□ 개요

○ 애셋

. 커브 데이터 : **TimeOfDayAsset**, **TimeOfDayPreset**, **TimeOfDayWeatherAsset**, **TimeOfDayWeatherPreset**

○ 시간 계산 & 라이팅/웨더 애셋 선택

. **TimeOfDayTime** : 시간 계산(로컬, 서버), TimeZone 에 따른 지리/물리 기반 Sun&Moon 계산.

. **TimeOfDayEnvironmentSource** : 라이팅 애셋, 웨더 애셋을 실시간 선택 및 세팅.

○ 그래픽 요소

. **TimeOfDaySky** : 스카이 박스. 해, 달, 별, 하늘색 등을 표현.

. **TimeOfDayVolumetricCloud** : 3D 클라우드.

. **TimeOfDayRainSnow** : 나이아가라를 활용한 비, 눈 표현.

. **TimeOfDayThunderLightning** : 천둥, 번개.

. **TimeOfDayBackLight** : 로컬 캐릭터 하이라이트 연출.

. **TimeOfDayWind** : 바람 제어.

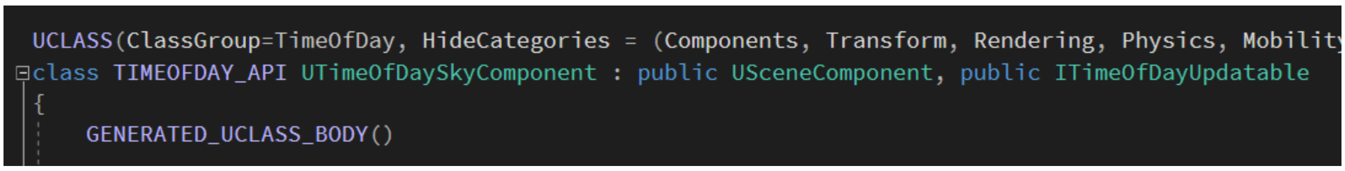
○ 커브 데이터 보간 및 적용

. **TimeOfDaySequencer** : 라이팅/웨더 애셋을 이용 시간별로 프로퍼티 값들을 보간하여 적용.

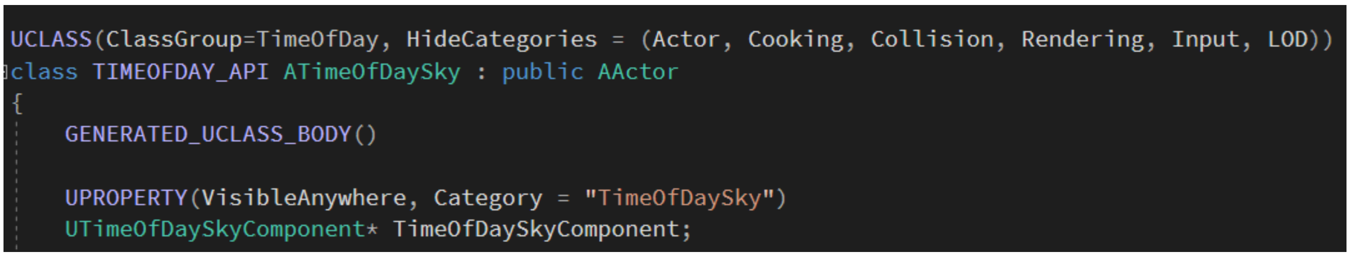
○ 위 기능들을 통합 관리 : **TimeOfDayWorld**.

□ 모듈화

○ 컴포넌트 기반으로 구현하고 액터를 제공.

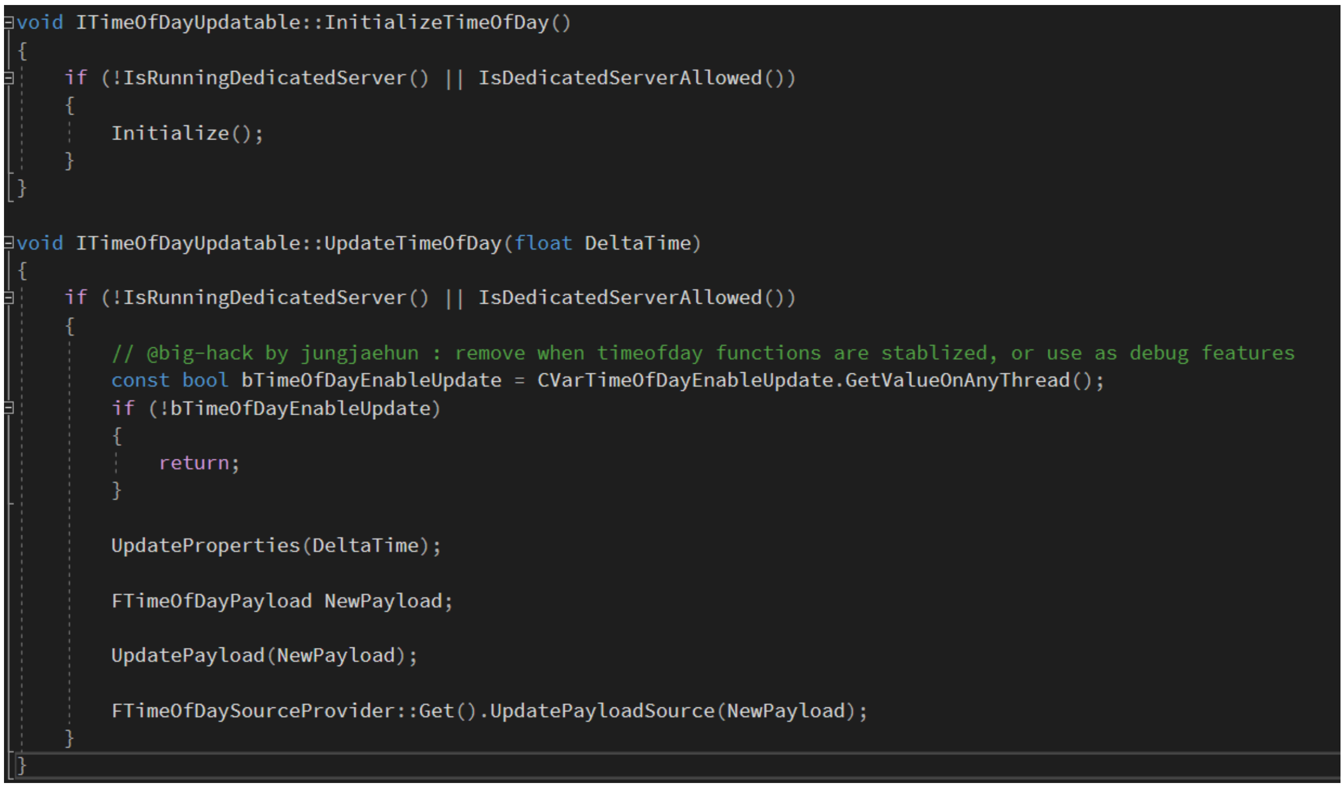


```
UCLASS(ClassGroup=TimeOfDay, HideCategories = (Components, Transform, Rendering, Physics, Mobilit
class TIMEOFDAY_API UTimeOfDaySkyComponent : public USceneComponent, public ITimeOfDayUpdatable
{
    GENERATED_UCLASS_BODY()
```



```
UCLASS(ClassGroup=TimeOfDay, HideCategories = (Actor, Cooking, Collision, Rendering, Input, LOD))
class TIMEOFDAY_API ATimeOfDaySky : public AActor
{
    GENERATED_UCLASS_BODY()

    UPROPERTY(VisibleAnywhere, Category = "TimeOfDaySky")
    UTimeOfDaySkyComponent* TimeOfDaySkyComponent;
```

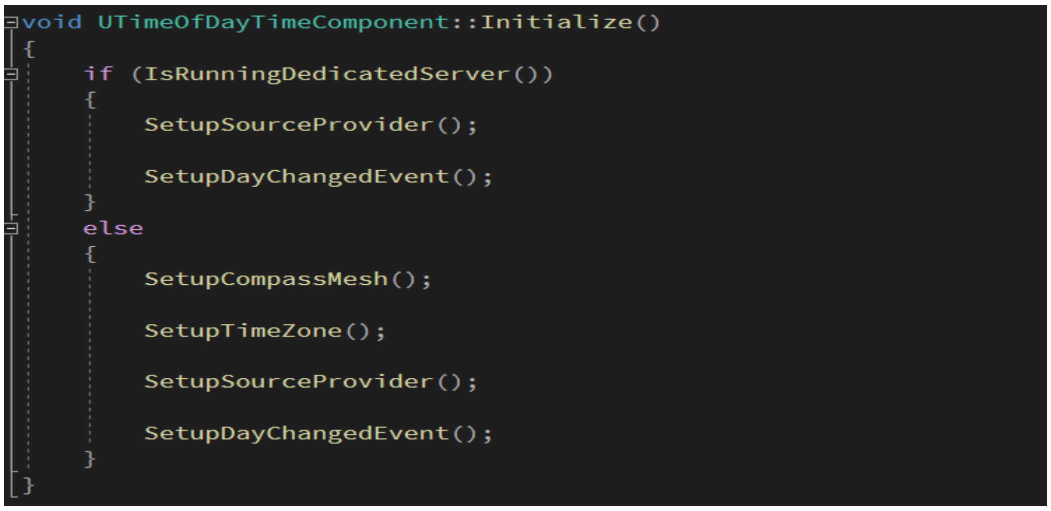
○ 컴포넌트는 ITimeOfDayUpdatable 을 상속받아 인터페이스 규칙에 따라 작동.



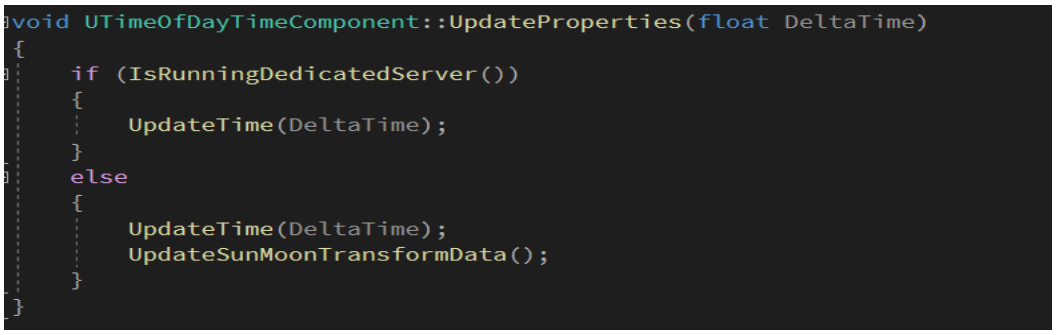
```
void ITimeOfDayUpdatable::InitializeTimeOfDay()
{
    if (!IsRunningDedicatedServer() || IsDedicatedServerAllowed())
    {
        Initialize();
    }
}

void ITimeOfDayUpdatable::UpdateTimeOfDay(float DeltaTime)
{
    if (!IsRunningDedicatedServer() || IsDedicatedServerAllowed())
    {
        // @big-hack by jungjaehun : remove when timeofday functions are stablized, or use as debug features
        const bool bTimeOfDayEnableUpdate = CVarTimeOfDayEnableUpdate.GetValueOnAnyThread();
        if (!bTimeOfDayEnableUpdate)
        {
            return;
        }

        UpdateProperties(DeltaTime);

        FTimeOfDayPayload NewPayload;

        UpdatePayload(NewPayload);

        FTimeOfDaySourceProvider::Get().UpdatePayloadSource(NewPayload);
    }
}
```

. 서버 허용 유무에 따라 Enable/Disable 되고, **'초기화 -> 프로퍼티 업데이트 -> 페이로드 업데이트 -> 페이로드 전달'** 의 과정을 거침.

