Ultimate Starter Kit

Documentation

Henry Jooste

None

Table of contents

1. Getting Started

1.1 Requirements

The Ultimate Starter Kit plugin is only available for Unreal Engine 4.27 and newer. The plugin also depends on the following plugins:

1. Niagara
2. Enchanced Input

1.2 Installation

1. Download the latest release from GitHub
2. Navigate to `C:\Program Files\Epic Games\UE_{VERSION}\Engine\Plugins`
3. Create a `Marketplace` folder if needed
4. Extract the release and copy to the `Marketplace` folder
5. Open Unreal Engine
6. Click on `Edit > Plugins`
7. Enable the plugin under the `Built-in > Other` category
8. Restart Unreal Engine

1.3 Plugin Content

The Ultimate Starter Kit plugin includes content that can be used in your Blueprints. You might need to enable this first:

1. Open the `Content Browser`
2. Click on the settings button
3. Enable the `Show Plugin Content` setting

2. Support the Project

I created this plugin because I love making games and I want to share my passion and knowledge with other game developers. I want to make Unreal Engine more accessible and powerful for everyone who wants to create amazing games.

But creating and maintaining this plugin is not easy. It requires a lot of work, resources, and expertise. That's why I need your support to keep this project alive and growing. By donating to this project, you can:

- Help me cover the costs of development, testing, documentation, and other expenses
- Encourage me to continue working on the project and adding new features
- Enable me to spend more time on the project and less on other jobs
- Join a community of like-minded game developers who care about the same plugin

Donating to this project is not only beneficial for me, but also for you as a user. By donating to this project, you can:

- Improve the plugin you rely on daily
- Learn new skills or improve on existing ones by studying the code or contributing yourself
- Gain a deeper knowledge about the plugin you're using
- Build your reputation and career by showing your involvement and expertise
- Have fun and satisfaction by being part of something bigger than yourself

So if you like this plugin and use it regularly, consider donating to it. Even a small amount can make a big difference. Donating to this project is a way of saying "thank you" to me and helping me create more amazing games and project for everyone.

You can donate through GitHub Sponsors here: https://github.com/sponsors/hfjooste

3. Special Thanks

This project uses a few assets from third parties:

- Generic Graph by jinyuliao
- Comic Helvetic Font by Alexander Pravdin
- Menu Sound Effects by Broumbroum
- Land & Jump Sound Effects by Felixyadomi
- Item Sound Effects by Scrampunk
- Music by InAudio

If you liked these assets and want to see more, please follow the links and show them some love!

4. Console Codenames

4.1 Code names

Due to copyright issues, the plugin is not allowed to mention Xbox, Playstation or Switch anywhere in the code (including file names). To get around this issue, the consoles are refered to by the following code names:

Console	Code name
Xbox	Console MX
Playstation	Console SP
Switch	Console NS

4.2 Obfuscated code

The plugin will also use the console names to determine the platform. To avoid any copyright issues, the application will instead use an obfuscated string to determine the platform. The obfuscation and deobfuscation code can be found in `PlatformUtils.cpp`

Console	Obfuscated Text	Deobfuscated Text
Xbox	PT1BZWk5R2U=	xbox
Playstation	PUEzYw== PT13YzBGR2RwOW1i	ps station
Switch	PTRXYXVSWFp1UjJi emRYYTBOR2E=	nintendoswitch

5. Core Functionality

5.1 Input Devices

5.1.1 Introduction

Supported input devices. This is used to update the input indicators when using different input devices

5.1.2 Values

Value	Description
KeyboardMouse	Using a keyboard and mouse
GenericController	Using a controller on a desktop build
MxController	Using an Console MX controller
SpController	Using a Console SP controller
NsController	Using a Console NS controller
Unknown	Unknown device (used before initializing the input indicators)

5.2 Game Instance

5.2.1 Introduction

A base game instance with support for saving and loading game data using multiple save slots

5.2.2 Dependencies

The `USKGameInstance` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

5.2.3 Using the Game Instance

You need to create a blueprint using the `USKGameInstance_Implementation` as a parent before using the game instance. The input indicators feature is already configured if you use this base class. If you prefer to set this up manually, you can use `USKGameInstance` instead. After creating your own game instance blueprint, set this as the default game instance:

1. Open the Project Settings
2. Go to Project > Maps & Modes
3. Change the `Game Instance Class` value to your own blueprint

5.2.4 Save Data

You need to create a `USK Save Game` object before you can save/load data. This object contains all the data that you want to save. Just add the data you want to save as variables to the blueprint. The `Game Instance` will handle the rest. You also need to set the following properties before you can save/load data:

- **Save Game Class:** A reference to the `USK Save Game` class that contains the data you want to save

***NB:** You are required to set the save slot before you can save/load data. If not, you will get a `nullptr` and might cause your game to crash*

5.2.5 Input Indicators

The Game Instance will automatically detect input events and update the current input device if needed. If the input device is changed, other classes will be notified through the `OnInputDeviceUpdated` event

