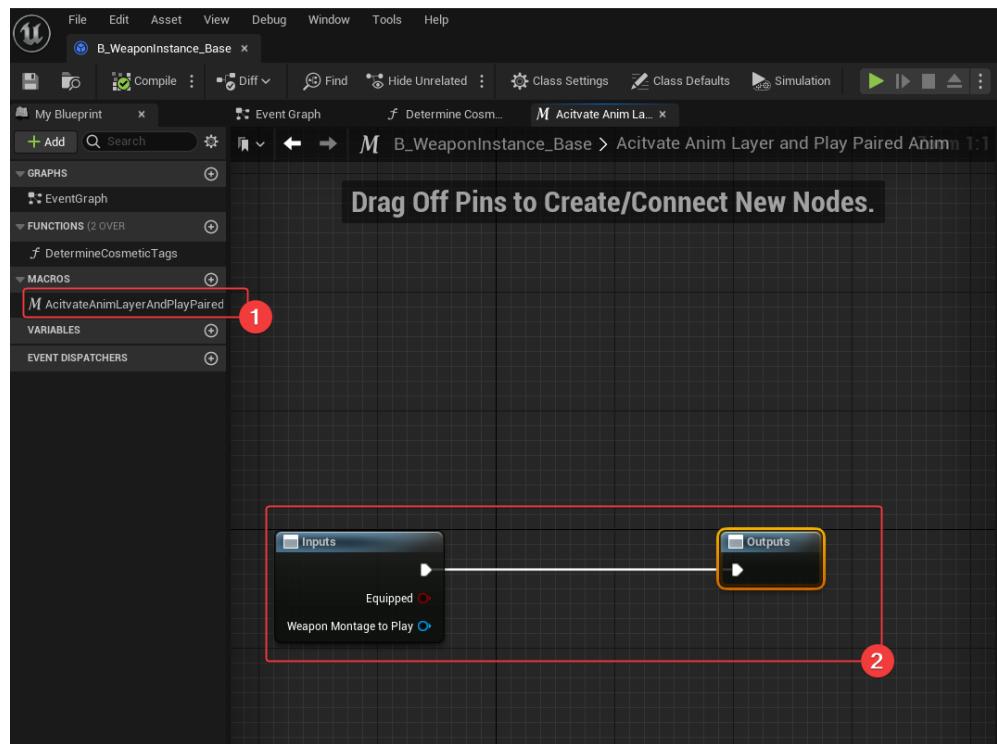
### ▼ 펼치기

- 이전 시간에 우리는 B\_WeaponInstance\_Base를 생성하였다:
  - 해당 클래스는 HakEquipmentInstance를 상속받고 있으며, 앞서 우리는 OnEquipped를 정의하였고, 해당 BP에서 해당 이벤트를 사용할 수 있다
- OnEquipped:
  - 우선 DetermineCosmeticTags 함수와 ActivateAnimLayerAndPlayPairedAnim을 정의하자
    - DetermineCosmeticTags:

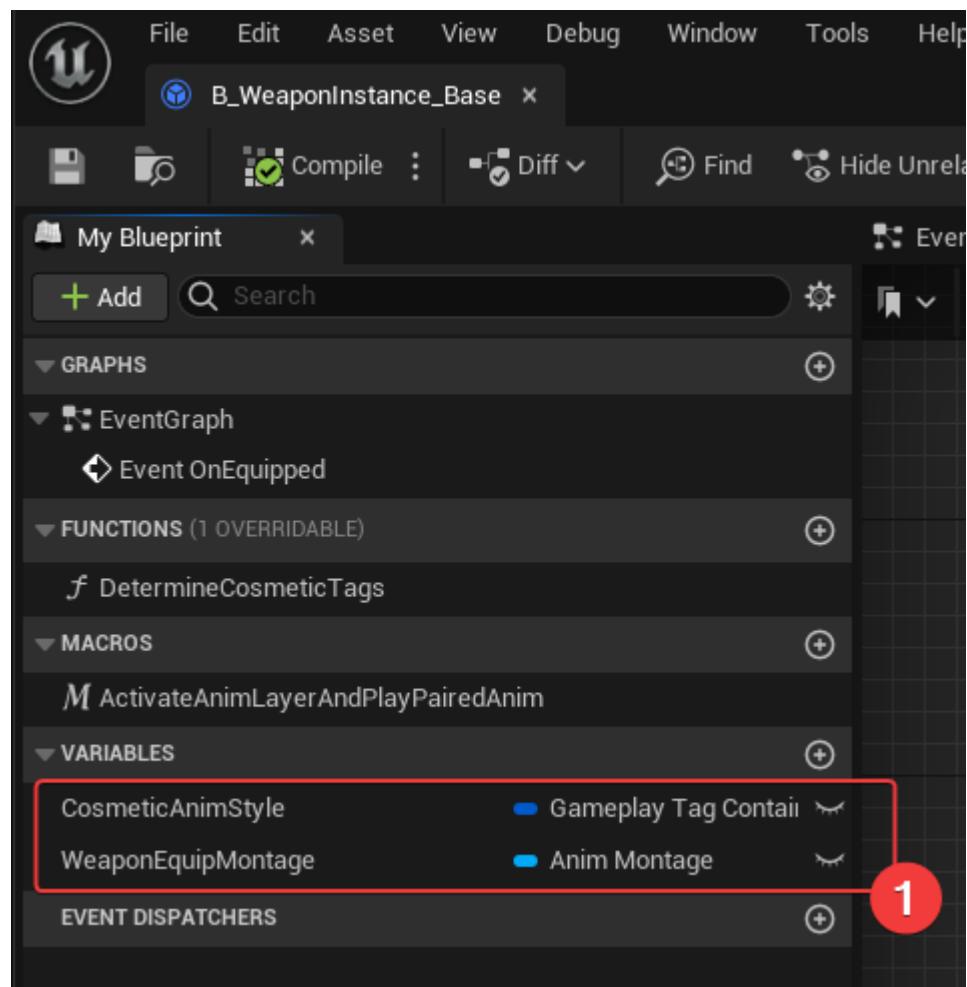


- ActivateAnimLayerAndPlayPairedAnim:





B\_WeaponInstance\_Base의 멤버 변수를 정의하자:

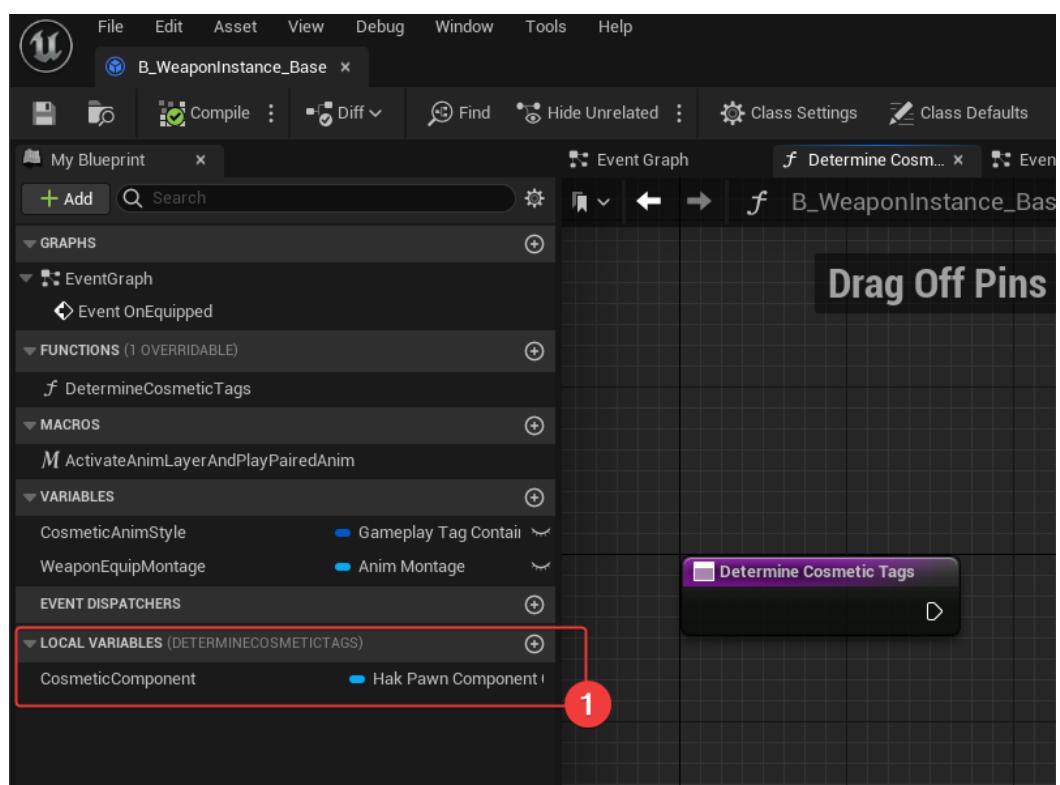


- OnEquipped의 Event 로직 형태를 완성하자:



- DetermineCosmeticTags:

  - Local 변수인 CosmeticComponent 정의:



- HakEquipmentInstance::GetPawn을 BP 호출 가능한 함수로 변환:

```

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 함수
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();

    UFUNCTION(BlueprintPure, Category=Equipment)
    APawn* GetPawn() const;

    void SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn);
    void DestroyEquipmentActors();

    /**
     * interfaces
     */
    virtual void OnEquipped();
    virtual void OnUnequipped();

    /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (후후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance들 */
    UPROPERTY()
    TArray<TObjectPtr<AAActor>> SpawnedActors;
};


```

- HakPawnComponent\_CharacterParts::GetCombinedTags를 BP 호출 가능한 함수로 변환:

```

/** 
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    USkeletalMeshComponent* GetParentMeshComponent() const;
    USceneComponent* GetSceneComponentToAttachTo() const;
    void BroadcastChanged();

    FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);
    void RemoveCharacterPart(FHakCharacterPartHandle Handle);

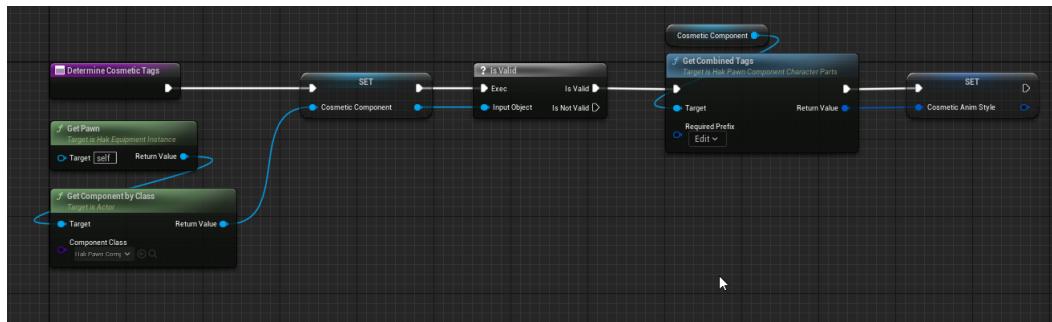
    UFUNCTION(BlueprintCallable, BlueprintPure=false, Category=Cosmetics)
    FGameplayTagContainer GetCombinedTags(FGameplayTag RequiredPrefix) const;

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};


```

- DetermineCosmeticTags 완성시키기:



## □ ActivateAnimLayerAndPlayPairedAnim

### □ HakWeaponInstance::PickAnimLayer 정의:

```

UCLASS()
class UHakWeaponInstance : public UHakEquipmentInstance
{
    GENERATED_BODY()
public:
    UHakWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Weapon에 적용할 AnimLayer를 선택하여 반환 */
    UPROPERTY(BlueprintCallable, BlueprintPure=false, Category=Animation)
    TSubclassOf<UAnimInstance> PickBestAnimLayer(bool bEquipped, const FGameplayTagContainer& CosmeticTags) const; 1

    /** Weapon에 Equip/Unequip에 대한 Animation Set 정보를 들고 있다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet EquippedAnimSet;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet UnequippedAnimSet;
};

TSubclassOf<UAnimInstance> UHakWeaponInstance::PickBestAnimLayer(bool bEquipped, const FGameplayTagContainer& CosmeticTags) const
{
    const FHakAnimLayerSelectionSet& SetToQuery = (bEquipped ? EquippedAnimSet : UnequippedAnimSet);
    return SetToQuery.SelectBestLayer(CosmeticTags);
}