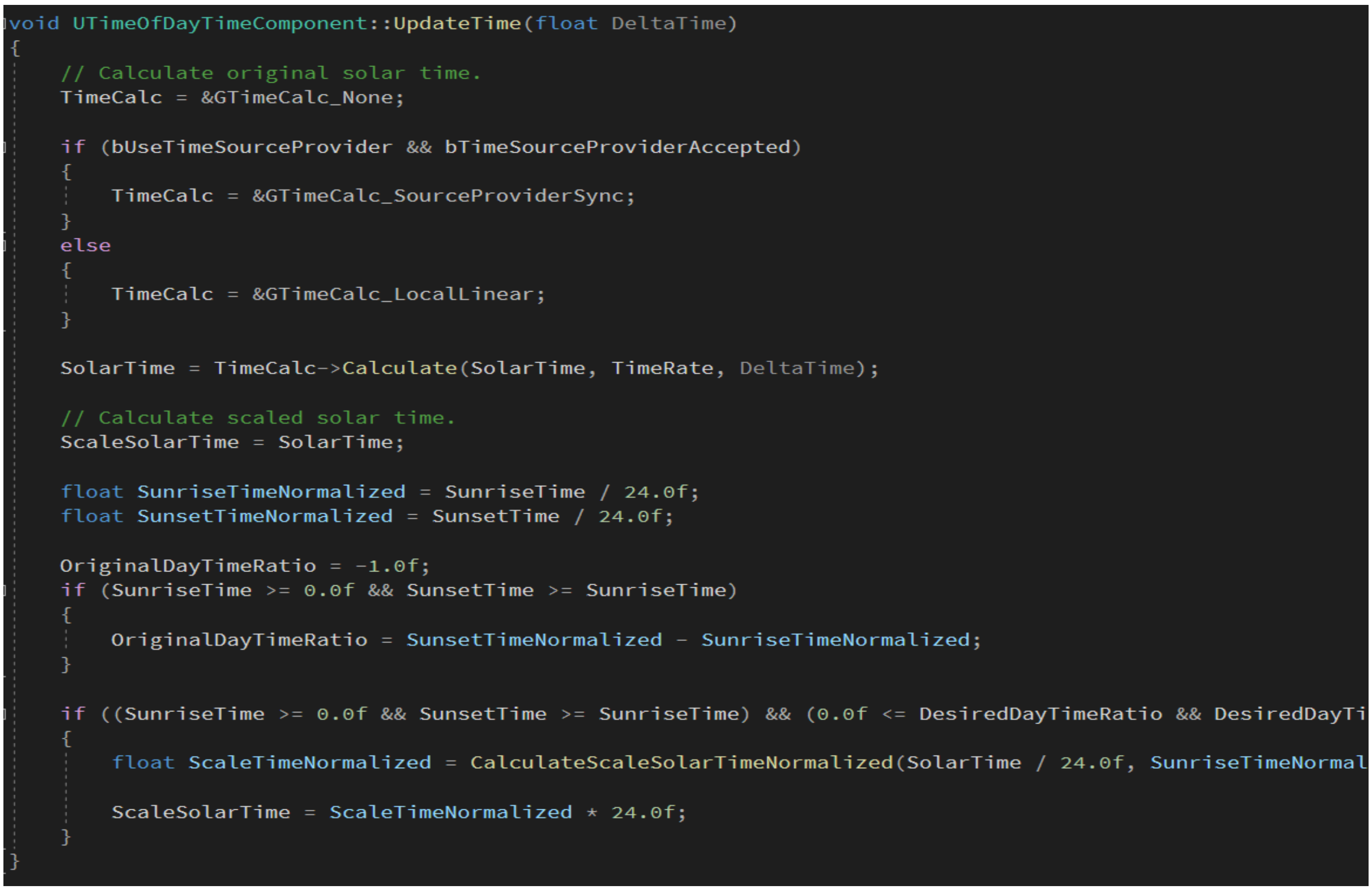
○ TimeOfDayTimeComponent 는 Initialize() 에서 해/달 위치 표현을 위한 컴파스 메시 설정, 물리 시간 존을 위한 설정, 하루가 바뀌었을 때 이벤트를 보내기 위한 콜백 설정 등을 수행하고, 매 프레임 UpdateProperties() 에서 시간을 계산하고 해와 달의 트랜스폼 정보를 갱신.



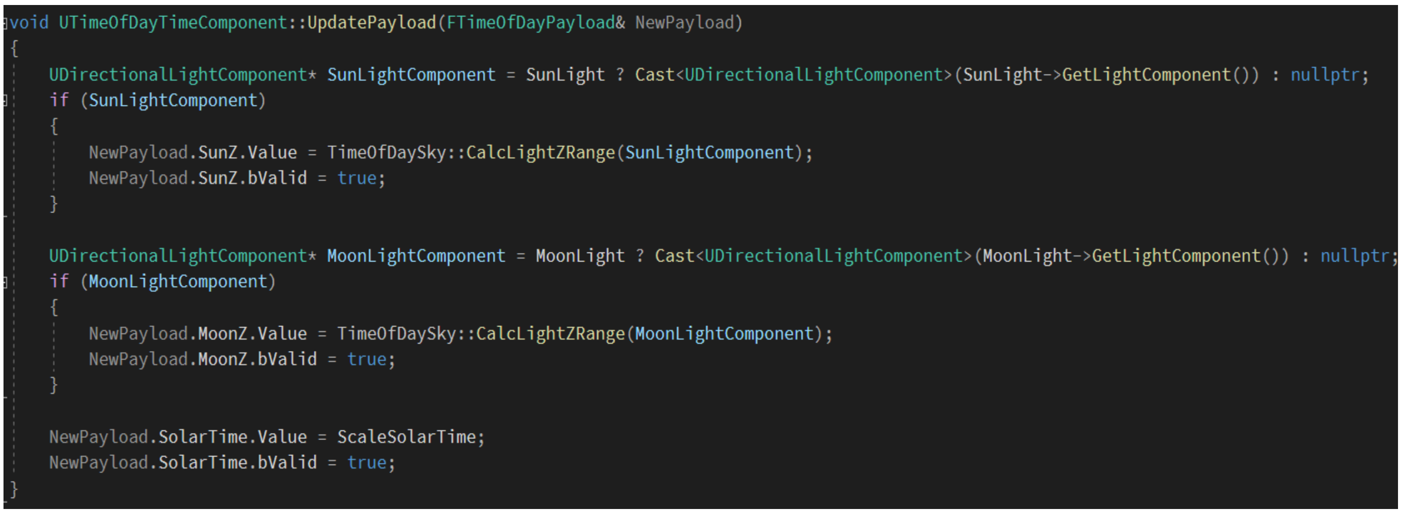
```
void UTimeOfDayTimeComponent::Initialize()
{
    if (IsRunningDedicatedServer())
    {
        SetupSourceProvider();

        SetupDayChangedEvent();
    }
    else
    {
        SetupCompassMesh();

        SetupTimeZone();

        SetupSourceProvider();

        SetupDayChangedEvent();
    }
}
```



```
void UTimeOfDayTimeComponent::UpdateProperties(float DeltaTime)
{
    if (IsRunningDedicatedServer())
    {
        UpdateTime(DeltaTime);
    }
    else
    {
        UpdateTime(DeltaTime);
        UpdateSunMoonTransformData();
    }
}
```



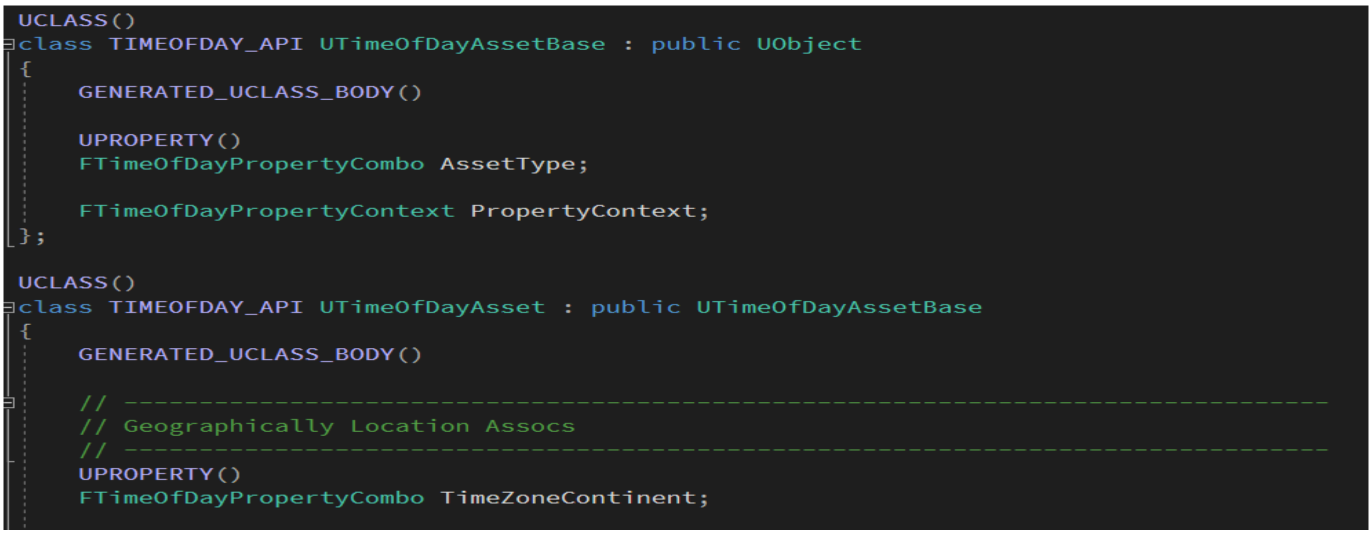
```cpp
void UTimeOfDayTimeComponent::UpdateTime(float DeltaTime)
{
    // Calculate original solar time.
    TimeCalc = &GTimeCalc_None;

    if (bUseTimeSourceProvider && bTimeSourceProviderAccepted)
    {
        TimeCalc = &GTimeCalc_SourceProviderSync;
    }
    else
    {
        TimeCalc = &GTimeCalc_LocalLinear;
    }

    SolarTime = TimeCalc->Calculate(SolarTime, TimeRate, DeltaTime);

    // Calculate scaled solar time.
    ScaleSolarTime = SolarTime;

    float SunriseTimeNormalized = SunriseTime / 24.0f;
    float SunsetTimeNormalized = SunsetTime / 24.0f;

    OriginalDayTimeRatio = -1.0f;
    if (SunriseTime >= 0.0f && SunsetTime >= SunriseTime)
    {
        OriginalDayTimeRatio = SunsetTimeNormalized - SunriseTimeNormalized;
    }

    if ((SunriseTime >= 0.0f && SunsetTime >= SunriseTime) && (0.0f <= DesiredDayTimeRatio && DesiredDayTi
    {
        float ScaleTimeNormalized = CalculateScaleSolarTimeNormalized(SolarTime / 24.0f, SunriseTimeNormal

        ScaleSolarTime = ScaleTimeNormalized * 24.0f;
    }
}
```



```cpp
void UTimeOfDayTimeComponent::UpdatePayload(FTimeOfDayPayload& NewPayload)
{
    UDirectionalLightComponent* SunLightComponent = SunLight ? Cast<UDirectionalLightComponent>(SunLight->GetLightComponent()) : nullptr;
    if (SunLightComponent)
    {
        NewPayload.SunZ.Value = TimeOfDaySky::CalcLightZRange(SunLightComponent);
        NewPayload.SunZ.bValid = true;
    }

    UDirectionalLightComponent* MoonLightComponent = MoonLight ? Cast<UDirectionalLightComponent>(MoonLight->GetLightComponent()) : nullptr;
    if (MoonLightComponent)
    {
        NewPayload.MoonZ.Value = TimeOfDaySky::CalcLightZRange(MoonLightComponent);
        NewPayload.MoonZ.bValid = true;
    }

    NewPayload.SolarTime.Value = ScaleSolarTime;
    NewPayload.SolarTime.bValid = true;
}
```

