# English

# 📘 J Asset Loader - User Guide

🎞️ **Preview Video**

👉 **Store Link**

💬 **Reddit**

---

## 🧩 Overview

J Asset Loader is a World Subsystem that recursively collects and asynchronously loads all Soft Object References inside a UObject or UStruct. It helps prevent hitches caused by asset loading and safely manages resources per Requester.

This system is designed for games that rely on complex data asset structures—such as characters and items—where nested resources like meshes and materials are stored as soft references.

---

## 🧩 Key Features

- Recursively collects and loads all Soft Object References inside a UObject or UStruct
- Fully Blueprint-exposed workflow
- Keeps loaded resources referenced to prevent GC
- Caching and deduplication system based on Requester and Request Key
- Built as a World Subsystem, not a GameInstance Subsystem

---

## 🧪 When Should You Use It?

In game development, it's often necessary to load a set of assets referenced via Soft References all at once.

For example, imagine a DataTable defining character data. Each row might include stats, model info, and default equipment. These are usually stored in DataAssets such as CharacterModelDataAsset or WeaponDataAsset, which in turn reference meshes, materials, etc.

If all references were hard (e.g., UStaticMesh*, TObjectPtr<>), the moment the DataTable is loaded, every character's resource would be loaded into memory.

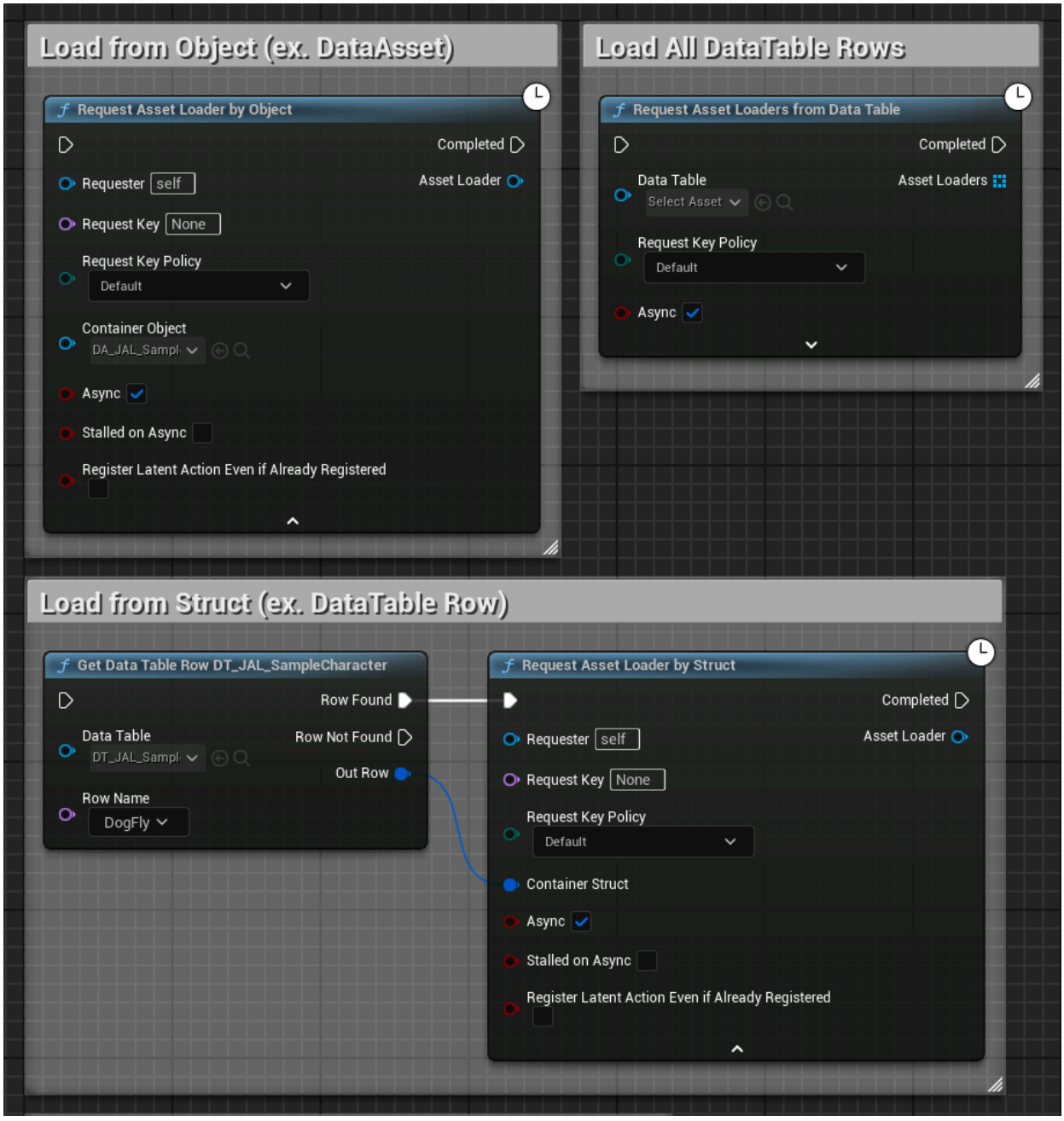
To prevent this, developers often use Soft References. However, loading nested soft references efficiently—especially asynchronously—can be challenging.

J Asset Loader solves this by scanning through all Soft References recursively in a given UObject or UStruct and issuing one async load request for all of them.

---

### 🎬 How It Works

Asset load requests begin with the **Request Asset Loader by Object** or **Request Asset Loader by Struct** Blueprint nodes.




If the request succeeds, the loader is wrapped in a `JAL Asset Loader` and registered with the `JAL Asset Loader Subsystem`.

This ensures that even if the loaded assets aren't referenced elsewhere, they won't be unloaded, as the subsystem manages them.

※ You can configure the Blueprint node to skip registration in the subsystem.

In that case, you must manually manage the returned Asset Loader or keep external references to the loaded assets to prevent them from being unloaded.

When a `Requester` is destroyed, the associated `Asset Loaders` are automatically released.

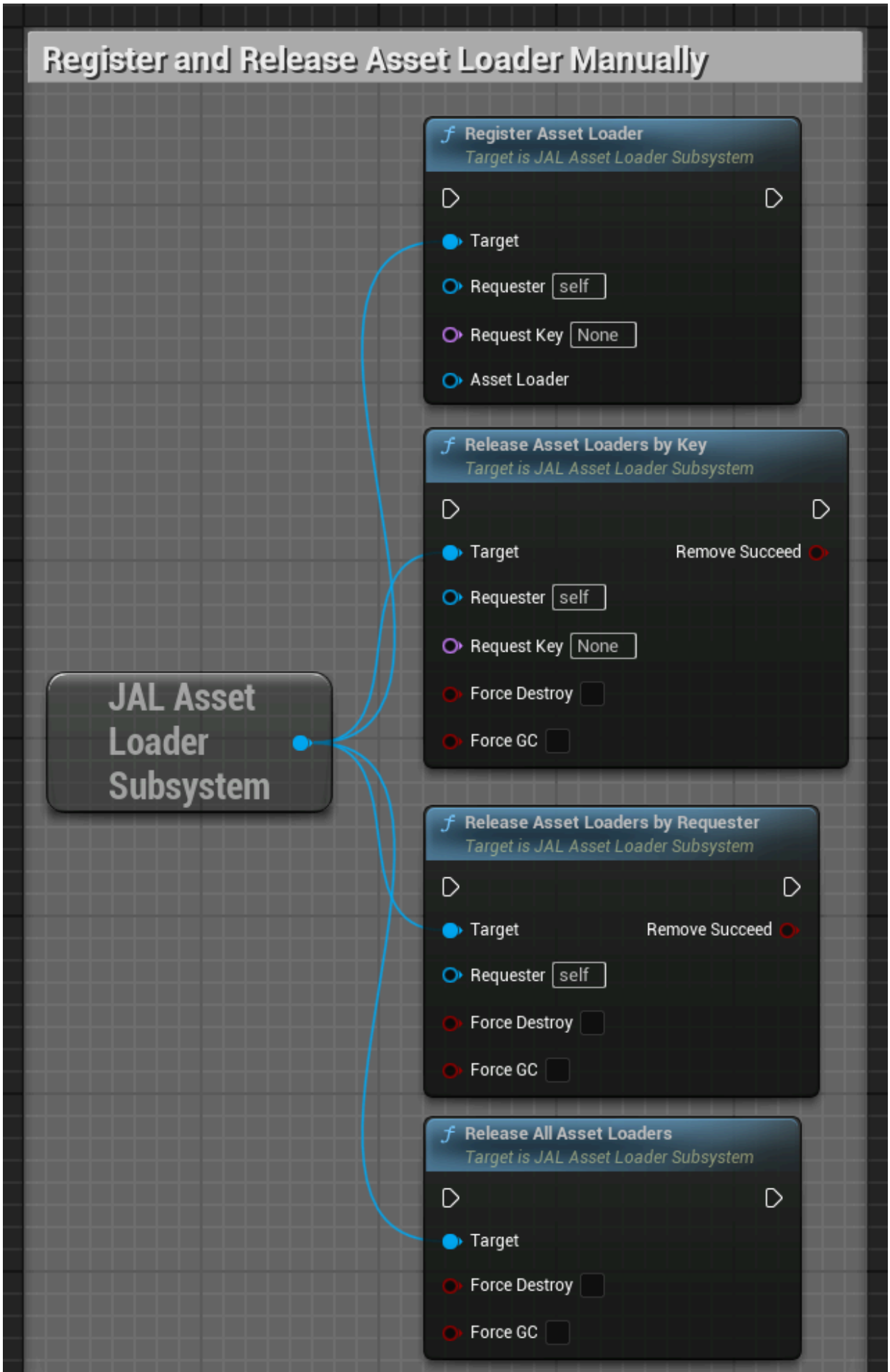
However, this release doesn't occur instantly—it is handled periodically (default: every 10 seconds), based on the settings in the `J Asset Loader` configuration.

Released Asset Loaders do not immediately trigger asset unloading.

Assets are only unloaded during **Garbage Collection (GC)** once there are no remaining references to the loaded assets and their loaders.

※ Since GC is not run frequently, the resources may stay in memory for a while even after being released.

You can also manually release requests registered in the subsystem.