5.2.6 API Reference

Properties

Property	Description	Type	Default Value
LogConfigEditor	The log configuration used when running the game in the editor	ULogConfig*	`nullptr`
LogConfigDebug	The log configuration used by debug builds	ULogConfig*	`nullptr`
LogConfigDevelopment	The log configuration used by development builds	ULogConfig*	`nullptr`
LogConfigShipping	The log configuration used by shipping builds	ULogConfig*	`nullptr`
SaveGameClass	The class that holds the data that should be saved/ loaded	TSubclassOf<UUSKSaveGame>	
SettingsConfig	The configuration for the settings	USettingsConfig*	`nullptr`
IsInputIndicatorsEnabled	Is the input indicators feature enabled?	bool	true
InputMappingContext	The input mapping context used to extract the keys based on specific input actions	UInputMappingContext*	`nullptr`
KeyboardMouseInputMappings	A map of all keyboard/mouse keys and the texture displayed in the indicator	TMap<FKey, UTexture2D*>	
GenericControllerInputMappings	A map of all generic controller keys and the texture displayed in the indicator	TMap<FKey, UTexture2D*>	
MxControllerInputMappings	A map of all Console MX controller keys and the texture displayed in the indicator	TMap<FKey, UTexture2D*>	
SpControllerInputMappings	A map of all Console SP controller keys and the texture displayed in the indicator	TMap<FKey, UTexture2D*>	
NsControllerInputMappings	A map of all Console NS controller keys and the texture displayed in the indicator	TMap<FKey, UTexture2D*>	

Events

Name	Description	Params
OnDataLoadedEvent	Event used to notify other classes when the save data is loaded	
OnInputDeviceUpdated	Event used to notify other classes when the current input device is updated	
OnGamePaused	Event used to notify other classes when the game is paused	
OnGameUnpaused	Event used to notify other classes when the game is unpaused	

Functions

Name	Description	Params	Return
GetSaveData	Get the save data that is currently loaded		**UUSKSaveGame*** A reference to the current save data
SaveData	Save the modified data currently in memory		
SetCurrentSaveSlot	Set the current save slot	**Index (int)** The index of the save slot	
IsSaveSlotUsed	Check if a save slot is used	**Index (int)** The index of the save slot to check	**bool** A boolean value indicating if the save slot is used
EnableInputIndicators	Enable the input indicators feature		
DisableInputIndicators	Disable the input indicators feature		
GetInputIndicatorIcon	Get the input indicator icon for a specific action	**InputAction (UInputAction*)** The input action **Amount (int)** The amount of icons to retrieve	**TArray<UTexture2D*>** An array of input indicator icons for the specified action
GetInputIndicatorIconForKey	Get the input indicator icon for a specific key	**Key (FKey)** The key used to retrieve the input indicator icon **InputDevice (EInputDevice)** The input device used to retrieve the input indicator icon	**UTexture2D*** The input indicator icon for the specified key
GetKeyForInputAction	Get the key used by a specific input action	**Context (UInputMappingContext*)** The input mapping context **InputAction (UInputAction*)** The input action **MappableName (FName)** The player mappable name for the action	**FKey** The key used by the specified input action
UpdateKeyBindings	Update the key bindings that was changed by the player		
PauseGame	Pause the game		
UnpauseGame	Unpause the game		

5.2.7 Blueprint Usage

You can use the `USKGameInstance` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Save Data > Get Save Data
- Ultimate Starter Kit > Save Data > Save Data
- Ultimate Starter Kit > Save Data > Set Current Save Slot
- Ultimate Starter Kit > Save Data > Is Save Slot Used
- Ultimate Starter Kit > Input > Enable Input Indicators
- Ultimate Starter Kit > Input > Disable Input Indicators
- Ultimate Starter Kit > Input > Get Input Indicator Icon
- Ultimate Starter Kit > Input > Get Input Indicator Icon For Key
- Ultimate Starter Kit > Input > Get Key For Input Action
- Ultimate Starter Kit > Input > Update Key Bindings
- Ultimate Starter Kit > Pause > Pause Game
- Ultimate Starter Kit > Pause > Unpause Game

5.2.8 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `USKGameInstance` can now be used in any of your C++ files:

```
#include "USK/Core/USKGameInstance.h"

void ATestActor::Test()
{
    // USKGameInstance is a pointer to the UUSKGameInstance
    UUSKSaveGame* SaveData = USKGameInstance->GetSaveData();
    USKGameInstance->SaveData();
    USKGameInstance->SetCurrentSaveSlot(Index);
    bool IsSaveSlotUsedValue = USKGameInstance->IsSaveSlotUsed(Index);
    USKGameInstance->EnableInputIndicators();
    USKGameInstance->DisableInputIndicators();
    TArray<UTexture2D*> InputIndicatorIcon = USKGameInstance->GetInputIndicatorIcon(InputAction, Amount);
    UTexture2D* InputIndicatorIconForKey = USKGameInstance->GetInputIndicatorIconForKey(Key, InputDevice);
    FKey KeyForInputAction = USKGameInstance->GetKeyForInputAction(Context, InputAction, MappableName);
    USKGameInstance->UpdateKeyBindings();
    USKGameInstance->PauseGame();
    USKGameInstance->UnpauseGame();
}
```