```

### □ HakAnimLayerSelectionSet::SelectBestLayer:

```

USTRUCT(BlueprintType)
struct FHakAnimLayerSelectionSet
{
    GENERATED_BODY()

    /** CosmeticTags 기반하여, 적절한 AnimLayer를 반환한다 */
    TSubclassOf<UAnimInstance> SelectBestLayer(const FGameplayTagContainer& CosmeticTags) const;

    /** 앞서 보았던 HakAnimBodyStyleSelection의 MeshRule과 같이 AnimInstance의 Rule을 가진 LayerRules로 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<FHakAnimLayerSelectionEntry> LayerRules;

    /** 디폴트 Layer */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TSubclassOf<UAnimInstance> DefaultLayer;
};

TSubclassOf<UAnimInstance> FHakAnimLayerSelectionSet::SelectBestLayer(const FGameplayTagContainer& CosmeticTags) const
{
    for (const FHakAnimLayerSelectionEntry& Rule : LayerRules)
    {
        if ((Rule.Layer != nullptr) && CosmeticTags.HasAll(Rule.RequiredTags))
        {
            return Rule.Layer;
        }
    }
    return DefaultLayer;
}

```

### □ HakEquipmentInstance::GetTypedPawn 정의:

```

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 핸들
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();

    UFUNCTION(BlueprintPure, Category=Equipment)
    APawn* GetPawn() const;

    /**
     * DeterminesOutputType은 C++ 정의에는 APawn* 반환하지만, BP에서는 PawnType에 따라 OutputType이 결정되도록 리다이렉트(Redirect)한다
     */
    UFUNCTION(BlueprintPure, Category=Equipment, meta=(DeterminesOutputType=PawnType))
    APawn* GetTypedPawn(TSubclassOf<APawn> PawnType) const;

    void SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn);
    void DestroyEquipmentActors();

    /**
     * interfaces
     */
    virtual void OnEquipped();
    virtual void OnUnequipped();

    /** 이전 InventoryItemInstance에 의해 활성화되었는지 (후후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipmentDefinition에 맞게 Spawn된 Actor Instance들 */
    UPROPERTY()
    TArray< TObjectPtr<AActor>> SpawnedActors;
};