Register and Release Asset Loader Manually
JAL Asset Loader Subsystem
Register Asset Loader
Target is JAL Asset Loader Subsystem
Target
Requester self
Request Key None
Asset Loader
Release Asset Loaders by Key
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Request Key None
Force Destroy
Force GC
Release Asset Loaders by Requester
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Force Destroy
Force GC
Release All Asset Loaders
Target is JAL Asset Loader Subsystem
Target
Force Destroy
Force GC

Note: `JAL Asset Loader Subsystem` inherits from `UWorldSubsystem`.
This means it will be reset and recreated when switching worlds.

---

## 🗺️ Sample Map

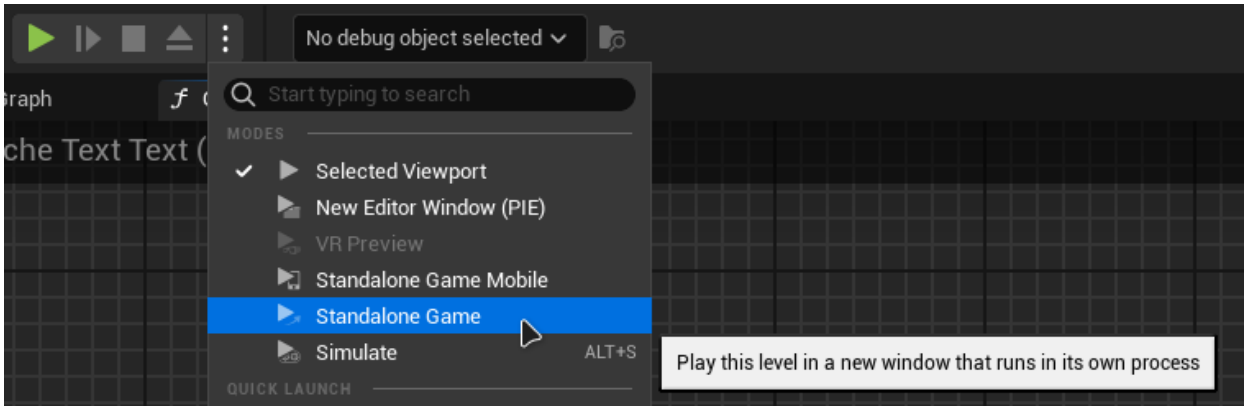
📹 Sample Map Preview: JAssetLoader_SampleMap.mp4

You can test the system by running the `JAL_SampleMap`.

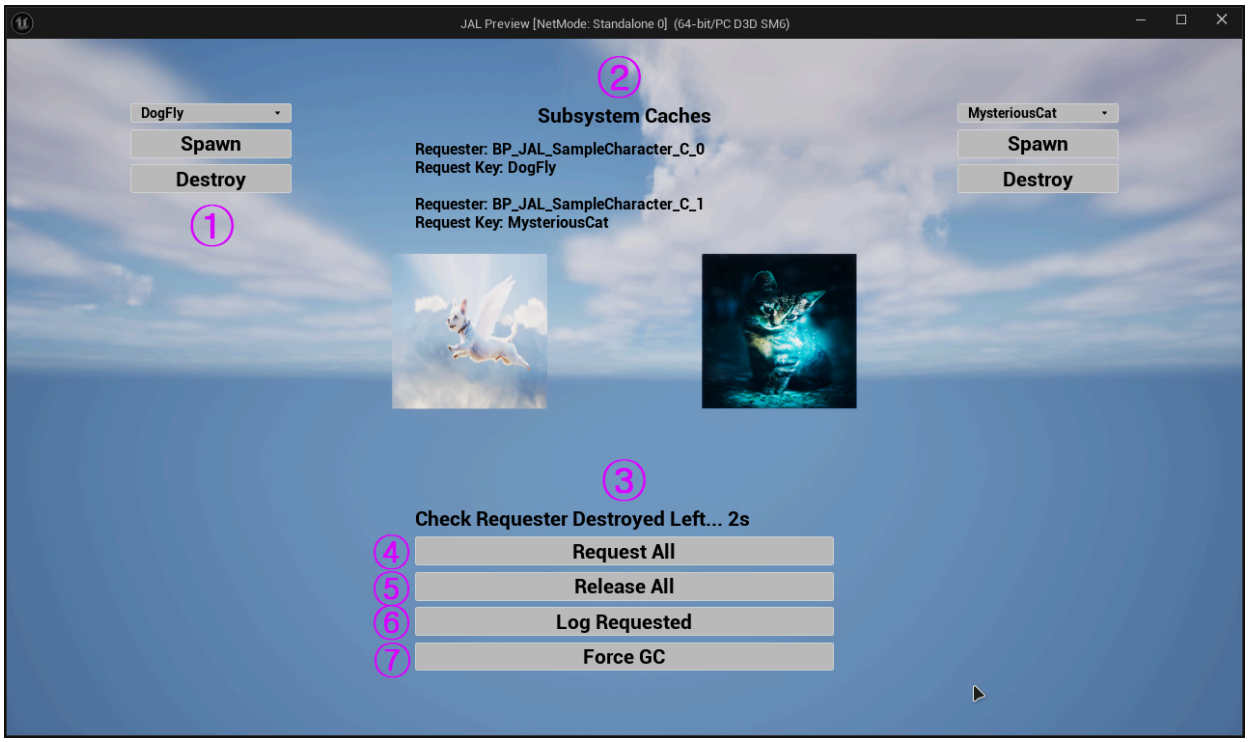
In this sample, an actor is spawned, and the material from a row in the `DT_JAL_SampleCharacter` DataTable is loaded and applied.

The material is loaded asynchronously via the Asset Loader.

※ Since resource unloading doesn't work correctly in PIE mode, testing in **Standalone mode** is strongly recommended.



## Main UI Overview




- **Spawn / Destroy Actor**
  Asynchronously load the material from the selected DataTable row and apply it to a newly spawned actor.

- **Show Registered Requests**
  View which asset requests are currently registered and managed by the subsystem.

- **Time Until Next Requester Check**
  The system periodically checks for invalid Requesters and releases related requests.

- **Request All from DataTable**
  Loads all resources from every row in `DT_JAL_SampleCharacter`.

- **Release All Requests**
  Force release all currently registered requests in the subsystem.

- **Print Request List**
  Output the list of registered requests to the console.

- **Force GC**
  Run Garbage Collection immediately to clear unreferenced resources.

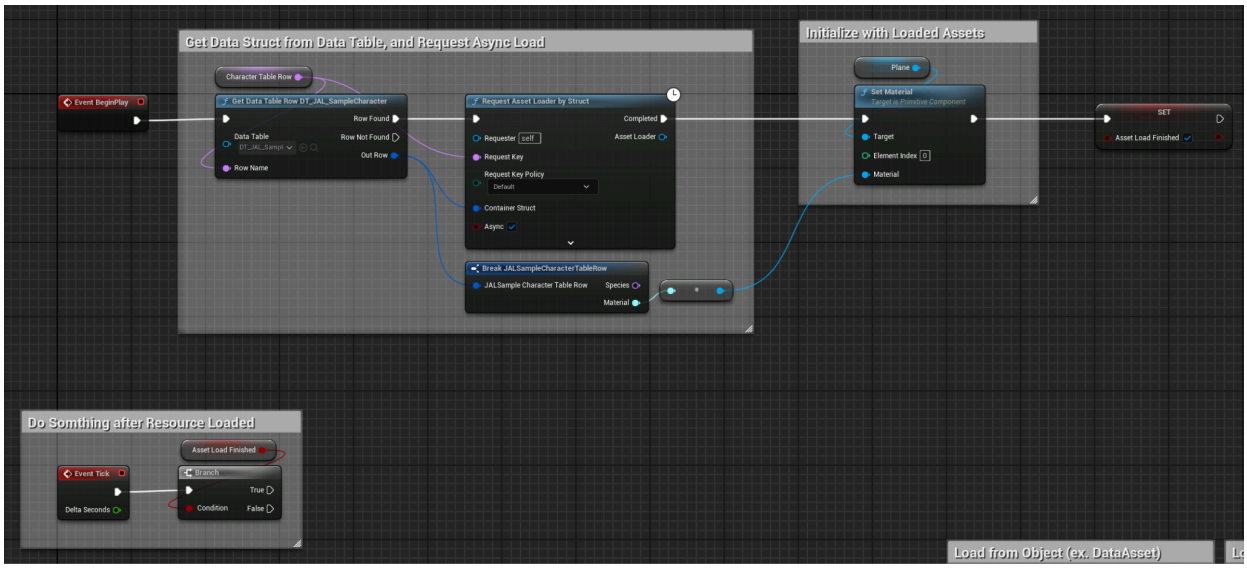
### Spawned Actor Behavior

Clicking the **Spawn** button creates a `BP_JAL_SampleCharacter` actor.

On `BeginPlay`, the actor asynchronously loads the material and applies it once loading completes.

You can view the exact loading and application process by opening the `JAL_SampleCharacter` Blueprint.