6. Logging System

6.1 Logger

6.1.1 Introduction

A system used to easily log info to file and via on-screen messages

6.1.2 Log Levels

This plugin supports the following log types:

1. **Trace:** Logs that contain the most detailed messages. These messages may contain sensitive application data. These messages are disabled by default and should never be enabled in a production environment
2. **Debug:** Logs that are used for interactive investigation during development. These logs should primarily contain information useful for debugging and have no long-term value
3. **Information:** Logs that track the general flow of the application. These logs should have long-term value
4. **Warning:** Logs that highlight an abnormal or unexpected event in the application flow, but do not otherwise cause the application execution to stop
5. **Error:** Logs that highlight when the current flow of execution is stopped due to a failure. These should indicate a failure in the current activity, not an application-wide failure

The log levels corresponds to the following verbosity level in Unreal Engine:

Log Level	Log Verbosity
Trace	VeryVerbose
Debug	Verbose
Information	Display
Warning	Warning
Error	Error

6.1.3 API Reference

Functions

Name	Description	Params	Return
Configure	Configure the logger	**Config (ULogConfig*)** The new config file used by the logger	
Error	Log an error	**Tag (FString)** The category of the log entry **Text (FString)** The text to log out	
Warning	Log a warning	**Tag (FString)** The category of the log entry **Text (FString)** The text to log out	
Info	Log info	**Tag (FString)** The category of the log entry **Text (FString)** The text to log out	
Debug	Log debug information	**Tag (FString)** The category of the log entry **Text (FString)** The text to log out	
Trace	Log trace information	**Tag (FString)** The category of the log entry **Text (FString)** The text to log out	

6.1.4 Blueprint Usage

You can easily log info using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Logger > Configure
- Ultimate Starter Kit > Logger > Log Trace
- Ultimate Starter Kit > Logger > Log Debug
- Ultimate Starter Kit > Logger > Log Info
- Ultimate Starter Kit > Logger > Log Warning
- Ultimate Starter Kit > Logger > Log Error

6.1.5 C++ Usage

The logging is handled through a static class/functions. You first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The logger can now be used in any of your C++ files:

```
#include "USK/Logger/Log.h"
```

```
void ATestActor::Test()
{
    USK_LOG_TRACE("Testing trace logging");
    USK_LOG_DEBUG("Testing debug logging");
    USK_LOG_INFO("Testing info logging");
    USK_LOG_WARNING("Testing warning logging");
    USK_LOG_ERROR("Testing error logging");

    ULog::Configure(Config);
    ULog::Trace("Custom Tag", "Testing trace logging");
    ULog::Debug("Custom Tag", "Testing debug logging");
    ULog::Info("Custom Tag", "Testing info logging");
    ULog::Warning("Custom Tag", "Testing warning logging");
    ULog::Error("Custom Tag", "Testing error logging");
}
```

6.2 Log Configuration

6.2.1 Introduction

Config file for the logger

6.2.2 API Reference

Properties

Property	Description	Type	Default Value
bErrorWriteToFile	Should error logs be written to file?	bool	true
bErrorPrintToScreen	Should error logs be printed to screen?	bool	true
bWarningWriteToFile	Should warning logs be written to file?	bool	true
bWarningPrintToScreen	Should warning logs be printed to screen?	bool	true
bInfoWriteToFile	Should info logs be written to file?	bool	true
bInfoPrintToScreen	Should info logs be printed to screen?	bool	true
bDebugWriteToFile	Should debug logs be written to file?	bool	true
bDebugPrintToScreen	Should debug logs be printed to screen?	bool	true
bTraceWriteToFile	Should debug logs be written to file?	bool	true
bTracePrintToScreen	Should debug logs be printed to screen?	bool	true

7. Dialogue System

7.1 Dialogue System

7.1.1 Introduction

The Ultimate Starter Kit plugin contains a full dialogue system with support for different participants, choices and brances

7.1.2 Creating a new dialogue

You can create a new dialogue by following these steps: 1. Right click in the Content Browser 2. Go to the Ultimate Starter Kit section 3. Click on Dialogue

This will open the newly created dialogue in the dialogue editor

7.1.3 Using the editor

The dialogue editor works similar to a Behaviour Tree. You start by creating a single root node. You can then drag connections to a new dialogue entry. The game will start off with the root and follow the path automatically until it reaches the end or a choice. If a choice is encountered, it will present the options to the player and follow the correct branch after a choice is selected