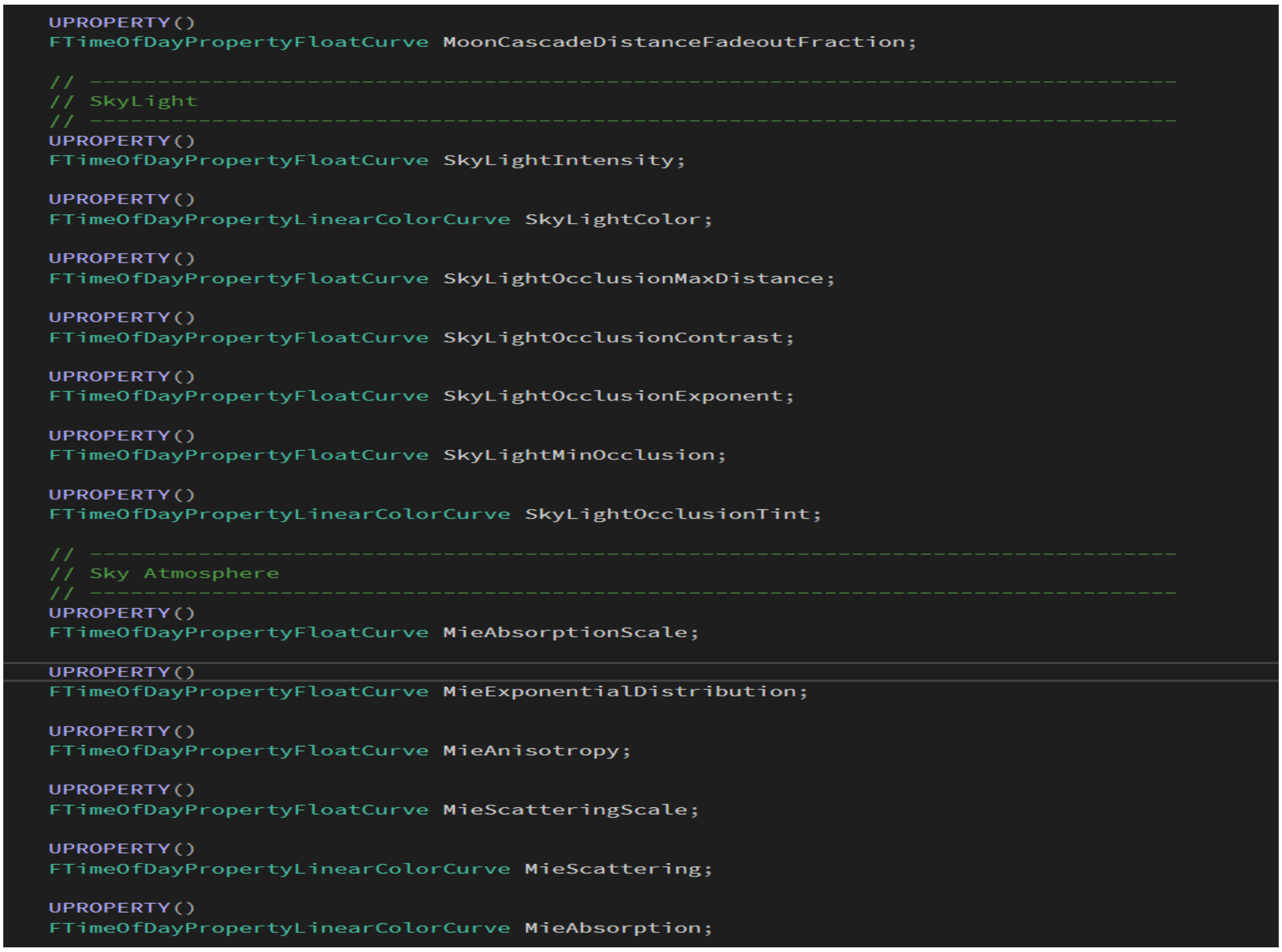
□ 라이팅 애셋

○ UTimeOfDayAsset : 하루의 시간 동안 변하는 값을 저장하기 위한 애셋.



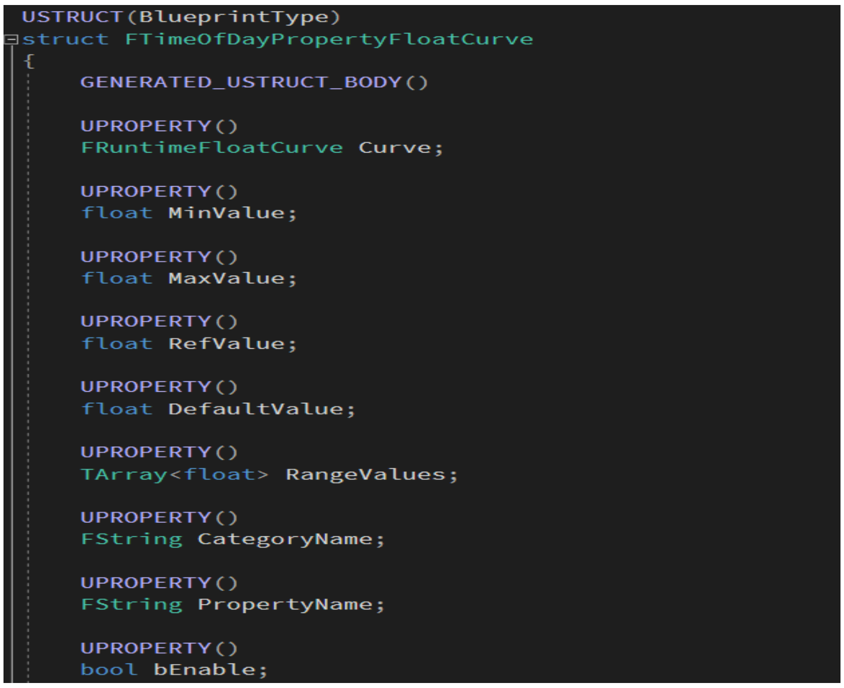
```
UCLASS()
class TIMEOFDAY_API UTimeOfDayAssetBase : public UObject
{
    GENERATED_UCLASS_BODY()

    UPROPERTY()
    FTimeOfDayPropertyCombo AssetType;

    FTimeOfDayPropertyContext PropertyContext;
};

UCLASS()
class TIMEOFDAY_API UTimeOfDayAsset : public UTimeOfDayAssetBase
{
    GENERATED_UCLASS_BODY()

    // -------------------------------------------------------------------------------
    // Geographically Location Assocs
    // -------------------------------------------------------------------------------
    UPROPERTY()
    FTimeOfDayPropertyCombo TimeZoneContinent;
```

○ 애셋 데이터.



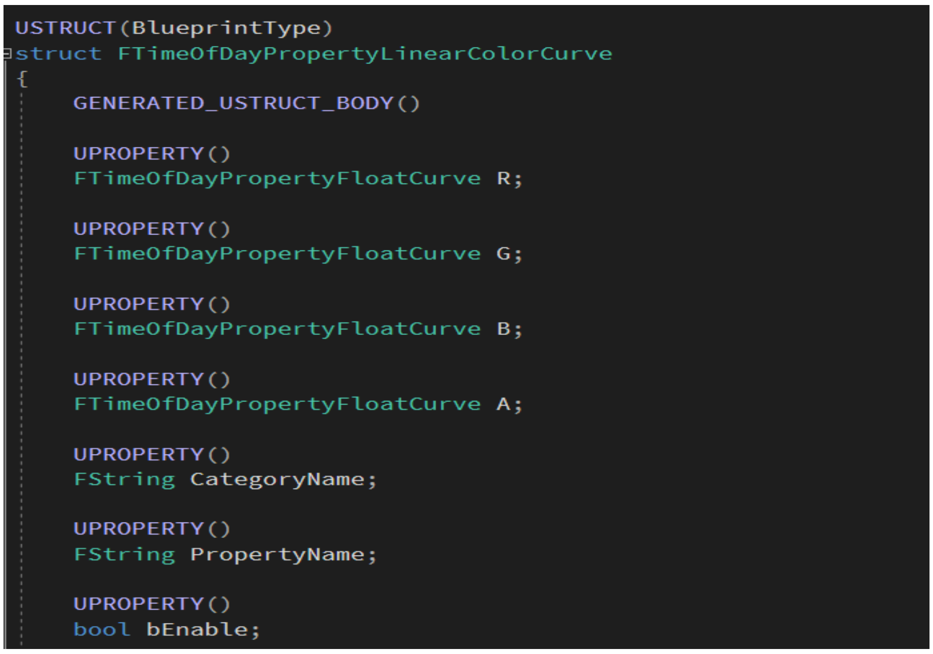
```
    UPROPERTY()
    FTimeOfDayPropertyFloatCurve MoonCascadeDistanceFadeoutFraction;

    // ------------------------------------------------------------------------------
    // SkyLight
    // ------------------------------------------------------------------------------
    UPROPERTY()
    FTimeOfDayPropertyFloatCurve SkyLightIntensity;

    UPROPERTY()
    FTimeOfDayPropertyLinearColorCurve SkyLightColor;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve SkyLightOcclusionMaxDistance;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve SkyLightOcclusionContrast;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve SkyLightOcclusionExponent;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve SkyLightMinOcclusion;

    UPROPERTY()
    FTimeOfDayPropertyLinearColorCurve SkyLightOcclusionTint;

    // ------------------------------------------------------------------------------
    // Sky Atmosphere
    // ------------------------------------------------------------------------------
    UPROPERTY()
    FTimeOfDayPropertyFloatCurve MieAbsorptionScale;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve MieExponentialDistribution;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve MieAnisotropy;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve MieScatteringScale;

    UPROPERTY()
    FTimeOfDayPropertyLinearColorCurve MieScattering;

    UPROPERTY()
    FTimeOfDayPropertyLinearColorCurve MieAbsorption;
```

○ FTimeOfDayPropertyFloatCurve : float



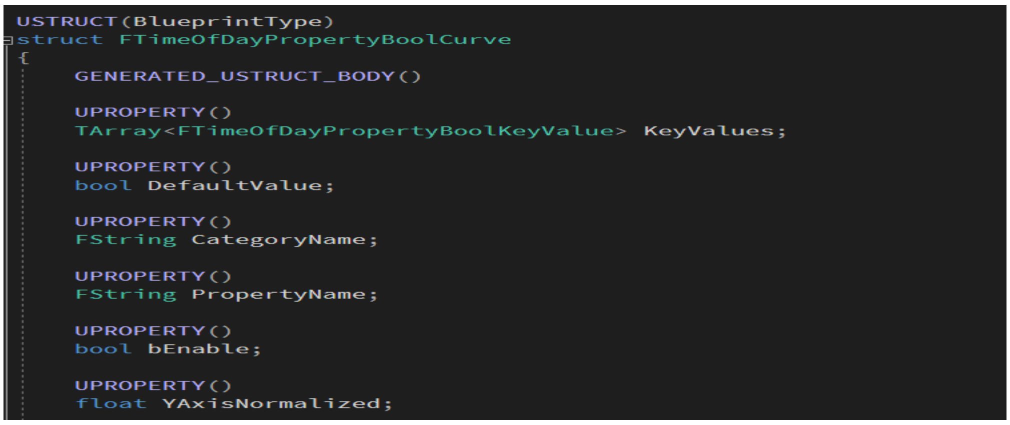
```
USTRUCT(BlueprintType)
struct FTimeOfDayPropertyFloatCurve
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    FRuntimeFloatCurve Curve;

    UPROPERTY()
    float MinValue;

    UPROPERTY()
    float MaxValue;

    UPROPERTY()
    float RefValue;

    UPROPERTY()
    float DefaultValue;

    UPROPERTY()
    TArray<float> RangeValues;

    UPROPERTY()
    FString CategoryName;

    UPROPERTY()
    FString PropertyName;

    UPROPERTY()
    bool bEnable;
```