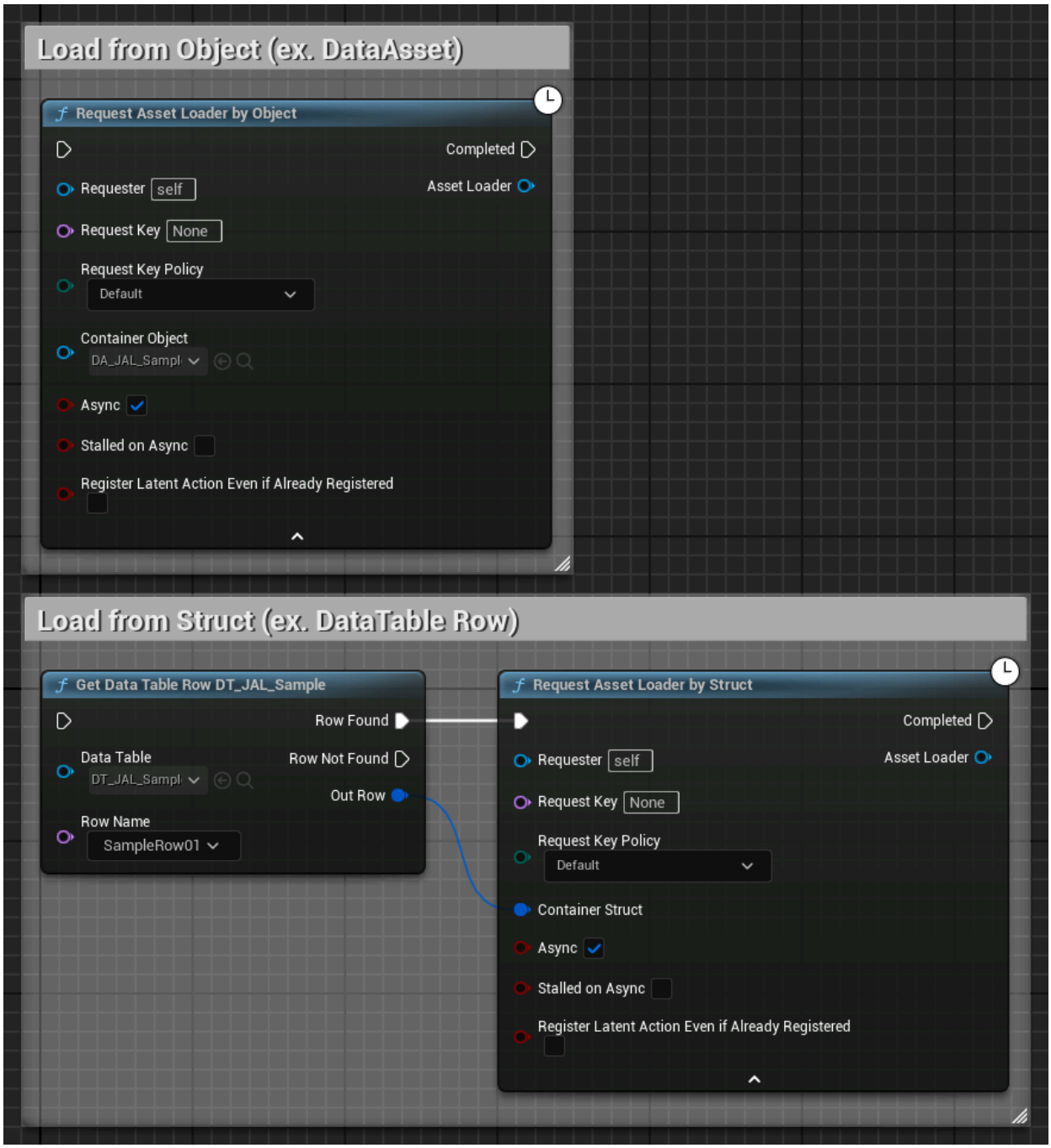
As shown in the preview video, if a character is already loaded, the material is applied immediately.




---

## 📎 Blueprint Functions

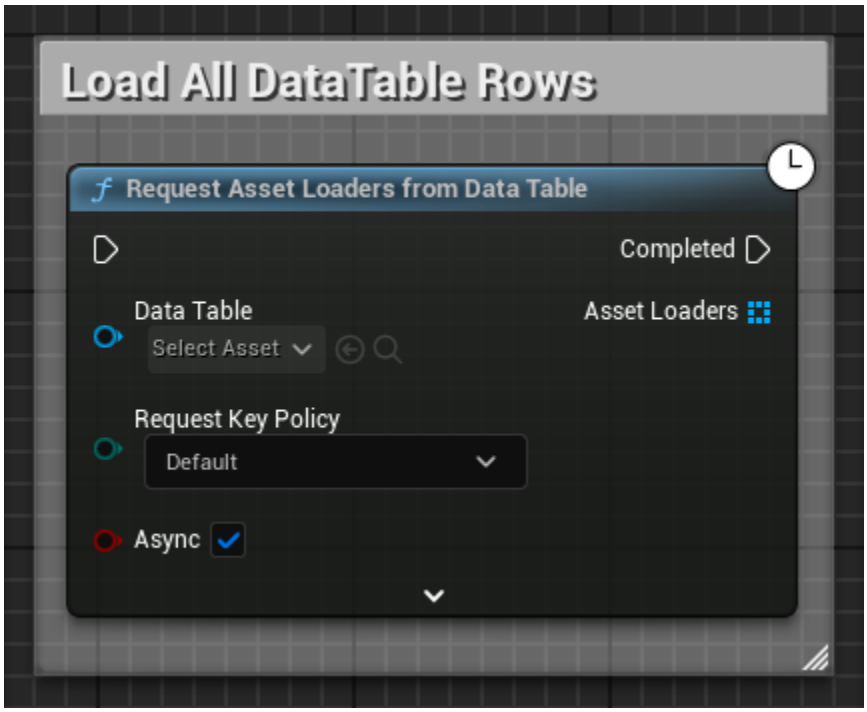
## 🔹 Request Asset Loader by Object / Struct




- Recursively loads all soft references in the input
- **Object**: Suitable for loading a single UObject (e.g., a DataAsset)
- **Struct**: Suitable for loading a struct (e.g., DataTable row)

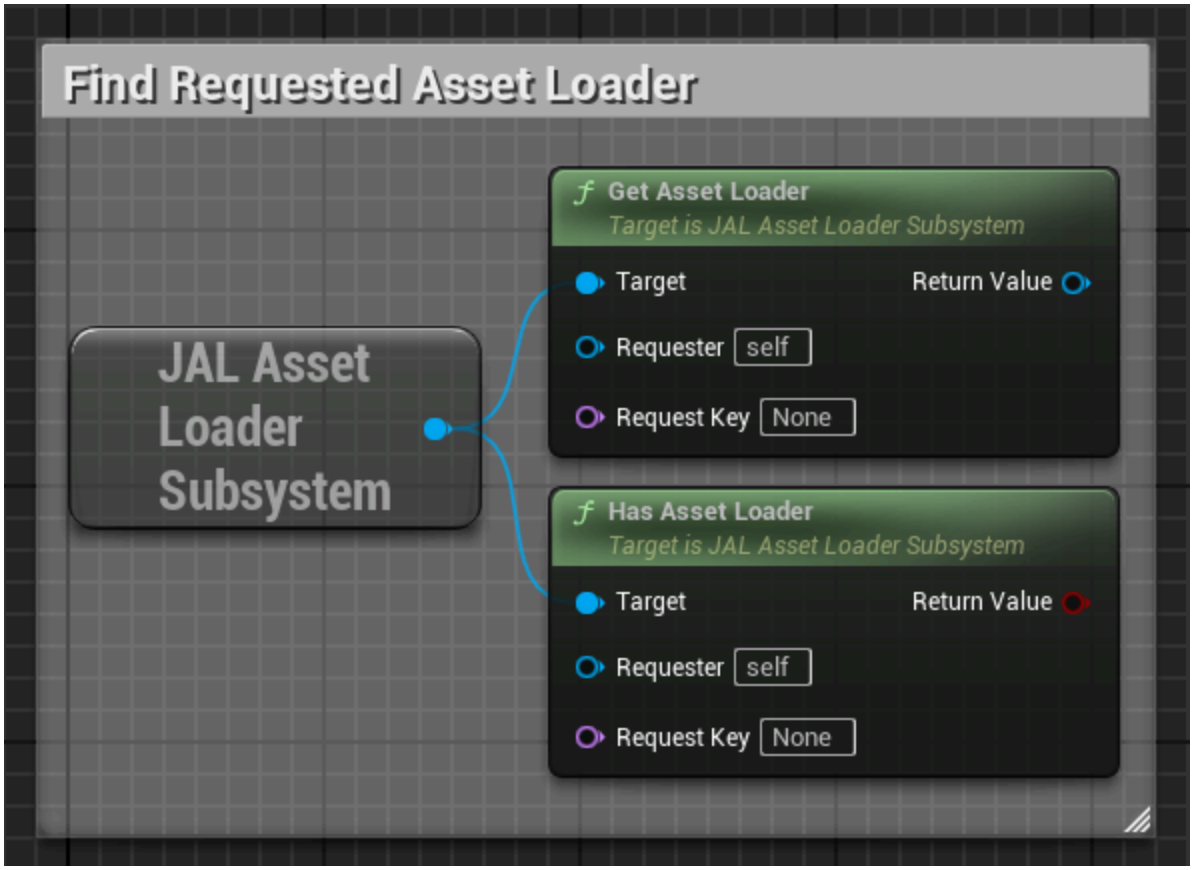
- Parameters
    - **Requester**: The object that initiates the request (used for lifetime management)
    - **Request Key**: A unique identifier for this request
    - **Request Key Policy**:
        - **Default** – Reuses existing loader if available
        - **Release and Request if Exists** – Cancels the existing request and starts a new one
        - **Always Request But Do Not Register** – Forces a new loader without registering it
    - **Container Object/Struct**: The UObject or UStruct containing soft references
    - **Async**: Whether to load asynchronously
    - **Stalled on Async**: Whether to stall execution until StartStalled() is manually called
    - **Register Latent Action Event if Already Registered**: If true, allows multiple execution of the Completed pin even if the node is already running