7.2 Dialogue

7.2.1 Introduction

An array of all the root entries contained by this dialogue

7.2.2 API Reference

Properties

Property	Description	Type	Default Value
RootEntries	An array of all the root entries contained by this dialogue	TArray<UDialogueEntry*>	
AllEntries	An array of all the entries contained by this dialogue	TArray<UDialogueEntry*>	
EditorGraph	A reference to the editor graph used by this dialogue	UEdGraph*	`nullptr`

7.3 Participant

7.3.1 Introduction

A participant in the dialogue

7.3.2 API Reference

Properties

Property	Description	Type	Default Value
Name	The name of the dialogue participant	FText	
Color	The color used to represent the dialogue participant	FLinearColor	FLinearColor::Black

7.4 Entry

7.4.1 Introduction

A single entry in a dialogue

7.4.2 API Reference

Properties

Property	Description	Type	Default Value
Owner	The owner participant of the dialogue entry	UDialogueParticipant*	`nullptr`
Transition	The type of transition for this dialogue entry	EDialogueTransitionType	
Id	The ID of the dialogue entry	FName	FNam
Text	The text to display	FText	
Speed	The speed of the dialogue	float	12.5f
Audio	The audio to play with this dialogue entry	USoundBase*	`nullptr`
Dialogue	A reference to the dialogue containing this entry	UDialogue*	`nullptr`
ParentNodes	An array of all the parent entries for this dialogue entry	TArray<UDialogueEntry*>	
ChildrenNodes	An array of all the child entries for this dialogue entry	TArray<UDialogueEntry*>	
Edges	A map of all the possible edges for this dialogue entry	TMap<UDialogueEntry*, UDialogueTransition*>	

Functions

Name	Description	Params	Return
IsLeafNode	Check if this entry is a leaf node		**bool** A boolean value indicating if this entry is a leaf node
GetTitle	Get the title displayed for this dialogue entry		**FText** The title displayed for this dialogue entry
GetText	Get the text displayed for this dialogue entry		**FText** The text displayed for this dialogue entry

7.4.3 Blueprint Usage

You can use the `DialogueEntry` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Dialogue > Entry > Is Leaf Node
- Ultimate Starter Kit > Dialogue > Entry > Get Title
- Ultimate Starter Kit > Dialogue > Entry > Get Text

7.4.4 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `DialogueEntry` can now be used in any of your C++ files:

```
#include "USK/Dialogue/DialogueEntry.h"

void ATestActor::Test()
{
    // DialogueEntry is a pointer to the UDialogueEntry
    bool IsLeafNodeValue = DialogueEntry->IsLeafNode();
    FText Title = DialogueEntry->GetTitle();
    FText Text = DialogueEntry->GetText();
}
```

7.5 Transition

7.5.1 Transition Type

Introduction

The type of transition used by the dialogue entry

Values

Value	Description
Auto	Automatically advance to the next dialogue entry
Choice	Give the player a few choices used to create branches in the dialogue

7.5.2 Transition Data

Introduction

A transition from one dialogue entry to the next

API Reference

PROPERTIES

Property	Description	Type	Default Value
Text	The text displayed for this transition (choice)	FText	
StartEntry	The dialogue entry where the transition starts	UDialogueEntry*	`nullptr`
EndEntry	The dialogue entry where the transition ends	UDialogueEntry*	`nullptr`

EVENTS

Name	Description	Params
OnMarkedForDeletion	Event used to notify other classes that the transition is marked for deletion	

7.6 Manager

7.6.1 Introduction

The pawn responsible for managing the dialogue

7.6.2 Dependencies

The `DialogueManager` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Audio: Used to play sound effects either 2D or at a specified location

7.6.3 Components

The `DialogueManager` uses the following components:

Name	Description	Type
AudioComponent	The audio component responsible for playing the audio files of the dialogue entries	UAudioComponent*

7.6.4 API Reference

Properties

Property	Description	Type	Default Value
Dialogue	The dialogue that should be played by the dialogue manager	UDialogue*	`nullptr`
DialogueWidgetClass	The class of the widget used to display the dialogue	TSubclassOf<UDialogueWidget>	
PlayOnStart	A boolean value indicating if the dialogue should automatically play when the level is started	bool	true
DestroyOnComplete	A boolean value indicating if the dialogue should automatically be destroyed when completed	bool	true
StopOnComplete	A boolean value indicating if the dialogue should automatically be stopped when completed	bool	true
SkipSFX	The sound effect to play when an entry is skipped	USoundBase*	`nullptr`
AdvanceSFX	The sound effect to play when advancing to the next entry	USoundBase*	`nullptr`
InputMappingContext	The input mapping context used to interact with the dialogue	UInputMappingContext*	`nullptr`
SkipAction	The input action used to skip the current dialogue entry	UInputAction*	`nullptr`