```

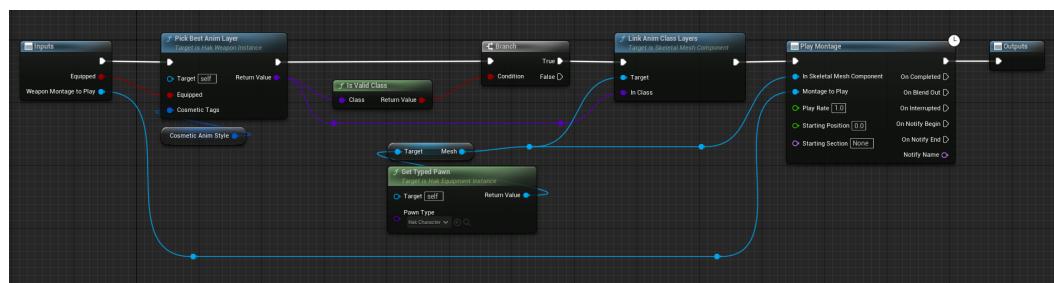
1

```

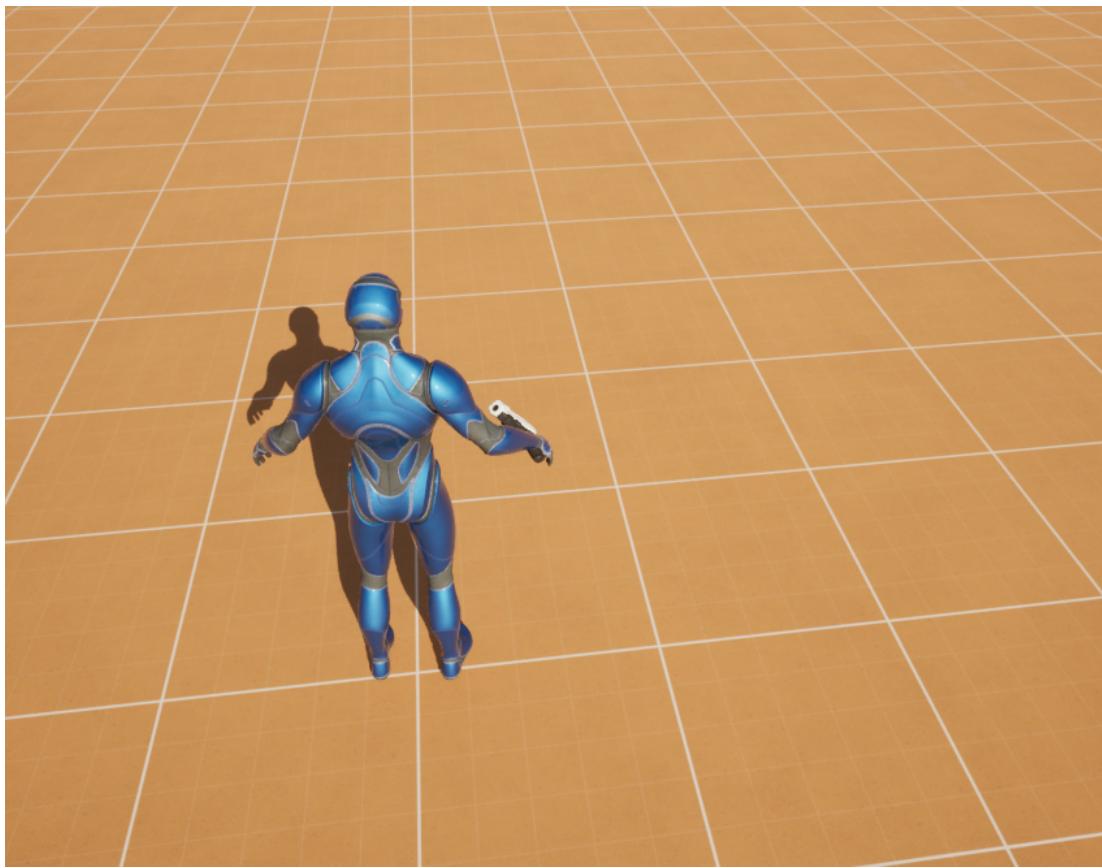
APawn* UHakEquipmentInstance::GetTypedPawn(TSubclassOf<APawn> PawnType) const
{
    APawn* Result = nullptr;
    if (UClass* ActualPawnType = PawnType)
    {
        if (GetOuter()->IsA(ActualPawnType))
        {
            Result = Cast<APawn>(GetOuter());
        }
    }
    return Result;
}

```

## □ BP 로직을 완성하자:



## □ 이제 실행해보자:



- 아직 변화가 없다...

## AnimLayer 설정:

### ▼ 펼치기

- 우선 앞서 구현한 로직이 잘 들어오는지 확인 겸, HakAnimLayerSelectionSet::SelectBestLayer를 디버깅해보자:

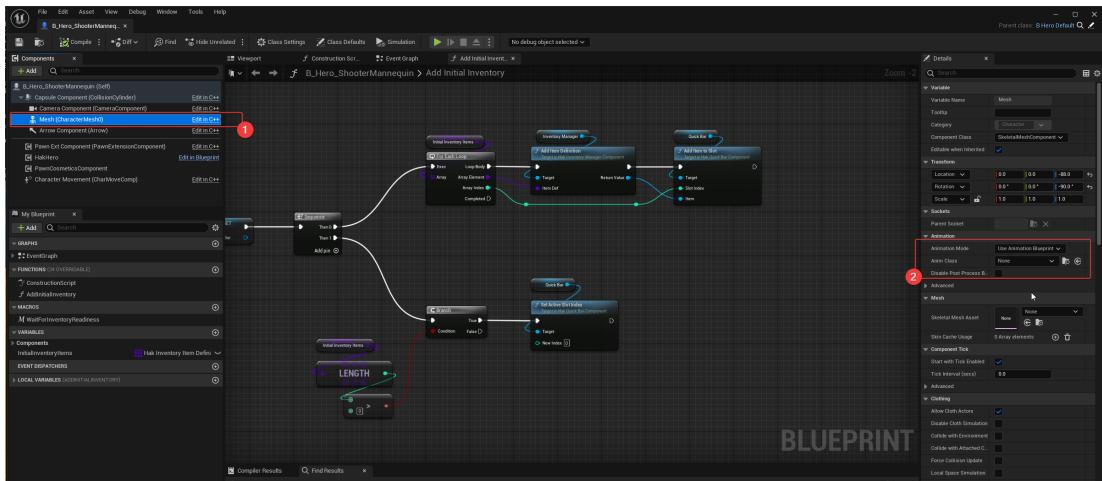
```

5   TSubclassOf<UAnimInstance> FHakAnimLayerSelectionSet::SelectBestLayer(const FGameplayTagContainer& CosmeticTags) const
6   {
7       for (const FHakAnimLayerSelectionEntry& Rule : LayerRules)
8       {
9           if ((Rule.Layer != nullptr) && CosmeticTags.HasAll(Rule.RequiredTags))
10          {
11              return Rule.Layer;
12          }
13      }
14  }

```

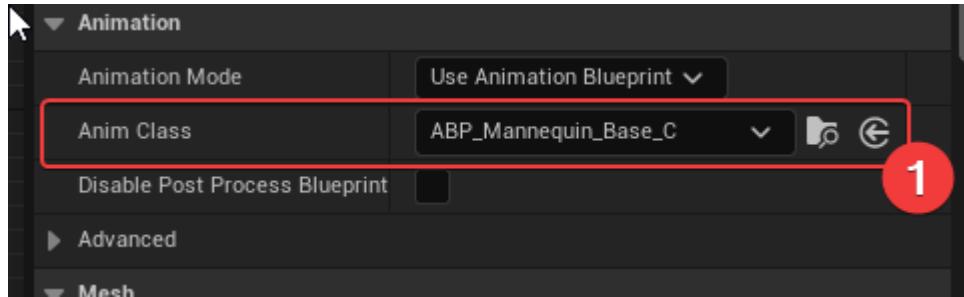
- 잘 들어온다!

- 지금까지 무기에 대한 애니메이션 설정을 진행했는데, 우리의 B\_Hero\_ShooterCoreMannequin의 AnimLayer를 확인해보자:

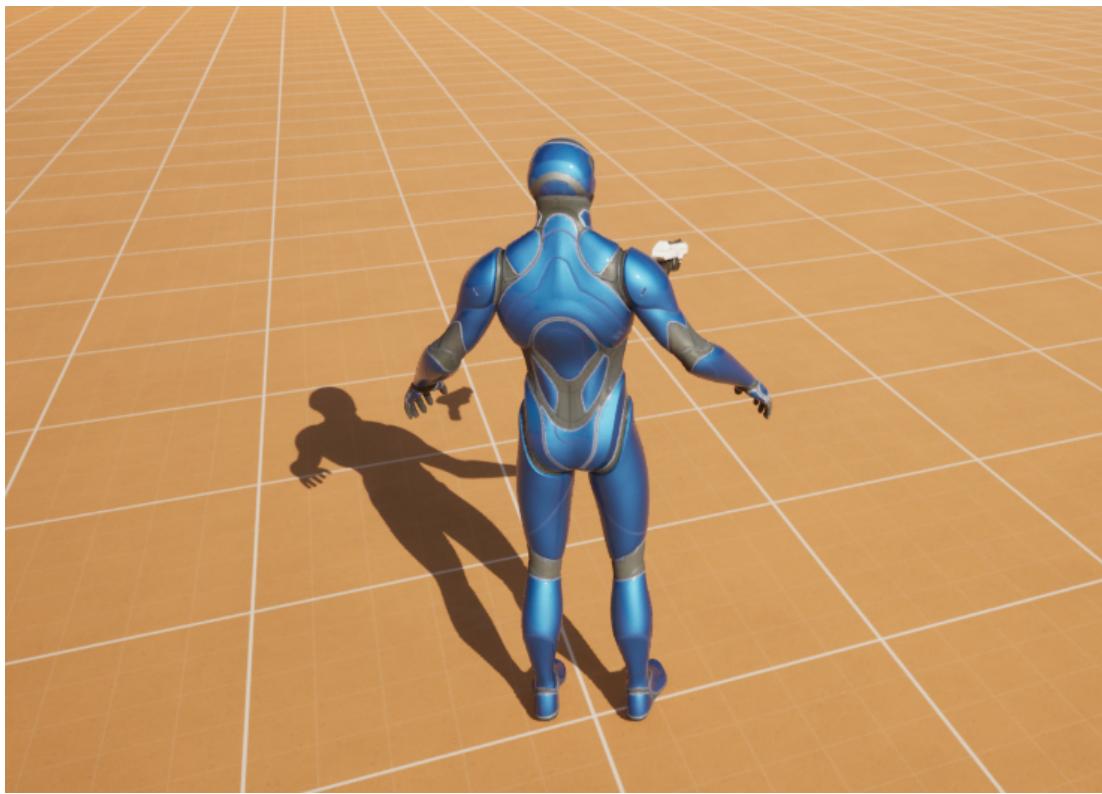


□ 아! 설정이 되어있지 않았다!

- ABP\_Mannequin\_Base로 설정해주자:

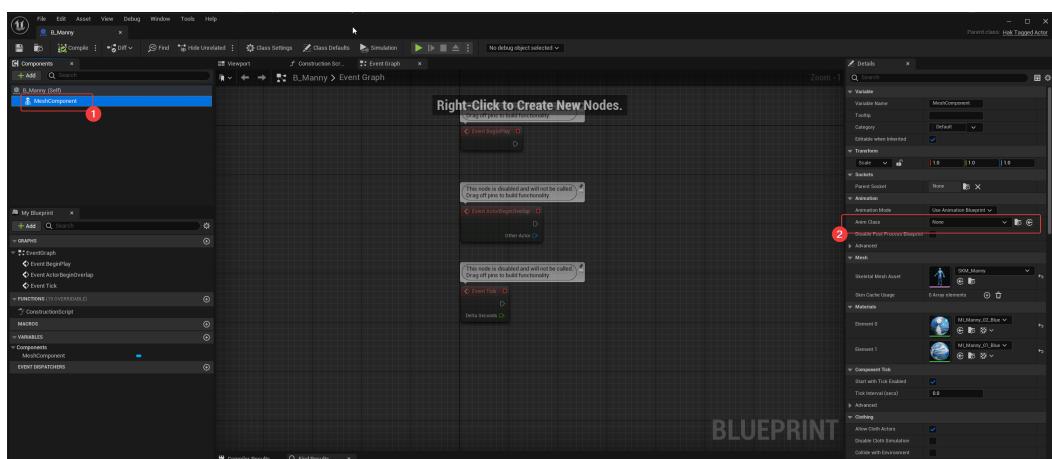
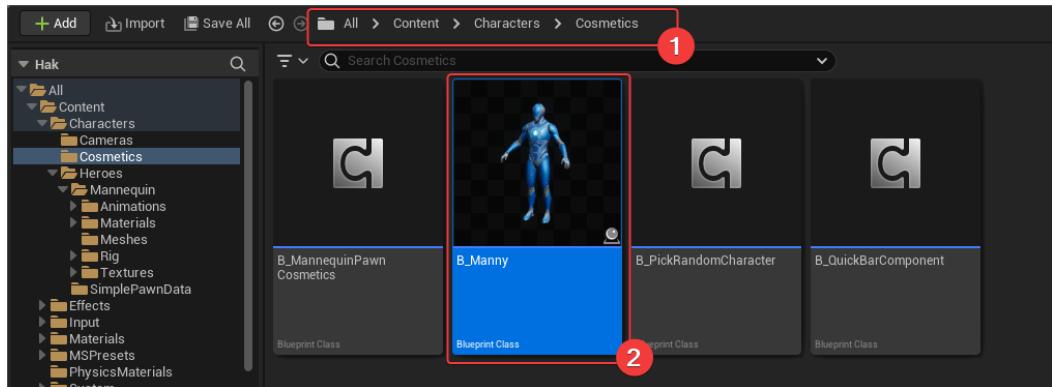