○ FTimeOfDayPropertyLinearColorCurve : Color



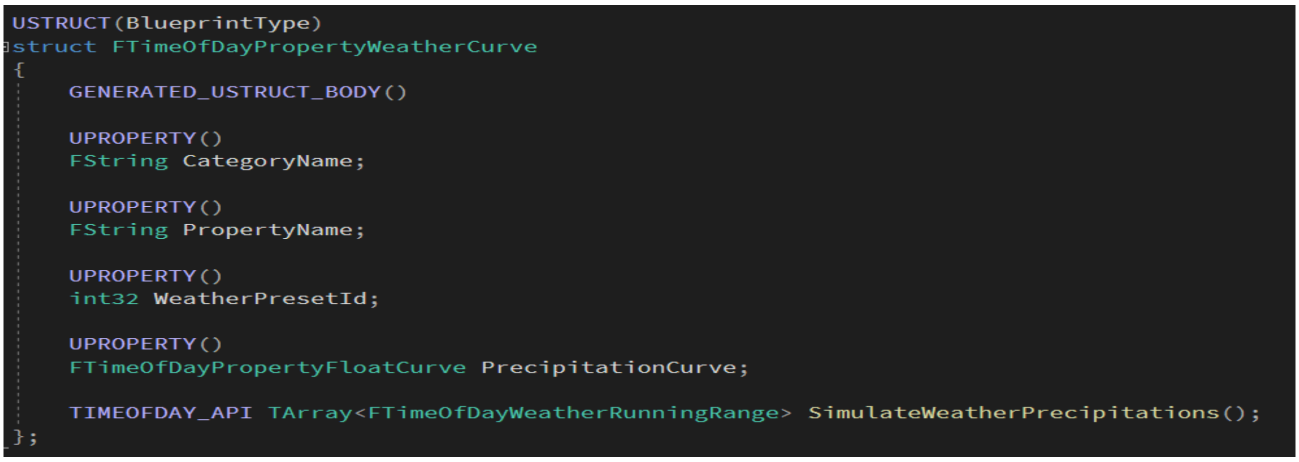
```
USTRUCT(BlueprintType)
struct FTimeOfDayPropertyLinearColorCurve
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve R;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve G;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve B;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve A;

    UPROPERTY()
    FString CategoryName;

    UPROPERTY()
    FString PropertyName;

    UPROPERTY()
    bool bEnable;
```

○ FTimeOfDayPropertyBoolCurve : 0, 1.



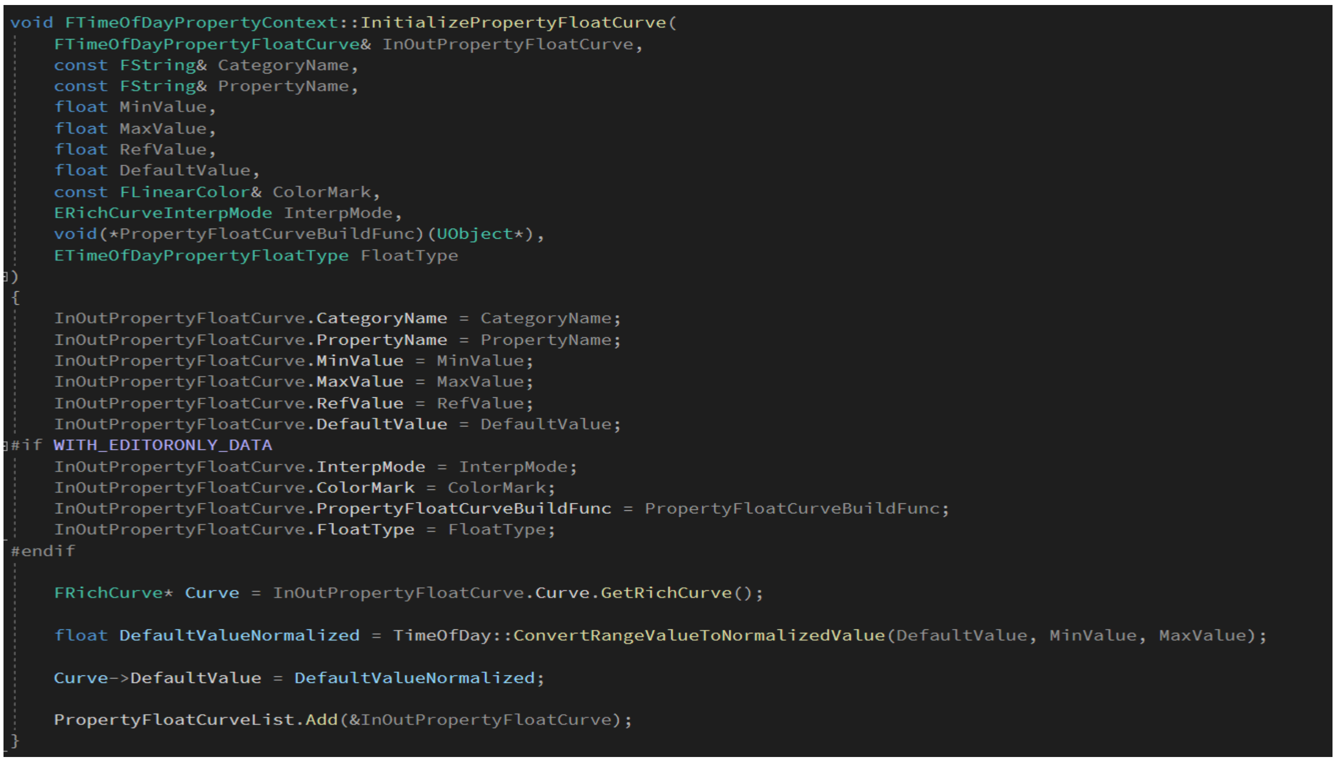
```
USTRUCT(BlueprintType)
struct FTimeOfDayPropertyBoolCurve
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    TArray<FTimeOfDayPropertyBoolKeyValue> KeyValues;

    UPROPERTY()
    bool DefaultValue;

    UPROPERTY()
    FString CategoryName;

    UPROPERTY()
    FString PropertyName;

    UPROPERTY()
    bool bEnable;

    UPROPERTY()
    float YAxisNormalized;
```

○ FTimeOfDayPropertyWeatherCurve : 격자 모양의 커브 사용(ERichCurveInterpMode::RCIM_Constant).



```
USTRUCT(BlueprintType)
struct FTimeOfDayPropertyWeatherCurve
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    FString CategoryName;

    UPROPERTY()
    FString PropertyName;

    UPROPERTY()
    int32 WeatherPresetId;

    UPROPERTY()
    FTimeOfDayPropertyFloatCurve PrecipitationCurve;

    TIMEOFDAY_API TArray<FTimeOfDayWeatherRunningRange> SimulateWeatherPrecipitations();
};
```

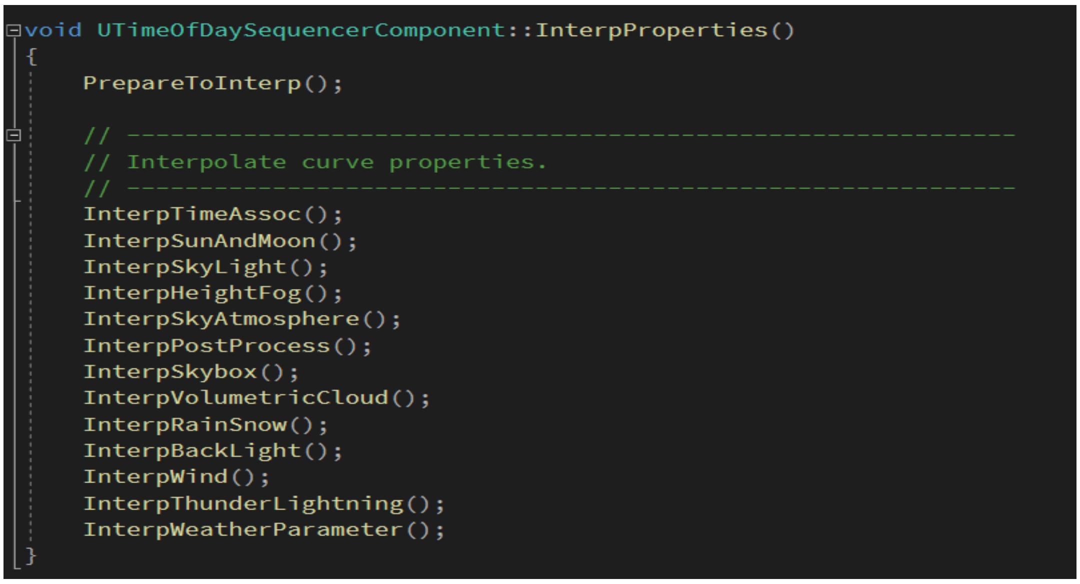
○ PropertyContext : FTimeOfDayPropertyXXXCurve 들을 초기화, 저장, 관리.



```
void FTimeOfDayPropertyContext::InitializePropertyFloatCurve(
    FTimeOfDayPropertyFloatCurve& InOutPropertyFloatCurve,
    const FString& CategoryName,
    const FString& PropertyName,
    float MinValue,
    float MaxValue,
    float RefValue,
    float DefaultValue,
    const FLinearColor& ColorMark,
    ERichCurveInterpMode InterpMode,
    void(*PropertyFloatCurveBuildFunc)(UObject*),
    ETimeOfDayPropertyFloatType FloatType
)
{
    InOutPropertyFloatCurve.CategoryName = CategoryName;
    InOutPropertyFloatCurve.PropertyName = PropertyName;
    InOutPropertyFloatCurve.MinValue = MinValue;
    InOutPropertyFloatCurve.MaxValue = MaxValue;
    InOutPropertyFloatCurve.RefValue = RefValue;
    InOutPropertyFloatCurve.DefaultValue = DefaultValue;
#if WITH_EDITORONLY_DATA
    InOutPropertyFloatCurve.InterpMode = InterpMode;
    InOutPropertyFloatCurve.ColorMark = ColorMark;
    InOutPropertyFloatCurve.PropertyFloatCurveBuildFunc = PropertyFloatCurveBuildFunc;
    InOutPropertyFloatCurve.FloatType = FloatType;
#endif

    FRichCurve* Curve = InOutPropertyFloatCurve.Curve.GetRichCurve();

    float DefaultValueNormalized = TimeOfDay::ConvertRangeValueToNormalizedValue(DefaultValue, MinValue, MaxValue);

    Curve->DefaultValue = DefaultValueNormalized;

    PropertyFloatCurveList.Add(&InOutPropertyFloatCurve);
}
```