Events

Name	Description	Params
OnDialogueEnded	Event used to notify other classes when the dialogue has ended	**LastEntryId (FName)** The ID of the last entry in the dialogue
OnDialogueEntryStarted	Event used to notify other classes when a dialogue entry has started	**LastEntryId (FName)** The ID of the dialogue entry
OnDialogueEntryEnded	Event used to notify other classes when a dialogue entry has ended	**LastEntryId (FName)** The ID of the dialogue entry

Functions

Name	Description	Params	Return
PlayDialogue	Play the dialogue		
StopDialogue	Stop playing the dialogue		
DestroyDialogue	Stop playing the dialogue and destroy the dialogue manager		
SkipEntry	Skip the current entry in the dialogue		

7.6.5 Blueprint Usage

You can use the `DialogueManager` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Dialogue > Play Dialogue
- Ultimate Starter Kit > Dialogue > Stop Dialogue
- Ultimate Starter Kit > Dialogue > Destroy Dialogue
- Ultimate Starter Kit > Dialogue > Skip Entry

7.6.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `DialogueManager` can now be used in any of your C++ files:

```
#include "USK/Dialogue/DialogueManager.h"

void ATestActor::Test()
{
    // DialogueManager is a pointer to the ADialogueManager
    DialogueManager->PlayDialogue();
    DialogueManager->StopDialogue();
    DialogueManager->DestroyDialogue();
    DialogueManager->SkipEntry();
}
```

7.7 Widget

7.7.1 Introduction

Widget used to display a dialogue

7.7.2 Dependencies

The `DialogueTransitionType` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

7.7.3 Required Widgets

You need to add the following before you can compile the `DialogueTransitionType` widget:

Name	Description	Type
DialogueTitle	The text block used to display the dialogue title	UTextBlock*
DialogueText	The text block used to display the dialogue text	UTextBlock*
ChoiceMenu	The menu used to display the dialogue choices	UMenu*

7.7.4 Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
SkipEntryImage	The image displayed when the dialogue entry is completed	UImage*

7.7.5 API Reference

Properties

Property	Description	Type	Default Value
ChoiceMenuItemClass	The class for all choice menu items	TSubclassOf<UMenuItem>	

Events

Name	Description	Params
OnChoiceSelected	Event used to notify other classes when a choice is selected	**Index (int)** The index of the choice that was selected

8. Inventory

8.1 Inventory Component

8.1.1 Introduction

Actor component responsible for tracking data in an inventory

8.1.2 Dependencies

The `InventoryComponent` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data

8.1.3 API Reference

Properties

Property	Description	Type	Default Value
InventoryId	The ID of the inventory used when saving/loading the data	FName	
AutoSave	Should the data in the inventory automatically be saved/loaded	bool	true
EnforceMaxAmount	Should a maximum amount be enforced for each item?	bool	
MaxAmount	The maximum amount of each item	int	99

Events

Name	Description	Params
OnInventoryItemUpdated	Event used to notify other classes every time an item in the inventory was updated	**Id (FName)** The ID of the item that was updated **Amount (FName)** The new amount of the item

Functions

Name	Description	Params	Return
GetItems	Get all the item currently in the inventory		**TArray<FInventoryItem>** An array of all the items in the inventory
AddItem	Add an item to the inventory	**Id (FName)** The ID of the item to add **Amount (int)** The amount to add	
RemoveItem	Remove an item from the inventory	**Id (FName)** The ID of the item to remove **Amount (int)** The amount to remove	
RemoveAll	Remove all the items with the specified ID	**Id (FName)** The ID of the item to remove	
Clear	Remove all items from the inventory		
LoadInventory	Load the inventory data		
SaveInventory	Save the inventory data		

8.1.4 Blueprint Usage

You can use the `InventoryComponent` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Inventory > Get Items
- Ultimate Starter Kit > Inventory > Add Item
- Ultimate Starter Kit > Inventory > Remove Item
- Ultimate Starter Kit > Inventory > Remove All
- Ultimate Starter Kit > Inventory > Clear
- Ultimate Starter Kit > Inventory > Load Inventory
- Ultimate Starter Kit > Inventory > Save Inventory

8.1.5 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `InventoryComponent` can now be used in any of your C++ files:

```
#include "USK/Inventory/InventoryComponent.h"

void ATestActor::Test()
{
    // InventoryComponent is a pointer to the UInventoryComponent
    TArray<FInventoryItem> Items = InventoryComponent->GetItems();
    InventoryComponent->AddItem(Id, Amount);
    InventoryComponent->RemoveItem(Id, Amount);
    InventoryComponent->RemoveAll(Id);
    InventoryComponent->Clear();
    InventoryComponent->LoadInventory();
    InventoryComponent->SaveInventory();
}
```

8.2 Inventory Data

8.2.1 Introduction

The inventory data for a single inventory

8.2.2 Properties

Property	Description	Type	Default Value
Items	The array of items in the inventory	TArray<FInventoryItem>	