## 🔹 Request Asset Loaders from Data Table



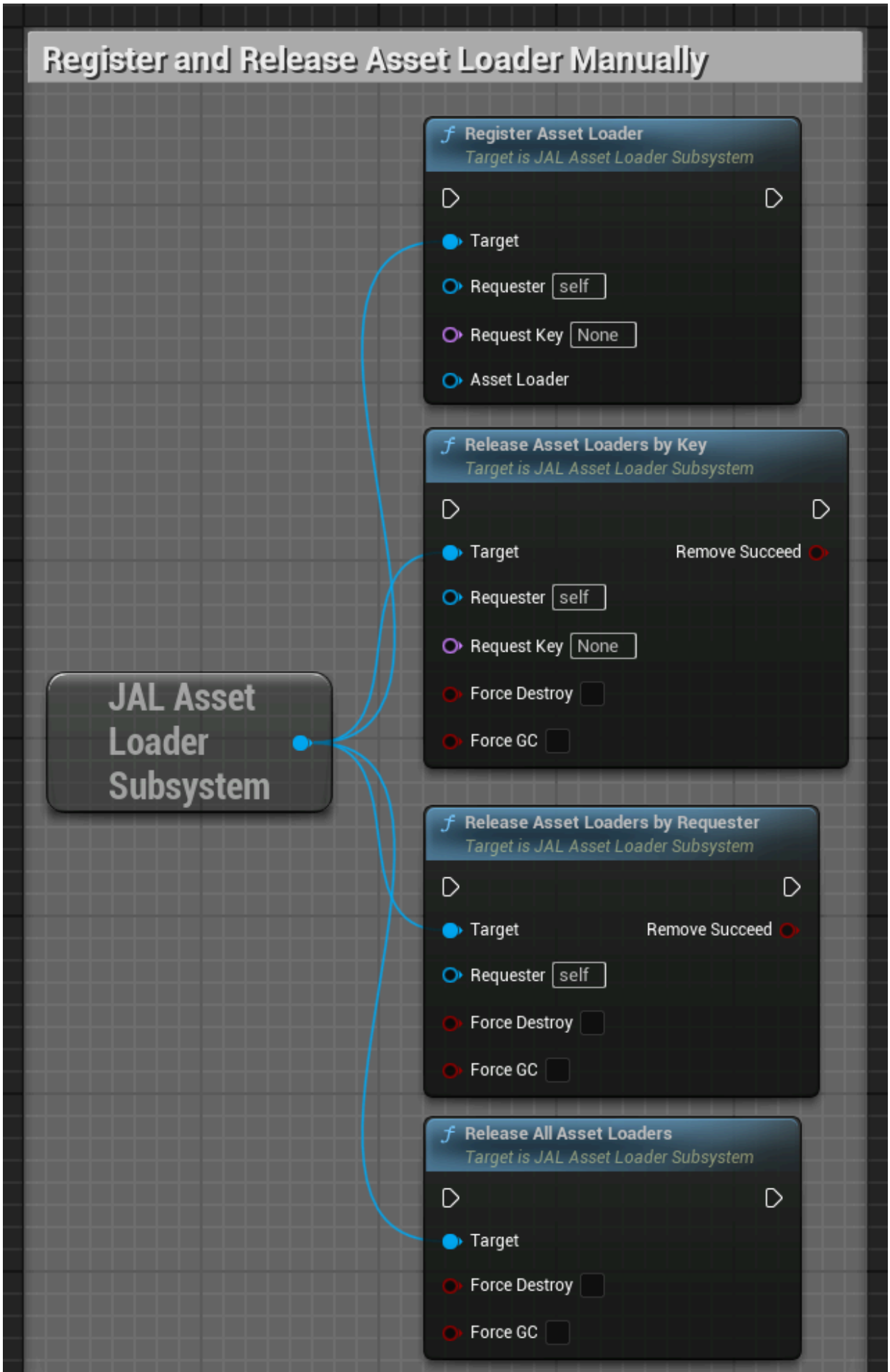

- Loads all assets referenced in every row of a specific Data Table.
- The Data Table asset is used as the Requester, and each row's key is used as the RequestKey when registering with the subsystem.

## 🔹 Get Asset Loader / Has Asset Loader




- Retrieves or checks existence of a loader for a specific Requester and Request Key

### 🔹 Register / Release Asset Loader


Register and Release Asset Loader Manually
JAL Asset Loader Subsystem
Register Asset Loader
Target is JAL Asset Loader Subsystem
Target
Requester self
Request Key None
Asset Loader
Release Asset Loaders by Key
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Request Key None
Force Destroy
Force GC
Release Asset Loaders by Requester
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Force Destroy
Force GC
Release All Asset Loaders
Target is JAL Asset Loader Subsystem
Target
Force Destroy
Force GC

- Manually register or release a loader from the subsystem

### 🔹 Log Requsters

```
LogJAL: ----------------------------------------
LogJAL: + Requester: MyGameCharacter_C_0
LogJAL:   |- Key: CharacterSkin
LogJAL:   |  |- Asset: SK_Character_Default
LogJAL:   |  |- Asset: M_Character_Base
LogJAL:   |- Key: CharacterWeapon
LogJAL:   |  |- Asset: SK_Weapon_Sword
LogJAL:   |  |- Asset: FX_Weapon_Trail
LogJAL: + Requester: MyGameController_C_2
LogJAL:   |- Key: PlayerUI
LogJAL:   |  |- Asset: WBP_PlayerHUD
LogJAL:   |  |- Asset: T_UI_Icons
LogJAL: + Requester: LevelBlueprintActor_1
LogJAL:   |- Key: LevelEnvironment
LogJAL:   |  |- Asset: SM_Environment_Trees
LogJAL:   |  |- Asset: SM_Environment_Rocks
LogJAL:   |- Key: BackgroundMusic
LogJAL:   |  |- Asset: A_Background_Music_01
LogJAL: ----------------------------------------
```

- Output the list of registered requests to the console.

---

## 🧩 Core Classes and Interfaces

### 🔹 UJALAssetLoader

Handles the actual load request and retains references to prevent GC

### 🔹 UJALAssetLoaderSubsystem

World Subsystem that manages loaders per Requester and Request Key

### 🔹 IJAssetLoaderContainerInterface

Interface required for recursive Soft Reference scanning inside nested UObjects

When using "Request Asset Loader by Object," the initially provided **Object A** does not need to implement this interface—it will be scanned directly for Soft References.

However, if **Object A** contains another object ***Object B*** as a Soft Reference and you want to also scan inside ***Object B***, then ***Object B*** must implement this interface.

Otherwise, ***Object B*** itself will be loaded, but its internal Soft References will not be included in the request.

---

## 📂 Loadable Property Types

- FSoftObjectPath
- FSoftClassPath
- TSoftObjectPtr
- TSoftClassPtr
- TWeakObjectPtr
- TLazyObjectPtr
- TObjectPtr, UObject*

- TScriptInterface
- FPrimaryAssetId
- FDataTableRowHandle
- FDataTableCategoryHandle
- Soft References inside TArray
- Soft References inside TSet
- Soft References inside TMap
- Soft References inside struct

---

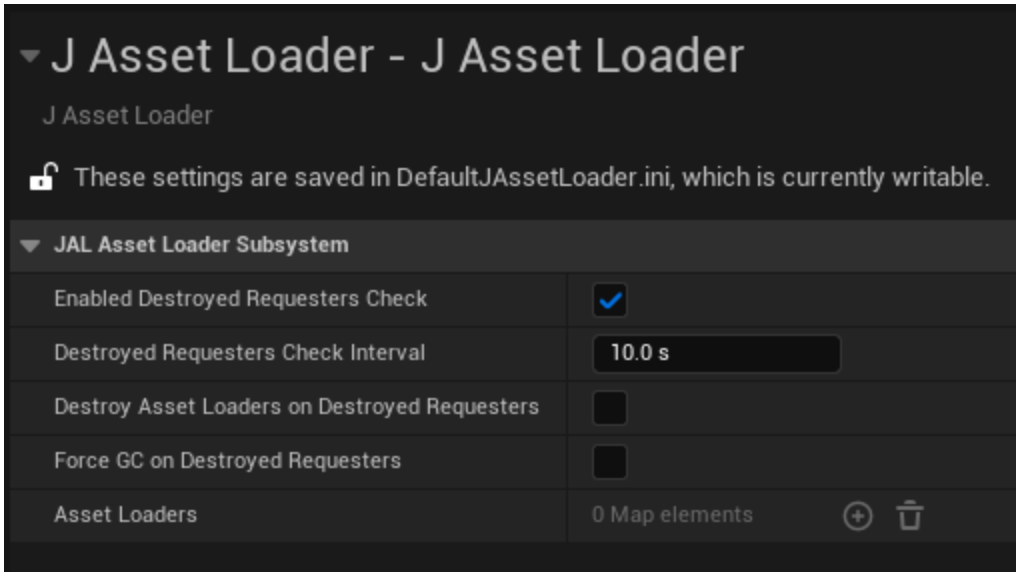
## 🧪 Tips

- Always assign a unique Request Key per logical operation
- When using **Stalled on Async**, call `StartStalled()` manually to proceed
- Unless using Do Not Register policy, you don't need to store the returned Asset Loader
- If the Requester is destroyed, related resources will be released

---

## ⚙️ J Asset Loader Config

In Project Settings, scroll to the bottom to find the **J Asset Loader** category. These are the available options:




- **Enabled Destroyed Requesters Check**
    - If enabled, checks Requesters periodically and releases loaders for deleted ones
- **Destroyed Requesters Check Interval**
    - Interval (in seconds) to check Requester validity
- **Destroy Asset Loaders on Destroyed Requesters**
    - Whether to explicitly delete Asset Loaders when their Requesters are destroyed
- **Force GC on Destroyed Requesters**
    - Whether to trigger Garbage Collection immediately after releasing loaders

---

## 🔄 Updates

## v1.1.0

- Added sample map, demo video, and testing resources
- New Blueprint nodes added:
    - **Request Asset Loaders from Data Table**
    - **Release All Asset Loaders**
    - **Log Requesters**

## v1.0.1

- Support new type: FDataTableRowHandle, FDataTableCategoryHandle

---

한국어

# 📘 J Asset Loader - 사용자 가이드

🎞️ **Preview Video**

👉 **Store Link**

💬 **Reddit**

---

## 🧩 개요

J Asset Loader는 UObject 또는 UStruct 내에 포함된 Soft Object Reference들을 재귀적으로 모두 수집하여 비동기 로드할 수 있게 해주는 월드 서브시스템입니다. 이 플러그인은 에셋 로딩으로 인한 히치를 방지하고, 요청자(Requester) 단위로 리소스를 안전하게 관리해 줍니다.

캐릭터, 아이템 등 복합적인 데이터 에셋 구조를 가진 게임에서 Soft Reference로 된 리소스를 손쉽게 비동기로 로딩할 수 있도록 설계되었습니다.

---

## 🧩 주요 기능 요약

- UObject 또는 UStruct 내부의 모든 Soft Object Reference를 재귀적으로 수집 및 로딩
- 블루프린트 완전 지원
- 로드된 리소스 자동 참조 유지로 GC 방지
- 요청자/요청 키 기반의 캐싱 및 중복 방지 정책 지원
- 게임인스턴스 서브시스템이 아닌 월드 서브시스템 기반 구조

---

## 🧪 언제 필요한가요?

게임을 개발하다 보면 Soft Reference 형태의 에셋들을 한 번에 로드해주는 기능이 꼭 필요할 때가 있습니다.

예를 들어 캐릭터를 정의하는 데이터테이블이 있다고 해봅시다. 하나의 행에는 캐릭터의 능력치, 모델 정보, 기본 장착 무기 정보 등이 포함되어 있을 수 있습니다.

이 정보들은 각각 CharacterModelDataAsset, WeaponDataAsset 등의 데이터 에셋으로 정의되고, 그 안에는 또 Mesh, Material 같은 리소스들이 있을 수 있습니다.

만약 이들을 Hard Reference(UStaticMesh*, TObjectPtr<>)로 연결하면 데이터테이블이 로드될 때 모든 행의 캐릭터 리소스가 한꺼번에 메모리에 올라오게 됩니다.