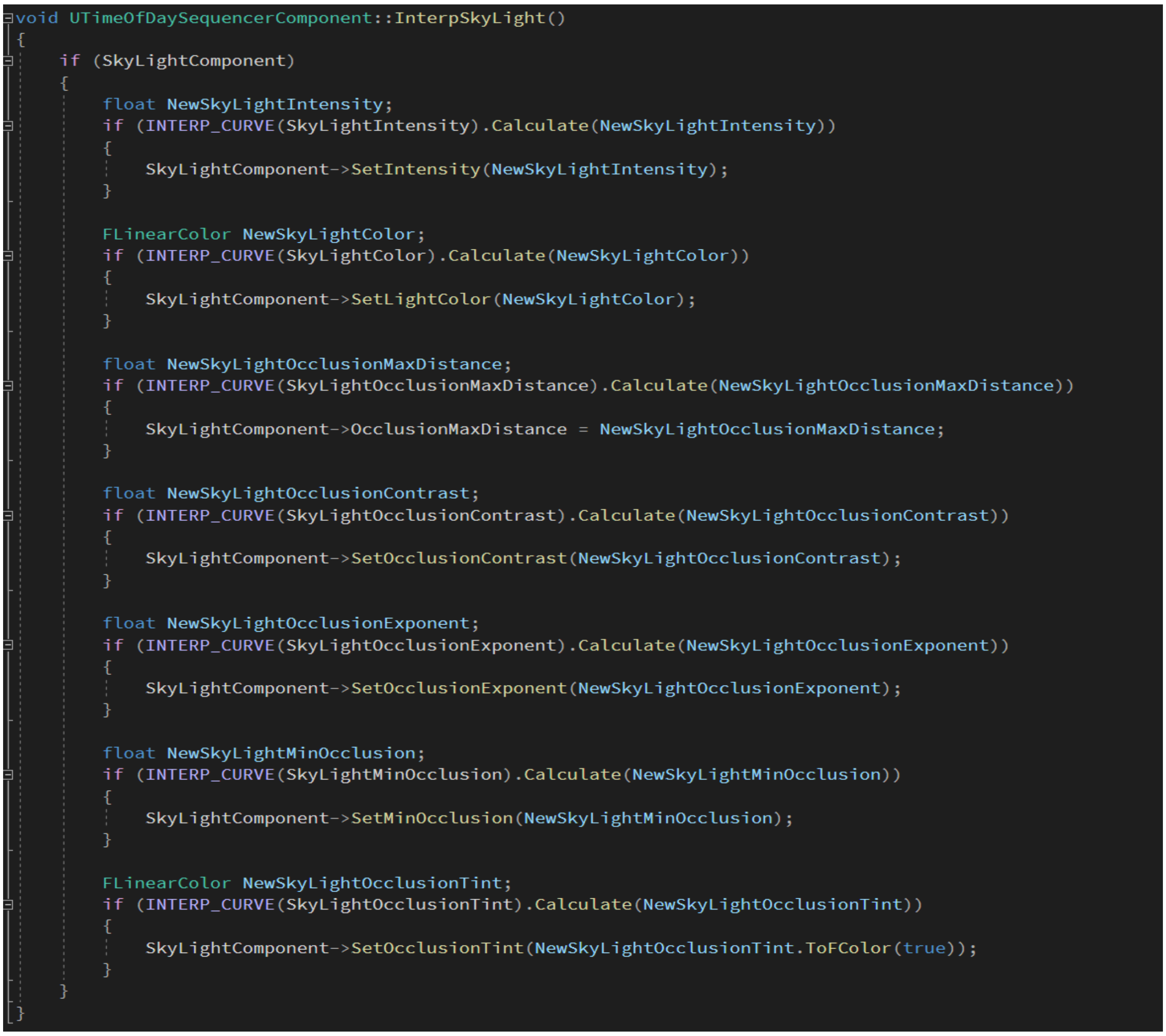
□ 애셋 커브 값 적용

○ 애셋의 커브 데이터 적용.



```
void UTimeOfDaySequencerComponent::InterpProperties()
{
    PrepareToInterp();

    // -------------------------------------------------------------
    // Interpolate curve properties.
    // -------------------------------------------------------------
    InterpTimeAssoc();
    InterpSunAndMoon();
    InterpSkyLight();
    InterpHeightFog();
    InterpSkyAtmosphere();
    InterpPostProcess();
    InterpSkybox();
    InterpVolumetricCloud();
    InterpRainSnow();
    InterpBackLight();
    InterpWind();
    InterpThunderLightning();
    InterpWeatherParameter();
}
```

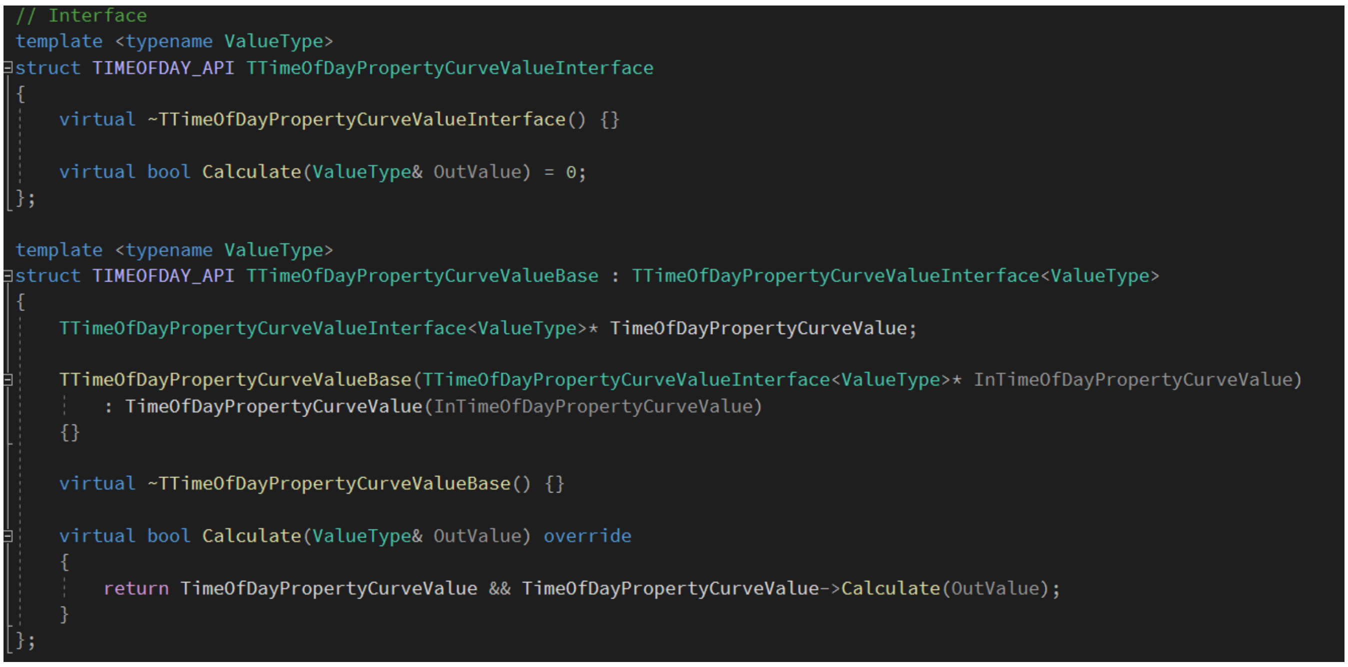
○ 스카이라이트의 예.



```
void UTimeOfDaySequencerComponent::InterpSkyLight()
{
    if (SkyLightComponent)
    {
        float NewSkyLightIntensity;
        if (INTERP_CURVE(SkyLightIntensity).Calculate(NewSkyLightIntensity))
        {
            SkyLightComponent->SetIntensity(NewSkyLightIntensity);
        }

        FLinearColor NewSkyLightColor;
        if (INTERP_CURVE(SkyLightColor).Calculate(NewSkyLightColor))
        {
            SkyLightComponent->SetLightColor(NewSkyLightColor);
        }

        float NewSkyLightOcclusionMaxDistance;
        if (INTERP_CURVE(SkyLightOcclusionMaxDistance).Calculate(NewSkyLightOcclusionMaxDistance))
        {
            SkyLightComponent->OcclusionMaxDistance = NewSkyLightOcclusionMaxDistance;
        }

        float NewSkyLightOcclusionContrast;
        if (INTERP_CURVE(SkyLightOcclusionContrast).Calculate(NewSkyLightOcclusionContrast))
        {
            SkyLightComponent->SetOcclusionContrast(NewSkyLightOcclusionContrast);
        }

        float NewSkyLightOcclusionExponent;
        if (INTERP_CURVE(SkyLightOcclusionExponent).Calculate(NewSkyLightOcclusionExponent))
        {
            SkyLightComponent->SetOcclusionExponent(NewSkyLightOcclusionExponent);
        }

        float NewSkyLightMinOcclusion;
        if (INTERP_CURVE(SkyLightMinOcclusion).Calculate(NewSkyLightMinOcclusion))
        {
            SkyLightComponent->SetMinOcclusion(NewSkyLightMinOcclusion);
        }

        FLinearColor NewSkyLightOcclusionTint;
        if (INTERP_CURVE(SkyLightOcclusionTint).Calculate(NewSkyLightOcclusionTint))
        {
            SkyLightComponent->SetOcclusionTint(NewSkyLightOcclusionTint.ToFColor(true));
        }
    }
}
```

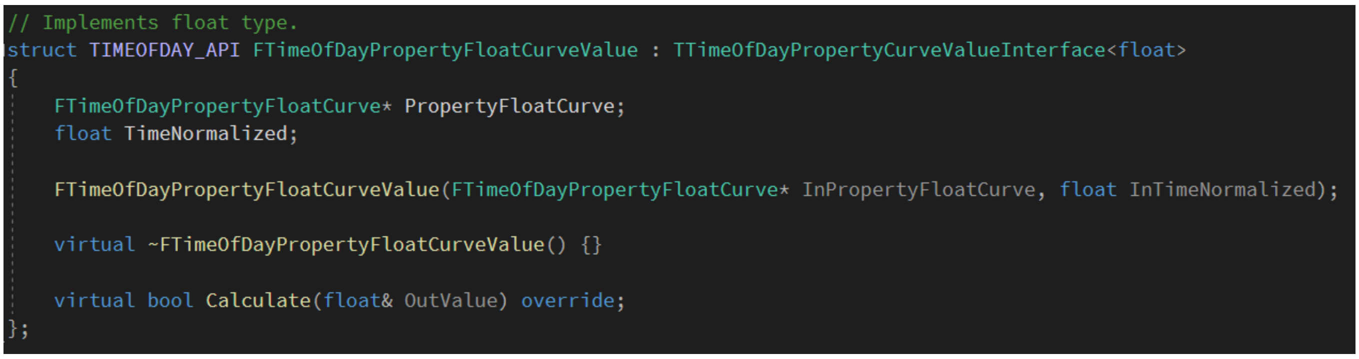
☐ INTERP_CURVE 매크로

○ INTERP_CURVE 매크로는 다양한 보간 방법을 처리하기 위해, Wrapper 인터페이스 객체를 사용.