8.3 Inventory Item

8.3.1 Introduction

The information about a specific inventory item

8.3.2 Properties

Property	Description	Type	Default Value
Id	The ID of the inventory item	FName	
Amount	The amount of the inventory item	int	

8.4 Inventory Item Data

8.4.1 Introduction

The data used to describe an inventory item

8.4.2 Properties

Property	Description	Type	Default Value
Name	The name of the inventory item	FText	
Description	The description of the inventory item	FText	
InventoryImage	The image displayed in the inventory	UTexture2D*	`nullptr`
PreviewImage	The preview image displayed when the item is highlighted	UTexture2D*	`nullptr`

8.5 Inventory Menu Item

8.5.1 Introduction

The menu item used to display an inventory item

8.5.2 Dependencies

The `InventoryMenuItem` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

8.5.3 Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
InventoryImage	The image of the inventory item	UImage*

8.5.4 API Reference

Properties

Property	Description	Type	Default Value

Functions

Name	Description	Params	Return
InitializeEmptyInventoryItem	Initialize an empty inventory item	**Widget (UInventoryWidget*)** The widget that owns this menu item	
InitializeInventoryItem	Initialize a non-empty inventory item	**Widget (UInventoryWidget*)** The widget that owns this menu item **Item (FInventoryItem)** The item tracked by this menu item **Data (FInventoryItemData)** The data for this menu item	
UpdateInventoryGridPosition	Update the grid position of the inventory menu item	**CurrentColumn (int)** The current column of the inventory menu item **CurrentRow (int)** The current row of the inventory menu item	
GetInventoryItem	Get the inventory item tracked by this menu item		**FInventoryItem** The inventory item tracked by this menu item
UpdateAmount	Update the amount of the inventory item	**Amount (int)** The new amount of the inventory item	

8.5.5 Blueprint Usage

You can use the `InventoryMenuItem` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Inventory > Initialize Empty Inventory Item
- Ultimate Starter Kit > Inventory > Initialize Inventory Item
- Ultimate Starter Kit > Inventory > Update Inventory Grid Position
- Ultimate Starter Kit > Inventory > Get Inventory Item
- Ultimate Starter Kit > Inventory > Update Amount

8.5.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `InventoryMenuItem` can now be used in any of your C++ files:

```
#include "USK/Inventory/InventoryMenuItem.h"

void ATestActor::Test()
{
    // InventoryMenuItem is a pointer to the UInventoryMenuItem
    InventoryMenuItem->InitializeEmptyInventoryItem(Widget);
    InventoryMenuItem->InitializeInventoryItem(Widget, Item, Data);
    InventoryMenuItem->UpdateInventoryGridPosition(CurrentColumn, CurrentRow);
    FInventoryItem InventoryItem = InventoryMenuItem->GetInventoryItem();
    InventoryMenuItem->UpdateAmount(Amount);
}
```

8.6 Inventory Size

8.6.1 Introduction

The size restrictions of the inventory

8.6.2 Values

Value	Description
FixedSize	Limit both the width and height of the inventory
FixedRows	Limit only the rows of the inventory and allow the columns to increase
FixedColumns	Limit only the columns of the inventory and allow the rows to increase

8.7 Inventory Widget

8.7.1 Introduction

The widget responsible for displaying the inventory

8.7.2 Dependencies

The `InventoryWidget` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

8.7.3 Required Widgets

You need to add the following before you can compile the `InventoryWidget` widget:

Name	Description	Type
InventoryMenu	The menu responsible for controlling all the menu items	UMenu*

8.7.4 Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
NameText	The widget responsible for displaying the highlighted item's name	UTextBlock*
DescriptionText	The widget responsible for displaying the highlighted item's description	UTextBlock*
PreviewImage	The widget responsible for displaying the highlighted item's image	UImage*
AmountText	The widget responsible for displaying the highlighted item's amount	UTextBlock*

8.7.5 API Reference

Properties

Property	Description	Type	Default Value
MenuItemClass	The inventory menu item class used to display the inventory items	TSubclassOf<UInventoryMenuItem>	
ItemData	The data table containing all the information about the inventory items	UDataTable*	`nullptr`
InventorySize	The size restrictions of the inventory	EInventorySize	
Rows	The amount of rows in the inventory	int	
Columns	The amount of columns in the inventory	int	

Events

Name	Description	Params
OnInventoryItemSelected	Event used to notify other classes every time an inventory item is selected	**Name (FName)** The ID of the selected inventory item

Functions

Name	Description	Params	Return
LoadInventory	Load a specific inventory	**InventoryComponent (UInventoryComponent*)** The inventory to load	
UpdatePreview	Update the preview of the inventory	**Item (FInventoryItem)** The item to preview	
UpdateHighlightedIndex	Update the highlighted index	**Column (int)** The column index of the item that is highlighted **Row (int)** The row index of the item that is highlighted	
SelectItem	Select an inventory item	**Id (FName)** The ID of the item to select	
GetInventory	Get the inventory managed by the widget		**UInventoryComponent*** The inventory managed by the widget
RefreshItem	Refresh a specific item in the inventory	**Id (FName)** The ID of the item to refresh **Amount (int)** The amount of the item	
RefreshInventory	Refresh the entire inventory		