□ 실행하면, 아래와 같이 총만... Animation이 진행된다:



□ 생각해보면, 우리의 BodyMesh는 Cosmetic에 의해, B\_Manny로 설정되었다:

□ B\_Manny의 AnimLayer를 확인해주자:

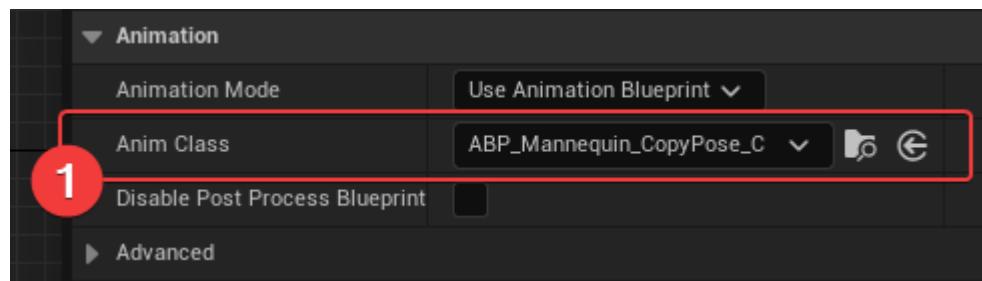


- 설정이 되어있지 않았다!

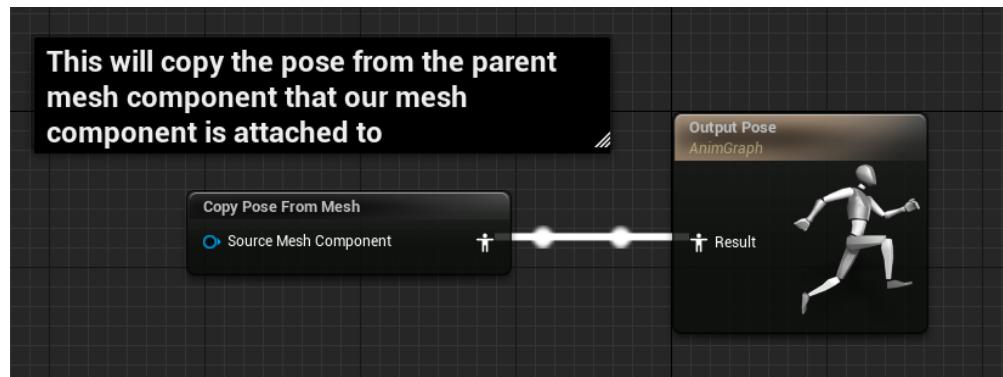


앞서 B\_Manny는 이전에 B\_PickRandomCharacter로  
ControllerComponent\_CharacterParts에 의해  
PawnComponent\_CharacterParts로 설정되었다!

- 우리는 사실 Invis가 이미 ABP\_Mannequin\_Base를 가지고 있다:
  - 우리가 원하는건 B\_Manny가 B\_Hero\_ShooterMannequin의 애니메이션  
을 상속받기를 원하는건데...
- ABP\_Mannequin\_CopyPose를 활용하면 된다:
  - ABP\_Mannequin\_CopyPose의 에셋을 Lyra로부터 가져오자



- 해당 ABP\_Mannequin\_CopyPose를 살펴보면, 아래와 같다:



- 설명 그대로, Parent MeshComponent로부터 가져온다!
  - 우리의 경우, B\_Hero\_ShooterCoreMannequin에서 가져오게 될 것이다.

- 이제 잘된다:

[https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/5eb4d7f4-f0c5-4cc8-a270-11ee186dec47/UnrealEditor\\_SYTIg2jgJl.mp4](https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/5eb4d7f4-f0c5-4cc8-a270-11ee186dec47/UnrealEditor_SYTIg2jgJl.mp4)