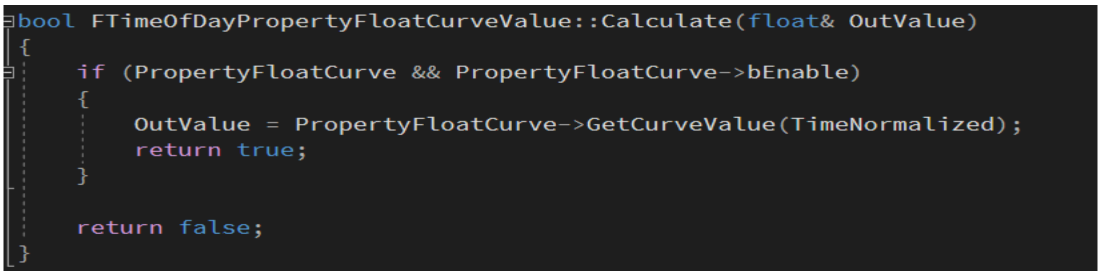


```cpp
// Interface
template <typename ValueType>
struct TIMEOFDAY_API TTimeOfDayPropertyCurveValueInterface
{
    virtual ~TTimeOfDayPropertyCurveValueInterface() {}

    virtual bool Calculate(ValueType& OutValue) = 0;
};

template <typename ValueType>
struct TIMEOFDAY_API TTimeOfDayPropertyCurveValueBase : TTimeOfDayPropertyCurveValueInterface<ValueType>
{
    TTimeOfDayPropertyCurveValueInterface<ValueType>* TimeOfDayPropertyCurveValue;

    TTimeOfDayPropertyCurveValueBase(TTimeOfDayPropertyCurveValueInterface<ValueType>* InTimeOfDayPropertyCurveValue)
        : TimeOfDayPropertyCurveValue(InTimeOfDayPropertyCurveValue)
    {}

    virtual ~TTimeOfDayPropertyCurveValueBase() {}

    virtual bool Calculate(ValueType& OutValue) override
    {
        return TimeOfDayPropertyCurveValue && TimeOfDayPropertyCurveValue->Calculate(OutValue);
    }
};
```

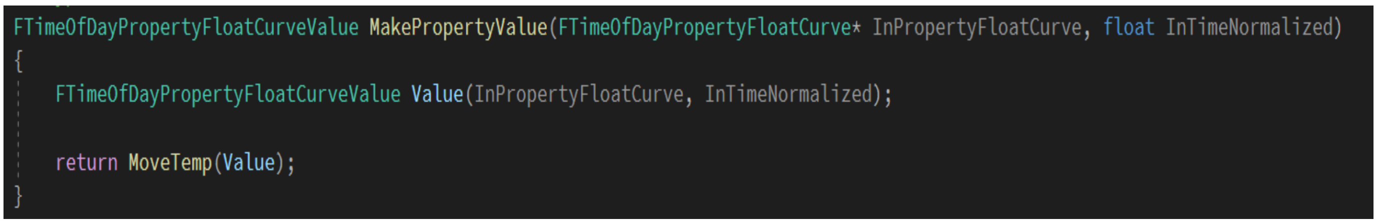
○ 단일 Value 객체. 특정 시간대의 값을 커브 데이터에서 가져옴.



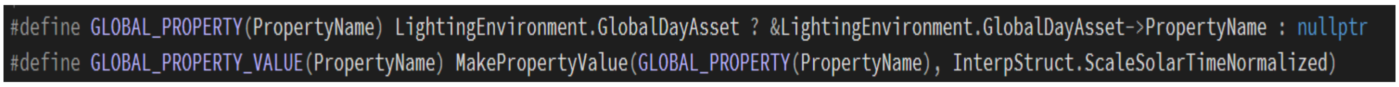
```cpp
// Implements float type.
struct TIMEOFDAY_API FTimeOfDayPropertyFloatCurveValue : TTimeOfDayPropertyCurveValueInterface<float>
{
    FTimeOfDayPropertyFloatCurve* PropertyFloatCurve;
    float TimeNormalized;

    FTimeOfDayPropertyFloatCurveValue(FTimeOfDayPropertyFloatCurve* InPropertyFloatCurve, float InTimeNormalized);

    virtual ~FTimeOfDayPropertyFloatCurveValue() {}

    virtual bool Calculate(float& OutValue) override;
};
```



```cpp
bool FTimeOfDayPropertyFloatCurveValue::Calculate(float& OutValue)
{
    if (PropertyFloatCurve && PropertyFloatCurve->bEnable)
    {
        OutValue = PropertyFloatCurve->GetCurveValue(TimeNormalized);
        return true;
    }

    return false;
}
```

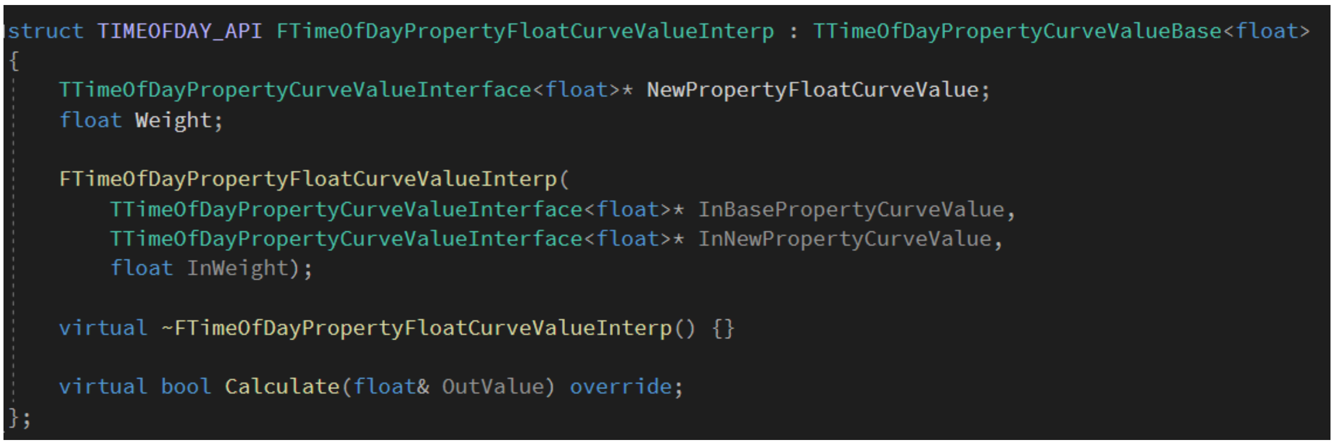


```cpp
FTimeOfDayPropertyFloatCurveValue MakePropertyValue(FTimeOfDayPropertyFloatCurve* InPropertyFloatCurve, float InTimeNormalized)
{
    FTimeOfDayPropertyFloatCurveValue Value(InPropertyFloatCurve, InTimeNormalized);

    return MoveTemp(Value);
}
```

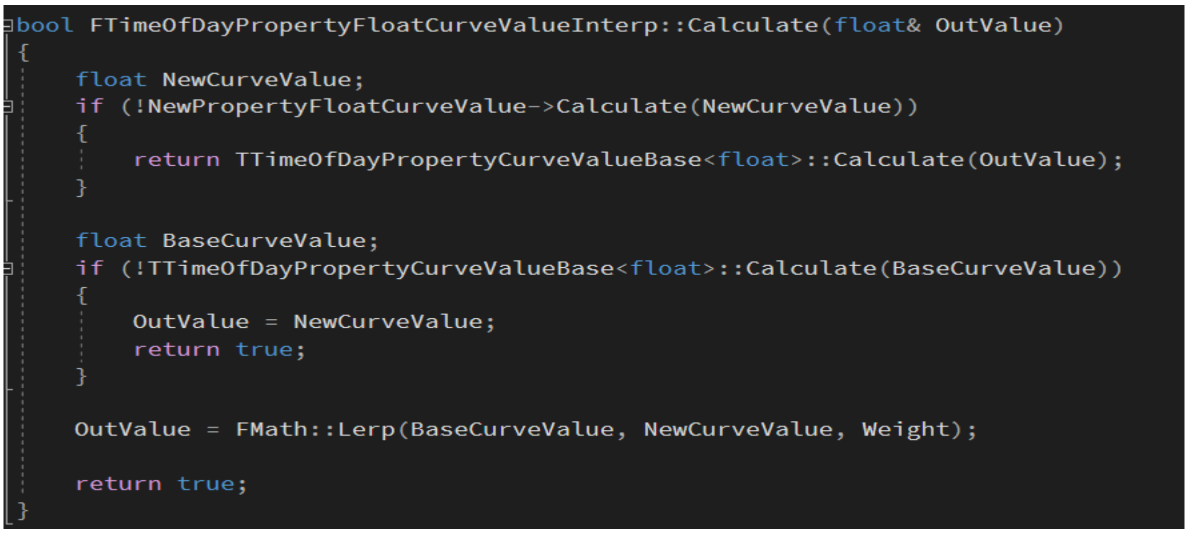


```cpp
#define GLOBAL_PROPERTY(PropertyName) LightingEnvironment.GlobalDayAsset ? &LightingEnvironment.GlobalDayAsset->PropertyName : nullptr
#define GLOBAL_PROPERTY_VALUE(PropertyName) MakePropertyValue(GLOBAL_PROPERTY(PropertyName), InterpStruct.ScaleSolarTimeNormalized)
```

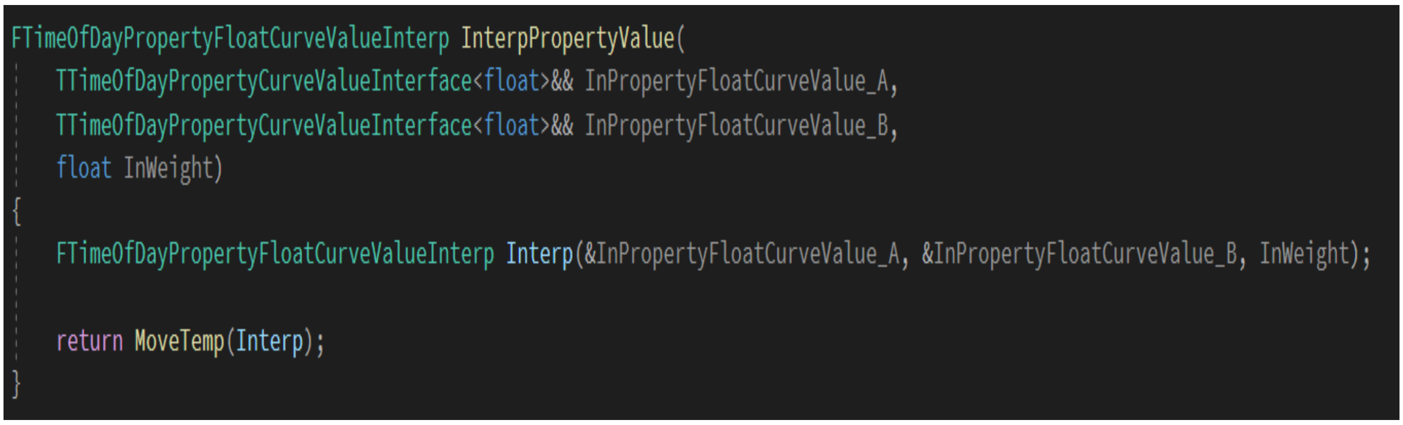
○ 두 Value 객체의 값을 단순 선형 보간



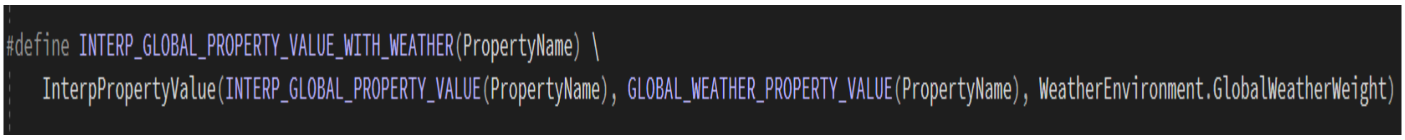
```
struct TIMEOFDAY_API FTimeOfDayPropertyFloatCurveValueInterp : TTimeOfDayPropertyCurveValueBase<float>
{
    TTimeOfDayPropertyCurveValueInterface<float>* NewPropertyFloatCurveValue;
    float Weight;

    FTimeOfDayPropertyFloatCurveValueInterp(
        TTimeOfDayPropertyCurveValueInterface<float>* InBasePropertyCurveValue,
        TTimeOfDayPropertyCurveValueInterface<float>* InNewPropertyCurveValue,
        float InWeight);

    virtual ~FTimeOfDayPropertyFloatCurveValueInterp() {}

    virtual bool Calculate(float& OutValue) override;
};
```



```
bool FTimeOfDayPropertyFloatCurveValueInterp::Calculate(float& OutValue)
{
    float NewCurveValue;
    if (!NewPropertyFloatCurveValue->Calculate(NewCurveValue))
    {
        return TTimeOfDayPropertyCurveValueBase<float>::Calculate(OutValue);
    }

    float BaseCurveValue;
    if (!TTimeOfDayPropertyCurveValueBase<float>::Calculate(BaseCurveValue))
    {
        OutValue = NewCurveValue;
        return true;
    }

    OutValue = FMath::Lerp(BaseCurveValue, NewCurveValue, Weight);

    return true;
}
```