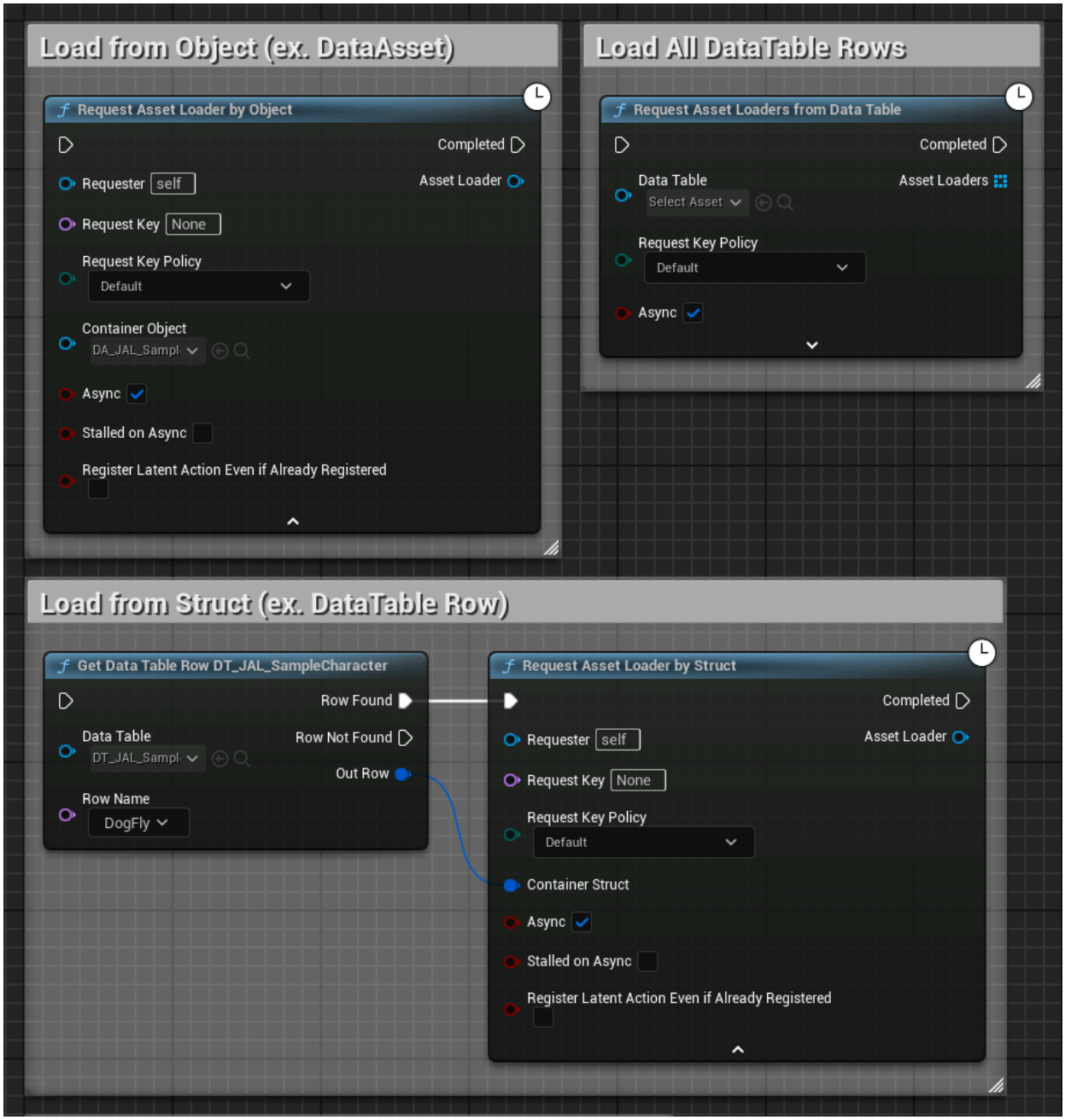
이런 비효율을 막기 위해 Soft Reference로 관리할 수 있지만, 문제는 이렇게 하면 얽힌 Soft Reference들을 한꺼번에 로드하는 일이 쉽지 않다는 것입니다. 동기로 로드하면 히치가 발생할 수 있고, 비동기 로딩 구현은 복잡합니다.

J Asset Loader는 이런 상황을 해결하기 위해 만들어졌습니다. 구조체 또는 UObject 내부의 Soft Reference들을 재귀적으로 탐색하여 한 번에 로드 요청할 수 있게 도와주는 서브시스템입니다.

---

## 🎬 동작 흐름

로드 요청은 `Request Asset Loader by Object` 또는 `Request Asset Loader by Struct` 블루프린트 노드를 통해 시작합니다.




요청이 성공하면 `JAL Asset Loader`에 담긴 뒤, `JAL Asset Loader Subsystem`에 등록됩니다.

이로 인해 로드된 에셋이 다른 곳에서 참조되지 않더라도, 서브시스템이 해당 에셋을 관리하고 있기 때문에 자동으로 언로드되지 않습니다.

※ 블루프린트 노드의 인자를 통해 서브시스템에 등록하지 않도록 설정할 수도 있습니다.

이 경우 반환된 `Asset Loader`를 직접 관리하거나, 로드된 에셋을 외부에서 참조하고 있어야 언로드를 막을 수 있습니다.

서브시스템에 등록된 `Asset Loader`는 **Requester**가 삭제되었을 때 자동으로 릴리즈됩니다.

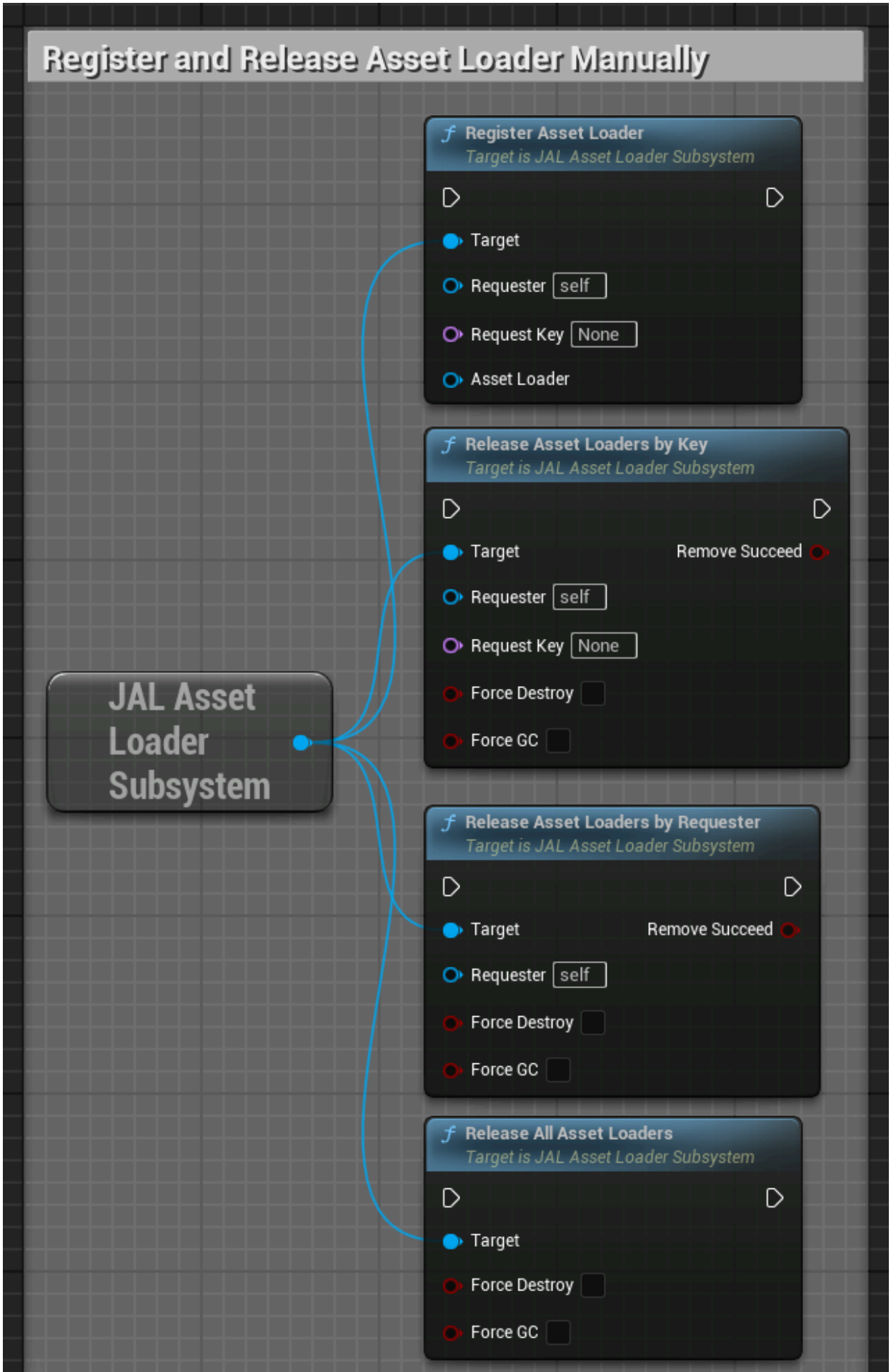
단, 삭제 즉시가 아니라 설정된 주기(기본값: 10초)마다 상태를 확인하여 유효하지 않은 Requester가 발견되면 그에 해당하는 모든 `Asset Loader`가 릴리즈됩니다.

릴리즈된다고 해서 에셋이 즉시 언로드되는 것은 아닙니다.

로드된 에셋 및 `JAL Asset Loader`에 대한 참조가 완전히 사라졌을 때만 **가비지 콜렉션(GC)**을 통해 언로드가 진행됩니다.

※ GC는 정기적으로 수행되지 않기 때문에, 참조가 없어도 한동안 메모리에 유지될 수 있습니다.

또한, 서브시스템에 등록된 요청은 수동으로 릴리즈할 수도 있습니다.


Register and Release Asset Loader Manually
JAL Asset Loader Subsystem
Register Asset Loader
Target is JAL Asset Loader Subsystem
Target
Requester self
Request Key None
Asset Loader
Release Asset Loaders by Key
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Request Key None
Force Destroy
Force GC
Release Asset Loaders by Requester
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Force Destroy
Force GC
Release All Asset Loaders
Target is JAL Asset Loader Subsystem
Target
Force Destroy
Force GC

참고: `JAL Asset Loader Subsystem`은 `UWorldSubsystem`을 상속하기 때문에 월드가 바뀌면 초기화되며 새로 생성됩니다.