8.7.6 Blueprint Usage

You can use the `InventoryWidget` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Inventory > Load Inventory
- Ultimate Starter Kit > Inventory > Update Preview
- Ultimate Starter Kit > Inventory > Update Highlighted Index
- Ultimate Starter Kit > Inventory > Select Item
- Ultimate Starter Kit > Inventory > Get Inventory
- Ultimate Starter Kit > Inventory > Refresh Item
- Ultimate Starter Kit > Inventory > Refresh Inventory

8.7.7 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `InventoryWidget` can now be used in any of your C++ files:

```
#include "USK/Inventory/InventoryWidget.h"

void ATestActor::Test()
{
    // InventoryWidget is a pointer to the UInventoryWidget
    InventoryWidget->LoadInventory(InventoryComponent);
    InventoryWidget->UpdatePreview(Item);
    InventoryWidget->UpdateHighlightedIndex(Column, Row);
```

```
    InventoryWidget->SelectItem(Id);
    UInventoryComponent* Inventory = InventoryWidget->GetInventory();
    InventoryWidget->RefreshItem(Id, Amount);
    InventoryWidget->RefreshInventory();
}
```

9. Trackable Data

9.1 Overview

9.1.1 Introduction

A system that is used to easily manage different types of actor data

9.1.2 Trackable Data Component

Before you can manage the data, you need to create a Trackable Data Component and add it to the actor/character containing the data

9.1.3 Built-in data

The following data can automatically be managed without creating custom components:

1. Currency (using the `Currency Component`)
2. Stats (using the `Stats Component`)

9.2 Data

9.2.1 Introduction

All trackable data use the FTrackableData struct to specify the default values and behaviours

9.2.2 Properties

Property	Description	Type	Default Value
InitialValue	The initial value of the data	float	
EnforceMaxValue	Should we enforce a maximum value?	bool	
MaxValue	The maximum value of the data	float	100.0f
AutoSave	Should all value updates automatically be saved using the game instance?	bool	
AutoGenerate	Should we automatically generate value every second?	bool	
GenerateAmount	The amount of value to generate every second	float	
GenerateDelay	The delay before the value starts generating after losing value	float	

9.3 Component

9.3.1 Introduction

A component that is used to easily manage/track different types of actor data

9.3.2 Dependencies

The `TrackableDataComponent` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data

9.3.3 Data

The data to track is configured by adding items to the `Data` map. The component should be added to the actor/character containing the data

9.3.4 API Reference

Properties

Property	Description	Type	Default Value
Data	The map of data to track	TMap<FName, FTrackableData>	

Events

Name	Description	Params
OnValueZero	Event used to notify other classes every time the data value reaches 0	**Name (FName)** The name of the data item
OnValueUpdated	Event used to notify other classes every time the data value is updated	**Name (FName)** The name of the data item **Value (FName)** The current value of the data item **ValuePercentage (FName)** The percentage of the current value compared to the max value of the data item

Functions

Name	Description	Params	Return
GetValue	Get the amount of the data	**Name (FName)** The name of the data item	**float** The current amount of the data item
GetValuePercentage	Get the value of the data as a percentage of to the max value	**Name (FName)** The name of the data item	**float** The value of the data as a percentage of to the max value
Add	Add an amount to the data	**Name (FName)** The name of the data item **Amount (float)** The amount to add	**float** The new amount of the data item
Remove	Remove an amount from the data	**Name (FName)** The name of the data item **Amount (float)** The amount to remove	**float** The new amount of the data item

9.3.5 Blueprint Usage

You can use the `TrackableDataComponent` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Trackable Data > Get Value
- Ultimate Starter Kit > Trackable Data > Get Value Percentage
- Ultimate Starter Kit > Trackable Data > Add
- Ultimate Starter Kit > Trackable Data > Remove

9.3.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `TrackableDataComponent` can now be used in any of your C++ files:

```
#include "USK/Data/TrackableDataComponent.h"

void ATestActor::Test()
{
    // TrackableDataComponent is a pointer to the UTrackableDataComponent
    float Value = TrackableDataComponent->GetValue(Name);
    float ValuePercentage = TrackableDataComponent->GetValuePercentage(Name);
    float AddValue = TrackableDataComponent->Add(Name, Amount);
    float RemoveValue = TrackableDataComponent->Remove(Name, Amount);
}
```

10. Audio

10.1 Audio Overview

10.1.1 Introduction

A system used to manage the basic properties of audio files. It includes different sound classes, a sound mix and sound attenuation settings

10.1.2 Sound Classes

The audio system includes a few basic preconfigured sound classes:

Class name	Group	Volume
USK_EffectsSoundClass	Effects	1.0
USK_MusicSoundClass	Music	0.5
USK_UISoundClass	UI	1.0
USK_VoiceSoundClass	Voice	3.0