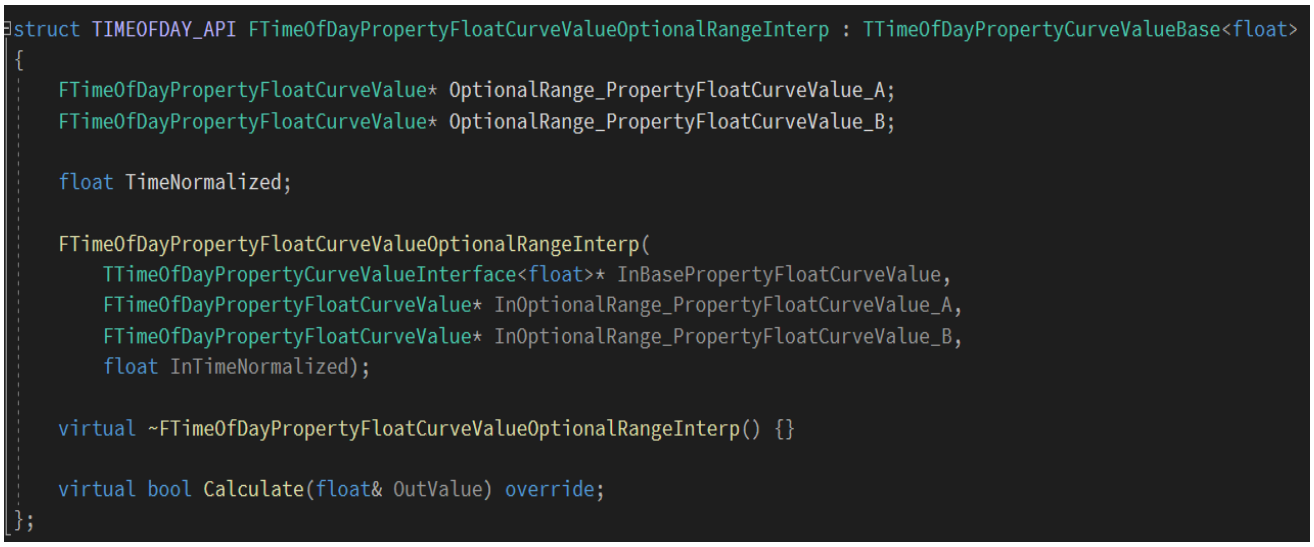


```
FTimeOfDayPropertyFloatCurveValueInterp InterpPropertyValue(
    TTimeOfDayPropertyCurveValueInterface<float>&& InPropertyFloatCurveValue_A,
    TTimeOfDayPropertyCurveValueInterface<float>&& InPropertyFloatCurveValue_B,
    float InWeight)
{
    FTimeOfDayPropertyFloatCurveValueInterp Interp(&InPropertyFloatCurveValue_A, &InPropertyFloatCurveValue_B, InWeight);

    return MoveTemp(Interp);
}
```

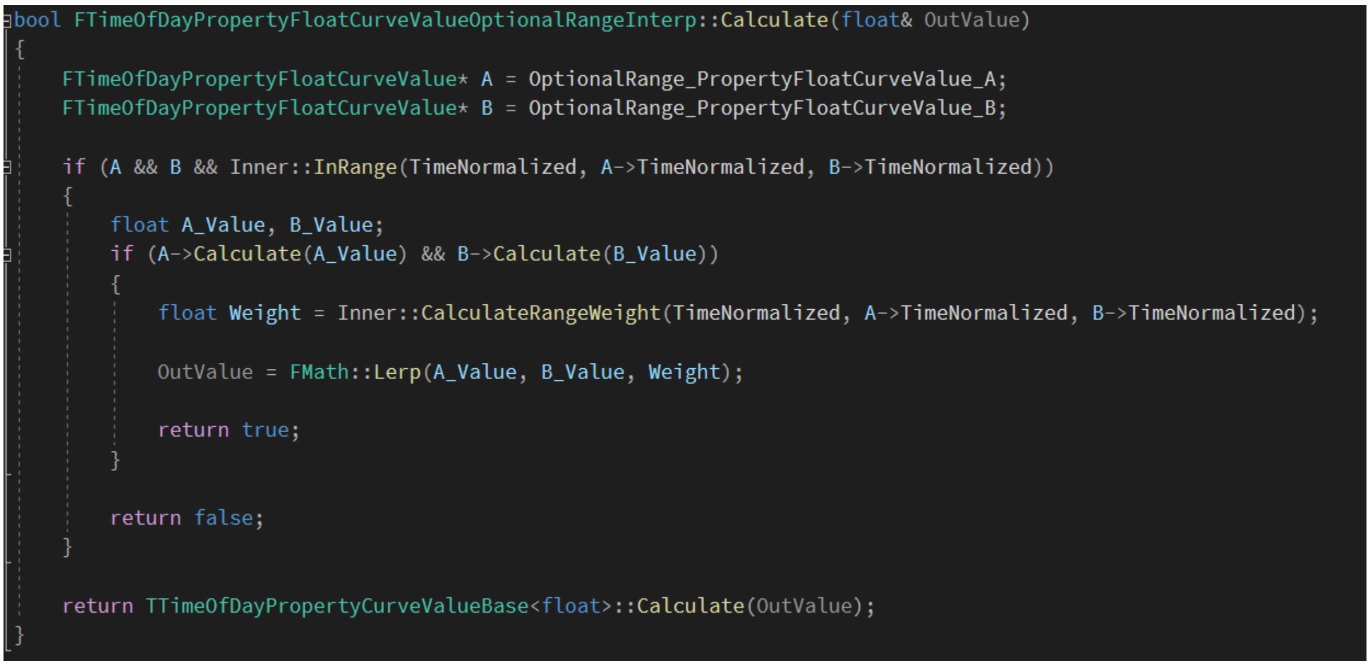


```
#define INTERP_GLOBAL_PROPERTY_VALUE_WITH_WEATHER(PropertyName) \
    InterpPropertyValue(INTERP_GLOBAL_PROPERTY_VALUE(PropertyName), GLOBAL_WEATHER_PROPERTY_VALUE(PropertyName), WeatherEnvironment.GlobalWeatherWeight)
```

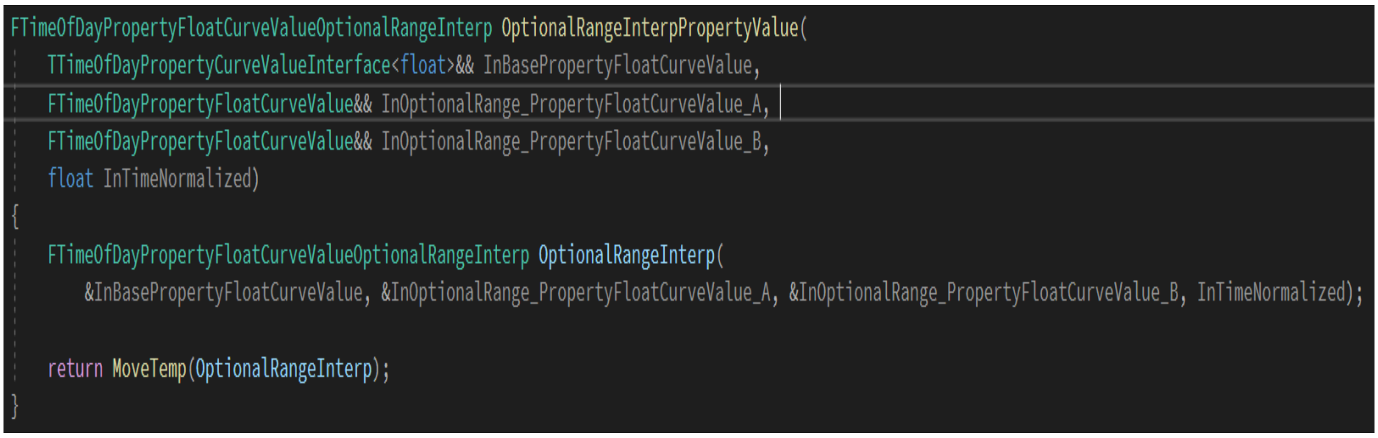
○ 특정 시간대만 단순 선형 보간.



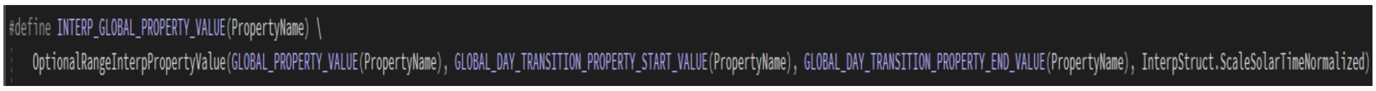
```cpp
struct TIMEOFDAY_API FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp : TTimeOfDayPropertyCurveValueBase<float>
{
    FTimeOfDayPropertyFloatCurveValue* OptionalRange_PropertyFloatCurveValue_A;
    FTimeOfDayPropertyFloatCurveValue* OptionalRange_PropertyFloatCurveValue_B;

    float TimeNormalized;

    FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp(
        TTimeOfDayPropertyCurveValueInterface<float>* InBasePropertyFloatCurveValue,
        FTimeOfDayPropertyFloatCurveValue* InOptionalRange_PropertyFloatCurveValue_A,
        FTimeOfDayPropertyFloatCurveValue* InOptionalRange_PropertyFloatCurveValue_B,
        float InTimeNormalized);

    virtual ~FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp() {}

    virtual bool Calculate(float& OutValue) override;
};
```



```cpp
bool FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp::Calculate(float& OutValue)
{
    FTimeOfDayPropertyFloatCurveValue* A = OptionalRange_PropertyFloatCurveValue_A;
    FTimeOfDayPropertyFloatCurveValue* B = OptionalRange_PropertyFloatCurveValue_B;

    if (A && B && Inner::InRange(TimeNormalized, A->TimeNormalized, B->TimeNormalized))
    {
        float A_Value, B_Value;
        if (A->Calculate(A_Value) && B->Calculate(B_Value))
        {
            float Weight = Inner::CalculateRangeWeight(TimeNormalized, A->TimeNormalized, B->TimeNormalized);

            OutValue = FMath::Lerp(A_Value, B_Value, Weight);

            return true;
        }

        return false;
    }

    return TTimeOfDayPropertyCurveValueBase<float>::Calculate(OutValue);
}
```



```cpp
FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp OptionalRangeInterpPropertyValue(
    TTimeOfDayPropertyCurveValueInterface<float>&& InBasePropertyFloatCurveValue,
    FTimeOfDayPropertyFloatCurveValue&& InOptionalRange_PropertyFloatCurveValue_A,
    FTimeOfDayPropertyFloatCurveValue&& InOptionalRange_PropertyFloatCurveValue_B,
    float InTimeNormalized)
{
    FTimeOfDayPropertyFloatCurveValueOptionalRangeInterp OptionalRangeInterp(
        &InBasePropertyFloatCurveValue, &InOptionalRange_PropertyFloatCurveValue_A, &InOptionalRange_PropertyFloatCurveValue_B, InTimeNormalized);

    return MoveTemp(OptionalRangeInterp);
}
```