---

## 🗺️ 샘플 맵

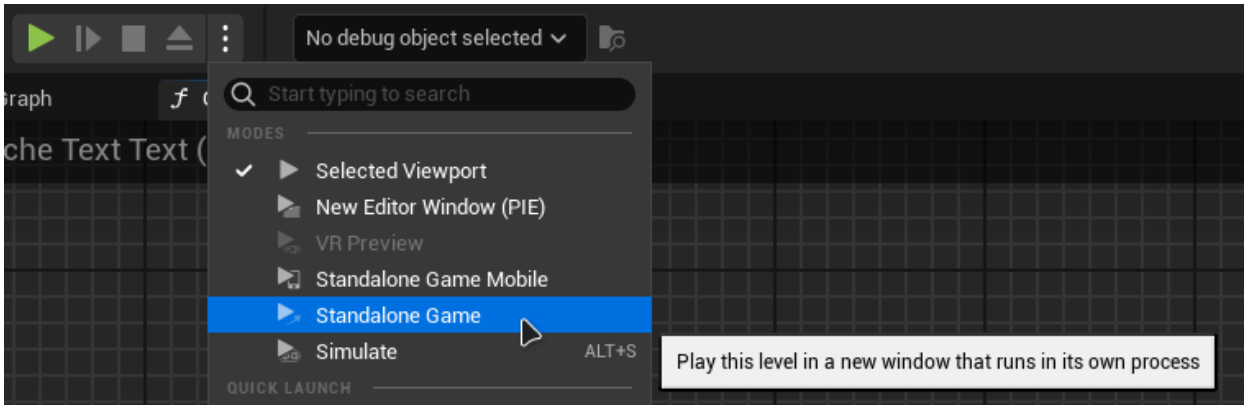
📹 샘플 맵 프리뷰 영상: JAssetLoader_SampleMap.mp4

`JAL_SampleMap` 맵을 실행하면 테스트를 직접 해볼 수 있습니다.

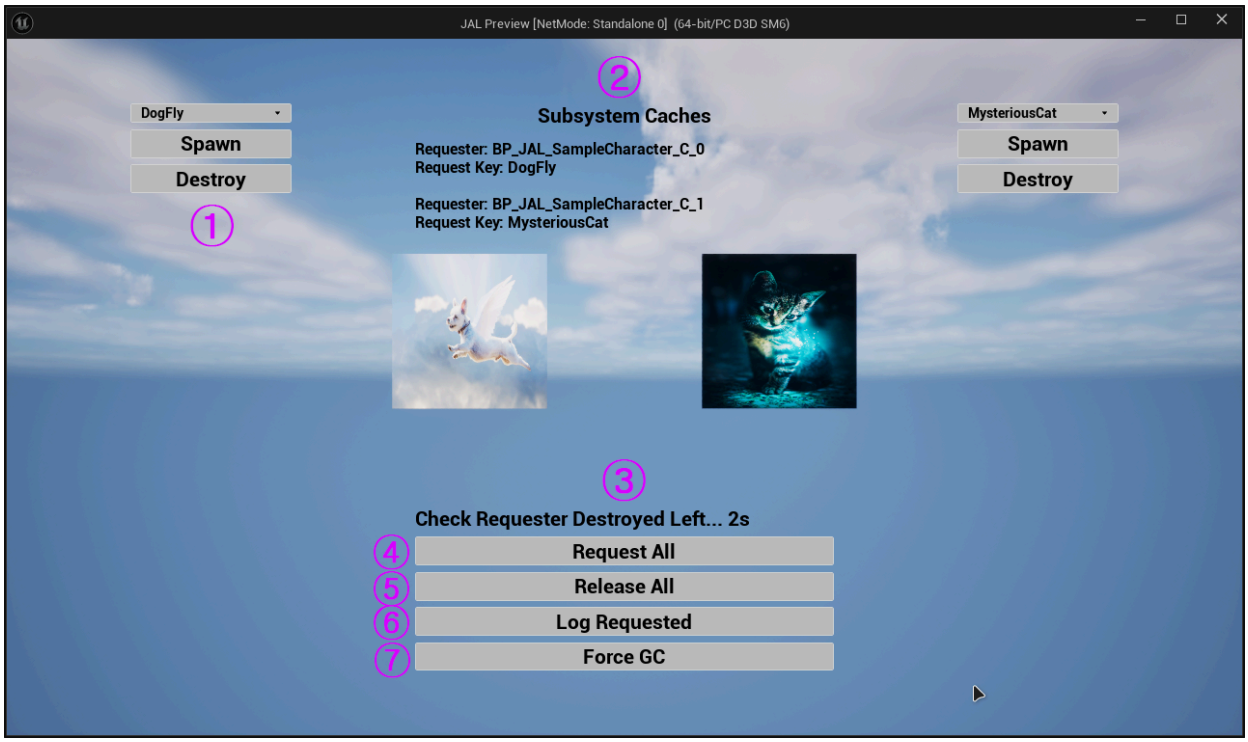
샘플은 액터를 스폰한 뒤, `DT_JAL_SampleCharacter` 데이터테이블의 행에서 머터리얼을 가져와 적용하는 구조입니다.

머터리얼 리소스는 **Asset Loader**를 통해 비동기 로드됩니다.

※ **PIE** 환경에서는 언로드가 정상 작동하지 않기 때문에 **Standalone** 모드에서 테스트하시길 권장합니다.



## 메인 화면 설명




1. 액터 생성 및 삭제
   선택된 데이터테이블 행의 머터리얼을 비동기로 로드하여 새로 생성된 액터에 적용합니다.

2. 서브시스템에 등록된 요청 정보 확인
   현재 어떤 요청들이 등록되어 관리 중인지 확인할 수 있습니다.

3. 다음 **Requester** 유효성 검사까지 남은 시간
   주기적으로 유효하지 않은 Requester를 탐색하여 관련 요청을 릴리즈합니다.

4. 데이터테이블 전체 요청.
   `DT_JAL_SampleCharacter`의 모든 행의 리소스를 로드합니다.

5. 모든 요청 릴리즈
   현재 서브시스템에 등록된 모든 요청을 강제로 릴리즈합니다.

6. 요청 리스트 출력
   등록된 요청들을 콘솔에 출력합니다.

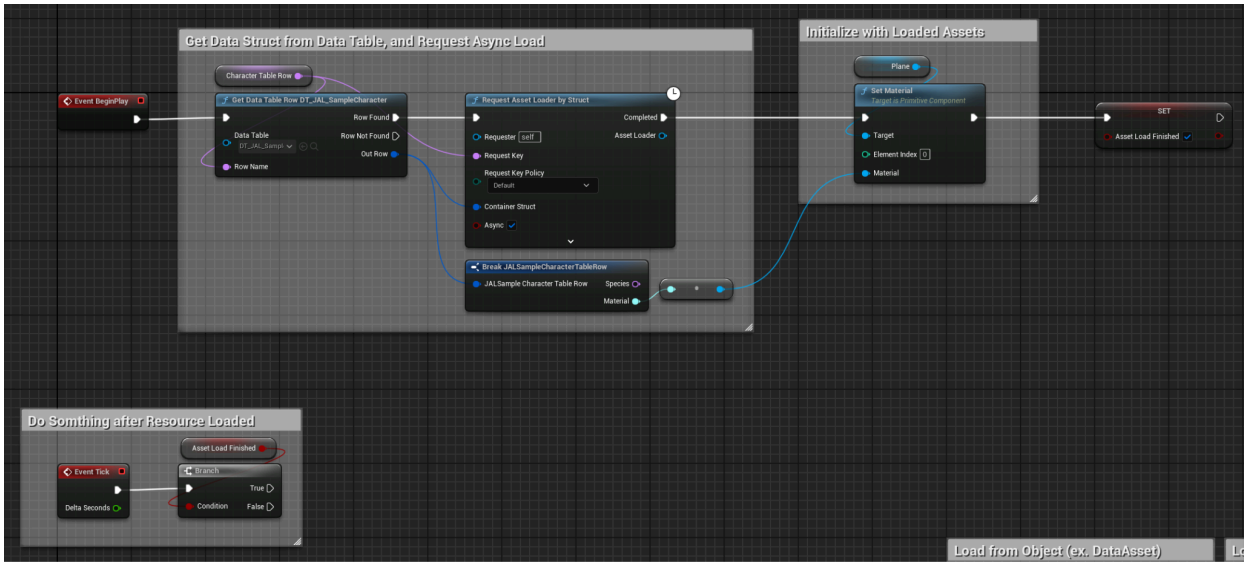
7. 가비지 콜렉션 강제 실행
   즉시 GC를 실행하여 참조되지 않은 리소스를 정리합니다.

### 스폰된 액터 동작

`Spawn` 버튼을 누르면 `BP_JAL_SampleCharacter` 액터가 생성됩니다.

이 액터는 `BeginPlay`에서 머터리얼을 비동기 로드한 뒤, 로드 완료 시 자동으로 머터리얼을 적용합니다.

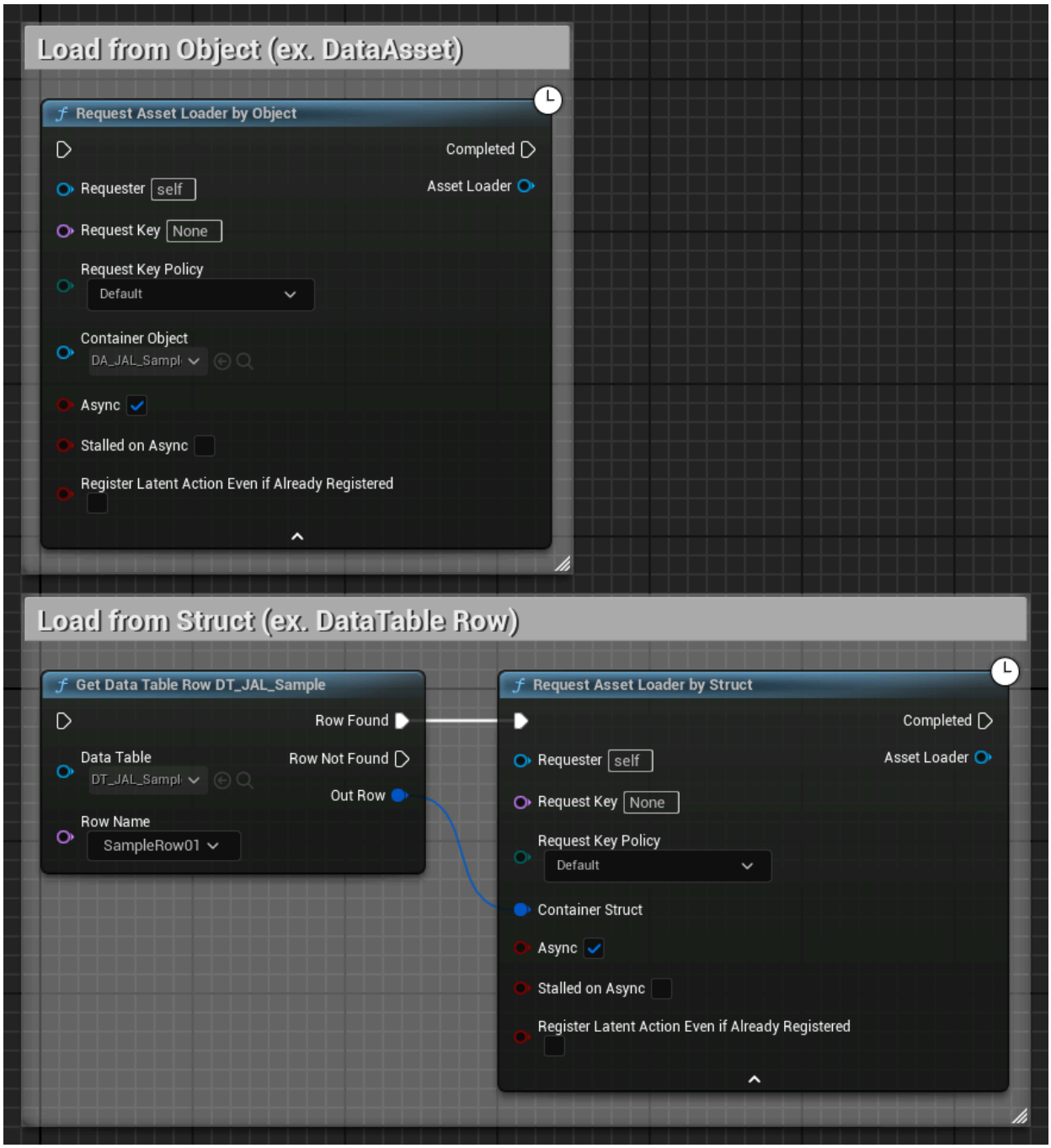
실제 로드 및 처리 방식은 `JAL_SampleCharacter` 블루프린트를 열어 확인하실 수 있습니다.




---

## 📎 블루프린트 함수

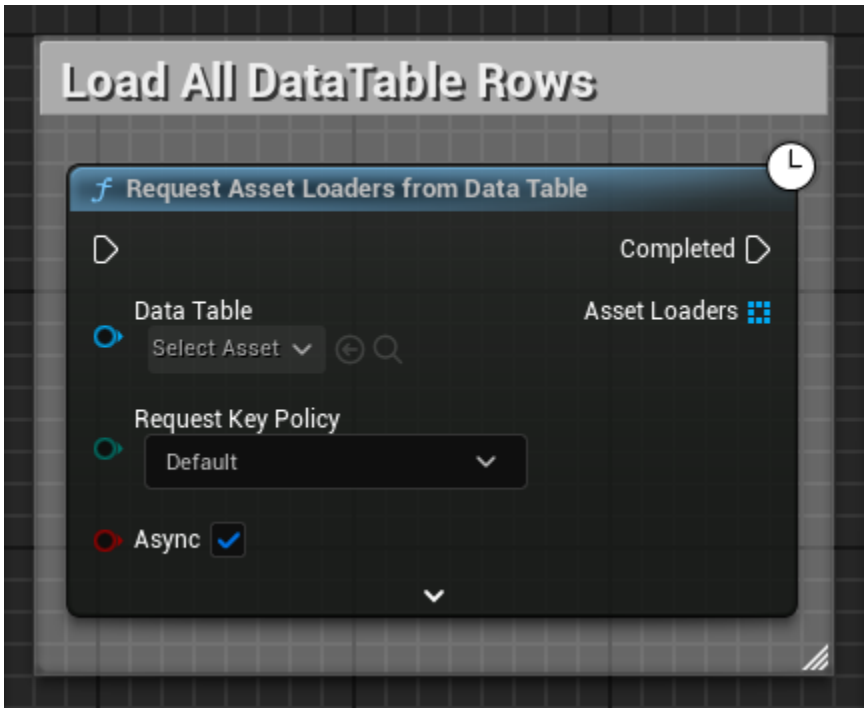
## 🔹 Request Asset Loader by Object / Struct




- 지정된 입력의 Soft Reference 에셋을 재귀적으로 모두 로드
- Object: 단일 UObject(예: DataAsset) 로딩에 적합
- Struct: 구조체(예: DataTable Row) 로딩에 적합

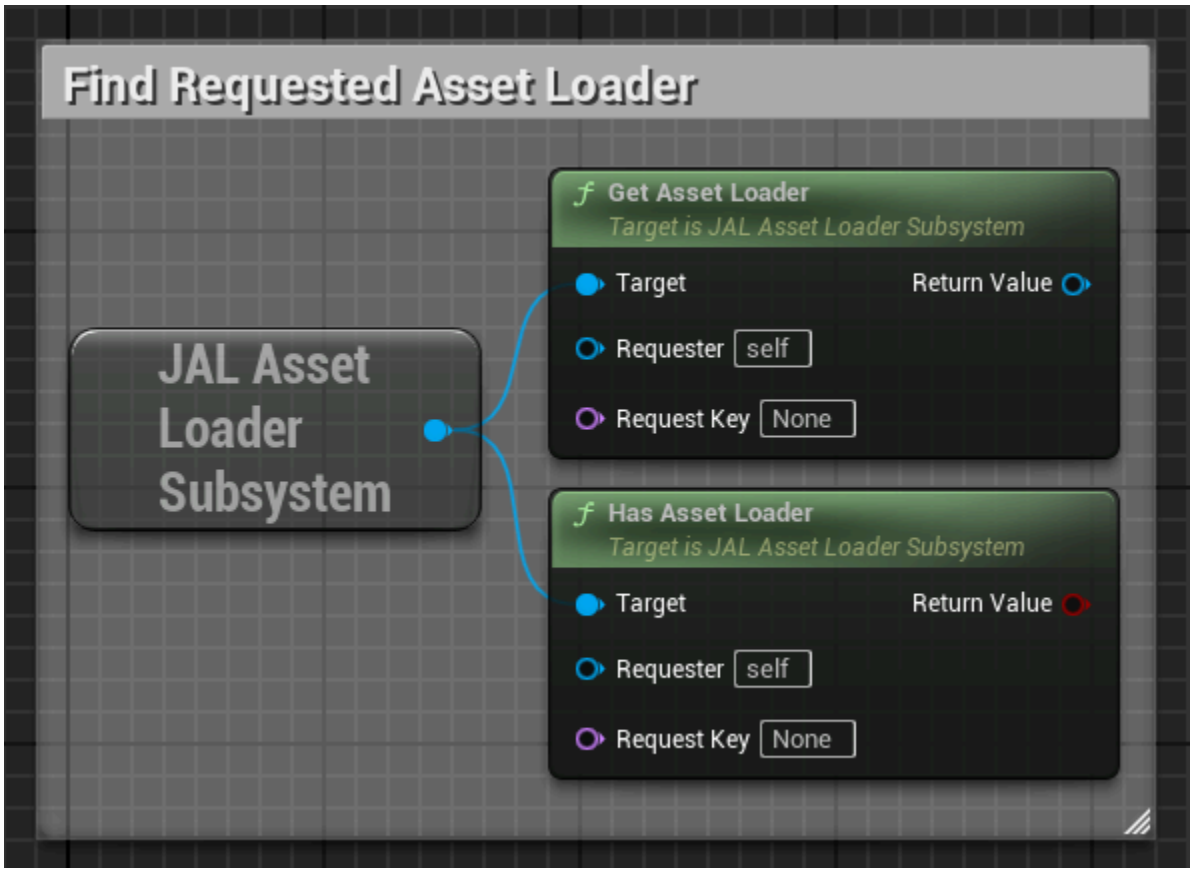
- 파라미터
    - **Requester**: 요청을 실행한 오브젝트 (수명 관리 기준)
    - **Request Key**: 요청을 구분할 수 있는 고유 키
    - **Request Key Policy**: 동일 키 존재 시 처리 방식
        - Default - 기존 요청이 있다면 재사용하고, 없거나 실패한 경우 새로 요청
        - Release and Request if Exists - 동일 키 요청이 존재하면 기존 요청을 해제하고 새로 요청
        - Always Request But Do Not Register - 항상 새로 요청하며, 서브시스템에 등록하지 않음
    - **Container Object/Struct**: Soft Reference를 포함한 구조체 또는 오브젝트
    - **Async**: 비동기 로드 여부
    - **Stalled on Async**: 로딩 요청을 지연시킬지 여부
    - **Register Latent Action Event if Already Registered**: 이 블루프린트 노드가 이미 실행 중이더라도 다시 실행할지를 결정