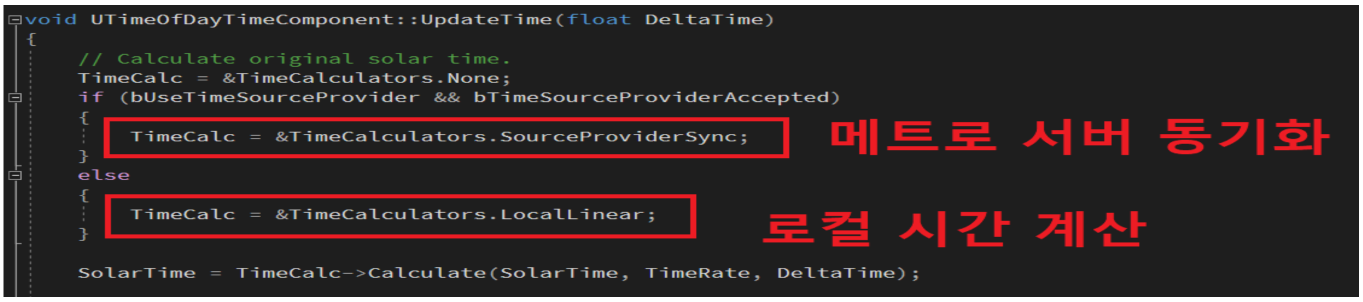
```cpp
#define INTERP_GLOBAL_PROPERTY_VALUE(PropertyName) \
    OptionalRangeInterpPropertyValue(GLOBAL_PROPERTY_VALUE(PropertyName), GLOBAL_DAY_TRANSITION_PROPERTY_START_VALUE(PropertyName), GLOBAL_DAY_TRANSITION_PROPERTY_END_VALUE(PropertyName), InterpStruct.ScaleSolarTimeNormalized)
```

○ 기타 보간 : LinearColor, bool 타입에 대한 구현도 위와 같이 처리되어 있음.

□ Time 업데이트

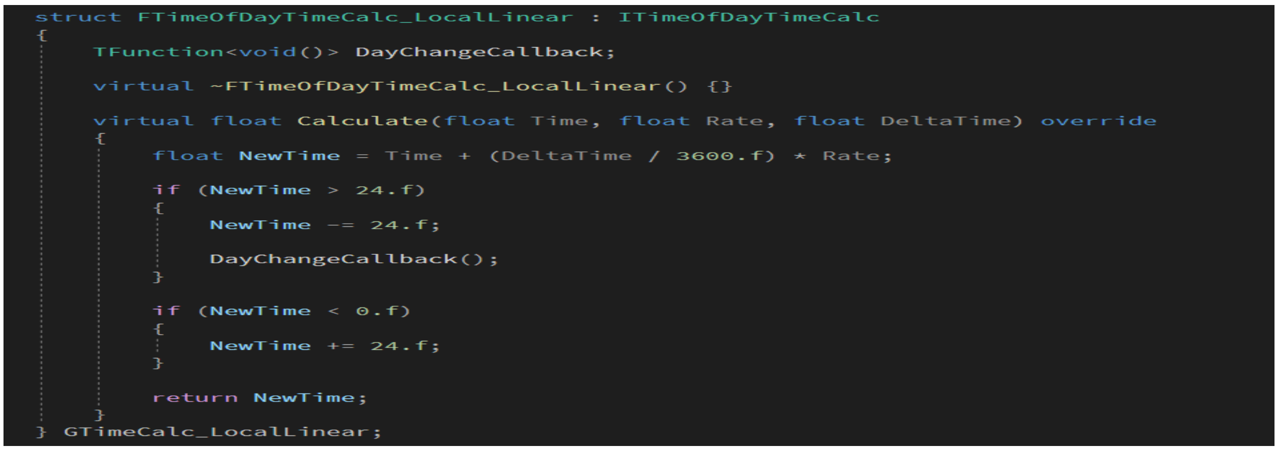
○ 에디터 : 로컬 시간 자체 계산.

○ 패키징 게임 : 메트로 서버 시간 동기화.



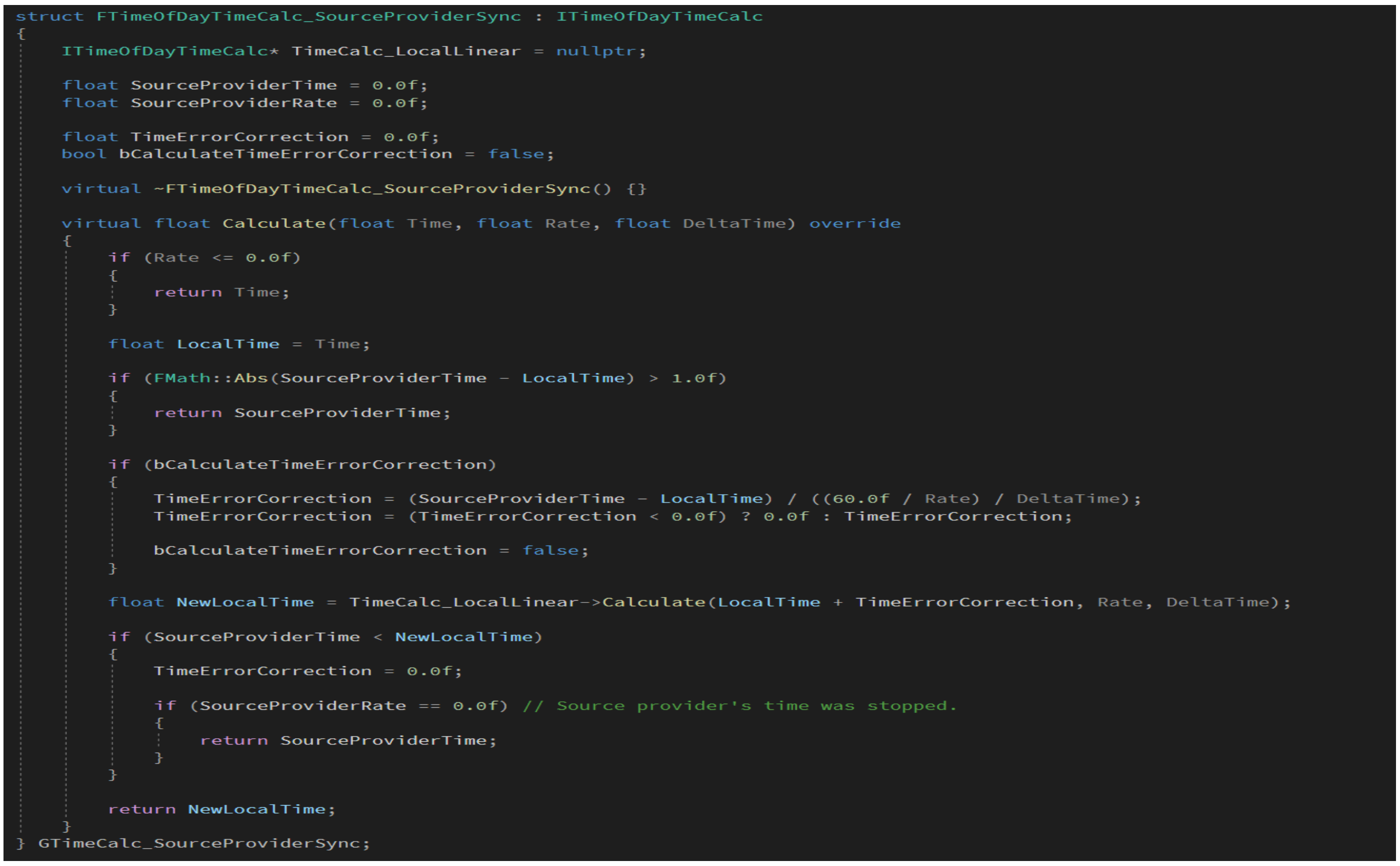
```cpp
void UTimeOfDayTimeComponent::UpdateTime(float DeltaTime)
{
    // Calculate original solar time.
    TimeCalc = &TimeCalculators.None;
    if (bUseTimeSourceProvider && bTimeSourceProviderAccepted)
    {
        TimeCalc = &TimeCalculators.SourceProviderSync;
    }
    else
    {
        TimeCalc = &TimeCalculators.LocalLinear;
    }

    SolarTime = TimeCalc->Calculate(SolarTime, TimeRate, DeltaTime);
```

○ 로컬 시간 계산 : 단순 Linear 업데이트



```cpp
struct FTimeOfDayTimeCalc_LocalLinear : ITimeOfDayTimeCalc
{
    TFunction<void()> DayChangeCallback;

    virtual ~FTimeOfDayTimeCalc_LocalLinear() {}

    virtual float Calculate(float Time, float Rate, float DeltaTime) override
    {
        float NewTime = Time + (DeltaTime / 3600.f) * Rate;

        if (NewTime > 24.f)
        {
            NewTime -= 24.f;

            DayChangeCallback();
        }

        if (NewTime < 0.f)
        {
            NewTime += 24.f;
        }

        return NewTime;
    }
} GTimeCalc_LocalLinear;
```

○ 메트로 서버 동기화 : Source Provider 를 경유하는 서버 시간으로 동기화



```cpp
struct FTimeOfDayTimeCalc_SourceProviderSync : ITimeOfDayTimeCalc
{
    ITimeOfDayTimeCalc* TimeCalc_LocalLinear = nullptr;

    float SourceProviderTime = 0.0f;
    float SourceProviderRate = 0.0f;

    float TimeErrorCorrection = 0.0f;
    bool bCalculateTimeErrorCorrection = false;

    virtual ~FTimeOfDayTimeCalc_SourceProviderSync() {}

    virtual float Calculate(float Time, float Rate, float DeltaTime) override
    {
        if (Rate <= 0.0f)
        {
            return Time;
        }

        float LocalTime = Time;

        if (FMath::Abs(SourceProviderTime - LocalTime) > 1.0f)
        {
            return SourceProviderTime;
        }

        if (bCalculateTimeErrorCorrection)
        {
            TimeErrorCorrection = (SourceProviderTime - LocalTime) / ((60.0f / Rate) / DeltaTime);
            TimeErrorCorrection = (TimeErrorCorrection < 0.0f) ? 0.0f : TimeErrorCorrection;

            bCalculateTimeErrorCorrection = false;
        }

        float NewLocalTime = TimeCalc_LocalLinear->Calculate(LocalTime + TimeErrorCorrection, Rate, DeltaTime);

        if (SourceProviderTime < NewLocalTime)
        {
            TimeErrorCorrection = 0.0f;

            if (SourceProviderRate == 0.0f) // Source provider's time was stopped.
            {
                return SourceProviderTime;
            }
        }

        return NewLocalTime;
    }
} GTimeCalc_SourceProviderSync;
```