## 🔹 Request Asset Loaders from Data Table



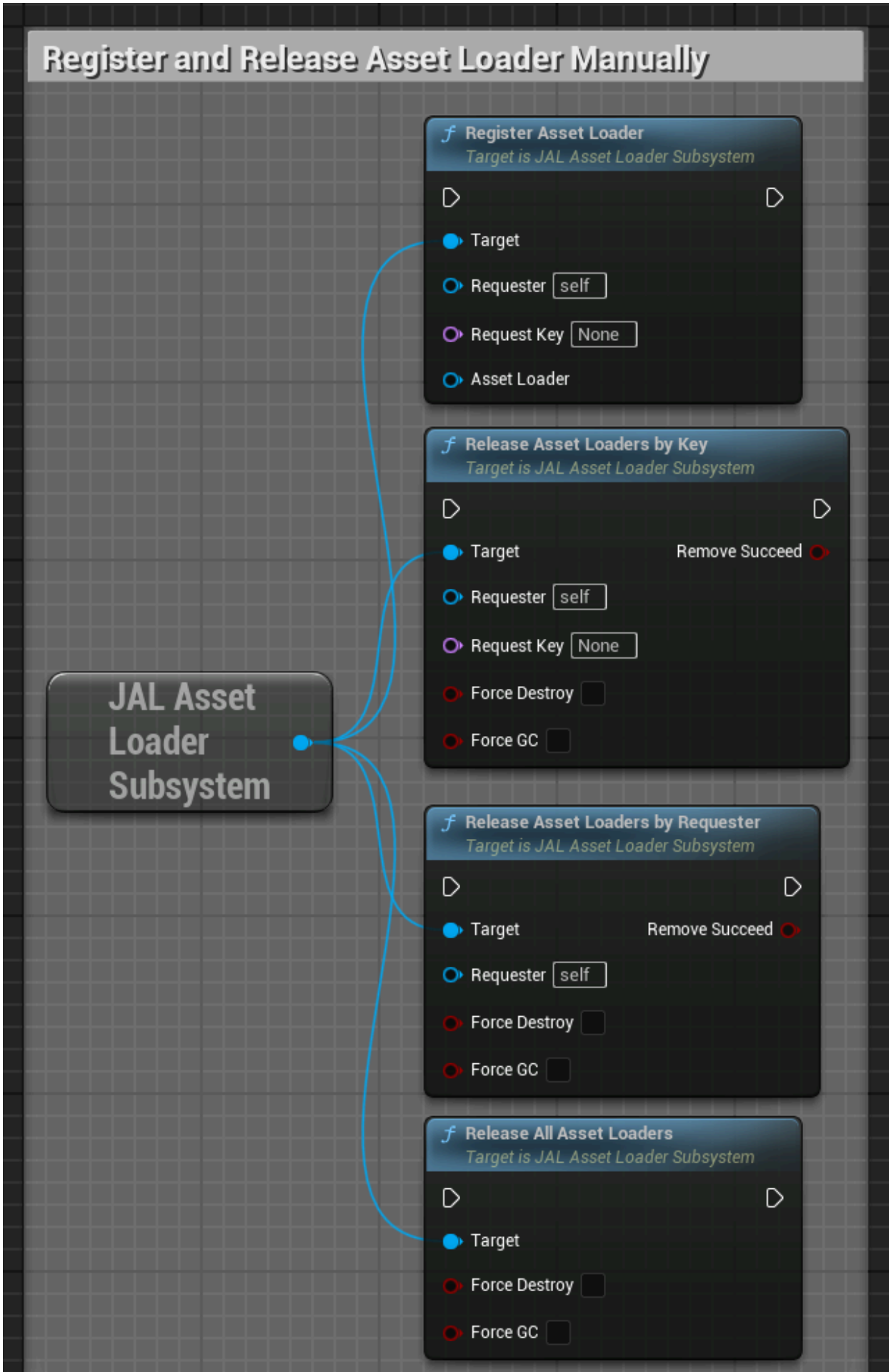

- 특정 데이터 테이블의 모든 행의 에셋을 로드합니다.
- DataTable 에셋을 Requster로, 각 행의 Key를 RequestKey로 사용하여 서브시스템에 등록합니다.

### 🔹 Get Asset Loader / Has Asset Loader




- 특정 요청자와 요청 키에 해당하는 Asset Loader를 가져오거나 존재 여부 확인

- **Register / Release Asset Loader**


Register and Release Asset Loader Manually
JAL Asset Loader Subsystem
Register Asset Loader
Target is JAL Asset Loader Subsystem
Target
Requester self
Request Key None
Asset Loader
Release Asset Loaders by Key
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Request Key None
Force Destroy
Force GC
Release Asset Loaders by Requester
Target is JAL Asset Loader Subsystem
Target
Remove Succeed
Requester self
Force Destroy
Force GC
Release All Asset Loaders
Target is JAL Asset Loader Subsystem
Target
Force Destroy
Force GC

- Asset Loader를 수동으로 등록 또는 해제

### 🔹 Log Requsters

```
LogJAL: ----------------------------------------
LogJAL: + Requester: MyGameCharacter_C_0
LogJAL:   |- Key: CharacterSkin
LogJAL:   |  |- Asset: SK_Character_Default
LogJAL:   |  |- Asset: M_Character_Base
LogJAL:   |- Key: CharacterWeapon
LogJAL:   |  |- Asset: SK_Weapon_Sword
LogJAL:   |  |- Asset: FX_Weapon_Trail
LogJAL: + Requester: MyGameController_C_2
LogJAL:   |- Key: PlayerUI
LogJAL:   |  |- Asset: WBP_PlayerHUD
LogJAL:   |  |- Asset: T_UI_Icons
LogJAL: + Requester: LevelBlueprintActor_1
LogJAL:   |- Key: LevelEnvironment
LogJAL:   |  |- Asset: SM_Environment_Trees
LogJAL:   |  |- Asset: SM_Environment_Rocks
LogJAL:   |- Key: BackgroundMusic
LogJAL:   |  |- Asset: A_Background_Music_01
LogJAL: ----------------------------------------
```

- 서브시스템에 등록되어 있는 요청들을 콘솔에 출력합니다.

---

## 🧩 주요 클래스 및 인터페이스 설명

### 🔹 UJALAssetLoader

개별 요청 단위로 에셋을 로드하고 보관하는 핵심 객체입니다. Soft Reference 로딩을 실제로 수행하며, 내부적으로 참조를 유지해 GC를 방지합니다.

### 🔹 UJALAssetLoaderSubsystem

월드 기반 서브시스템으로 여러 요청자(Requester)와 요청 키(Request Key)에 따른 Asset Loader 객체들을 생성, 캐싱, 관리합니다.

### 🔹 IJAssetLoaderContainerInterface

Asset Loader가 UObject 내부를 탐색할 수 있도록 하기 위해 필요한 인터페이스입니다. UObject에 Soft Reference 탐색 기능을 적용하려면 이 인터페이스를 구현해야 하며, 이 인터페이스를 구현하지 않은 UObject는 탐색 대상에서 제외됩니다.

좀 더 쉽게 풀어서 설명드리자면,

"Request Asset Loader by Object"를 사용할 때, Container Object에 입력된 **Object A**는 이 인터페이스를 구현하고 있지 않아도 Soft Reference를 모두 찾아서 요청 리스트에 담습니다.

하지만 **Object A**가 Soft Reference로 가지고 있는 또 다른 ***Object B***의 내부까지 재귀적으로 탐색하여 Soft Reference를 함께 요청하고 싶다면, ***Object B***는 반드시 이 인터페이스를 구현하고 있어야 합니다.

그렇지 않으면 ***Object B*** 자체는 요청되지만, 그 내부에 포함된 Soft Reference들은 요청 리스트에 포함되지 않습니다.

---

## 📂 로드 가능한 프로퍼티 타입

- FSoftObjectPath
- FSoftClassPath
- TSoftObjectPtr
- TSoftClassPtr
- TWeakObjectPtr

- TLazyObjectPtr
- TObjectPtr, UObject*
- TScriptInterface
- FPrimaryAssetId
- FDataTableRowHandle
- FDataTableCategoryHandle
- TArray 내부의 Soft Reference
- TSet 내부의 Soft Reference
- TMap 내부의 Soft Reference
- 구조체 내부의 Soft Reference

---

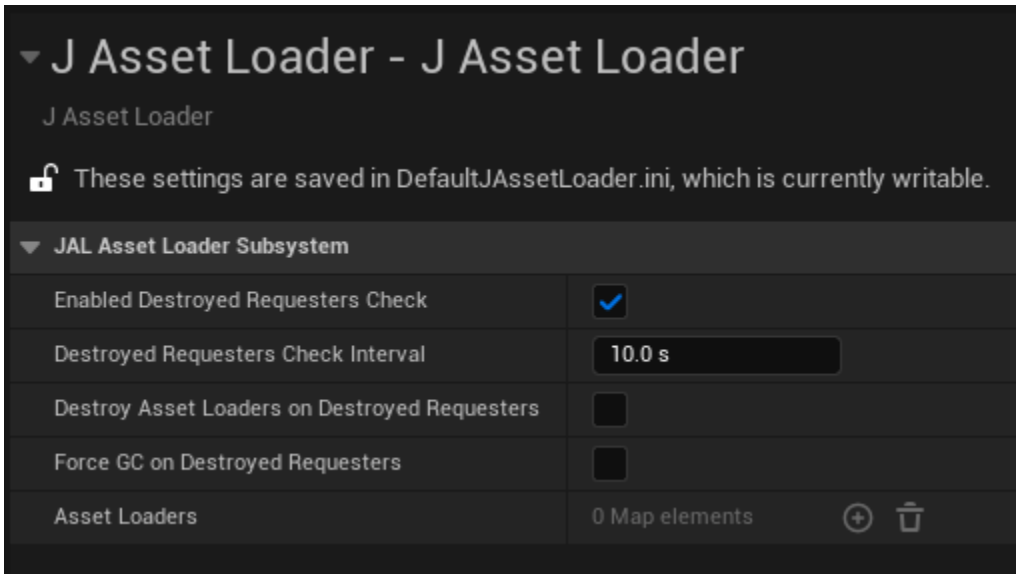
## 🧪 팁

- 논리적으로 구분되는 로딩마다 고유한 Request Key를 설정하세요.
- **Stalled on Async**를 true로 설정한 경우, 반환된 Asset Loader에서 `StartStalled()`를 직접 호출해야 로딩이 시작됩니다.
- Do Not Register 정책을 사용하는 경우가 아니라면 반환된 Asset Loader를 따로 저장할 필요는 없습니다.
- Requester가 삭제되면 연결된 리소스도 Release됩니다.

---

# ⚙️ J Asset Loader 설정 (Config)

Project Setting 창의 가장 아래쪽에서 **J Asset Loader** 카테고리를 확인할 수 있으며, 다음 설정을 조절할 수 있습니다:




- **Enabled Destroyed Requesters Check**
    - 켜져 있으면 Requester를 주기적으로 확인하여 삭제된 경우 해당 Asset Loader를 자동으로 Release합니다.
- **Destroyed Requesters Check Interval**
    - 위 기능이 활성화되어 있을 때, Requester 상태를 검사하는 주기를 초 단위로 지정합니다.
- **Destroy Asset Loaders on Destroyed Requesters**
    - 삭제된 Requester가 발견되었을 때, 관리 중인 Asset Loader를 명시적으로 삭제할지 여부를 설정합니다.
- **Force GC on Destroyed Requesters**
    - 삭제된 Requester가 발견되었을 때, Asset Loader를 Release한 후 즉시 Garbage Collection을 강제로 수행할지 여부를 설정합니다.

---

# 🔄 Updates

## v1.1.0

- 샘플 맵 및 영상, 리소스 추가
- 블루프린트 노드 추가
    - **Request Asset Loaders from Data Table**
    - **Release All Asset Loaders**
    - **Log Requesters**

## v1.0.1

- 신규 타입 지원: FDataTableRowHandle, FDataTableCategoryHandle

---