10.2 Audio Utils

10.2.1 Introduction

The audio utils class is used to easily play sound effects

10.2.2 Dependencies

The `AudioUtils` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

10.2.3 API Reference

Functions

Name	Description	Params	Return
PlaySound2D	Play a 2D sound	**WorldContext (UObject*)** The top level object representing a map **SoundFX (USoundBase*)** The USoundBase to play	
PlayRandomSound2D	Play a random 2D sound	**WorldContext (UObject*)** The top level object representing a map **SoundFX (TArray)** The array of USoundBase to select the random sound from	
PlaySound	Play a sound at the specified actor's location	**Actor (AActor*)** The actor where the sound will be played **SoundFX (USoundBase*)** The USoundBase to play	
PlayRandomSound	Play a random sound at the specified actor's location	**Actor (AActor*)** The actor where the sound will be played **SoundFX (TArray)** The array of USoundBase to select the random sound from	

10.2.4 Blueprint Usage

You can use the `AudioUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Audio > Play Sound2D
- Ultimate Starter Kit > Audio > Play Random Sound2D
- Ultimate Starter Kit > Audio > Play Sound
- Ultimate Starter Kit > Audio > Play Random Sound

10.2.5 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `AudioUtils` can now be used in any of your C++ files:

```
#include "USK/Audio/AudioUtils.h"

void ATestActor::Test()
{
    UAudioUtils::PlaySound2D(WorldContext, SoundFX);
    UAudioUtils::PlayRandomSound2D(WorldContext, SoundFX);
    UAudioUtils::PlaySound(Actor, SoundFX);
    UAudioUtils::PlayRandomSound(Actor, SoundFX);
}
```

10.3 Music Player

10.3.1 Introduction

Actor responsible for playing, pausing and stopping music. It also allows you to adjust music volume

10.3.2 Dependencies

The `inal` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

10.3.3 Components

The `inal` uses the following components:

Name	Description	Type
AudioPlayer	Actor responsible for playing, pausing and stopping music. It also allows you to adjust music volume	UAudioComponent*

10.3.4 API Reference

Properties

Property	Description	Type	Default Value
PlayOnStart	Should the music automatically play when the actor is spawned?	bool	true

Functions

Name	Description	Params	Return
SetVolume	Adjust the playback volume of the music	**Volume (float)** The new volume of the music	
Play	Play the music		
Pause	Pause the music		
Stop	Stop the music		

10.3.5 Blueprint Usage

You can use the `inal` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Audio > Set Volume
- Ultimate Starter Kit > Audio > Play
- Ultimate Starter Kit > Audio > Pause
- Ultimate Starter Kit > Audio > Stop

10.3.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `inal` can now be used in any of your C++ files:

```
#include "USK/Audio/MusicPlayer.h"

void ATestActor::Test()
{
    // inal is a pointer to the final
    inal->SetVolume(Volume);
    inal->Play();
    inal->Pause();
    inal->Stop();
}
```

11. Characters

11.1 Overview

11.1.1 Introduction

The plugin includes a basic 3D platformer character and animation template. This can easily be extended to add unique functionality

11.2 Base Character

11.2.1 Introduction

Base character class

11.2.2 Dependencies

The `USKCharacter` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Audio: Used to play sound effects either 2D or at a specified location

11.2.3 Components

The `USKCharacter` uses the following components:

Name	Description	Type
CameraComponent	Base character class	UCameraComponent*

11.2.4 API Reference

Properties

Property	Description	Type	Default Value
IsDoubleJumping	Is the character double jumping?	bool	
InputMappingContext	The input mapping context used by the player	UInputMappingContext*	`nullptr`
MoveAction	The move input action	UInputAction*	`nullptr`
LookAroundAction	The camera rotation input action	UInputAction*	`nullptr`
JumpAction	The jump input action	UInputAction*	`nullptr`
FireWeaponAction	The fire weapon input action	UInputAction*	`nullptr`
ShadowDecalClass	The shadow decal class used to draw a shadow below the character while in the air	TSubclassOf<AShadowDecal>	
JumpSoundEffects	An array of sound effects played when jumping	TArray<USoundBase*>	
JumpParticleFx	The particle effects spawned when jumping	UNiagaraSystem*	`nullptr`
JumpParticleFxSpawnOffset	The offset applied to the location of the jump particles when spawning	FVector	
LandedSoundEffects	An array of sound effects played when landing	TArray<USoundBase*>	
LandParticleFx	The particle effects spawned when landing	UNiagaraSystem*	`nullptr`
LandParticleFxSpawnOffset	The offset applied to the location of the land particles when spawning	FVector	
VariableJumpHeight	Does the character support variable jump height?	bool	true
VariableJumpHeightMaxHoldTime	The amount of time to hold the jump button to reach the max height	float	0.3f
JumpVelocity	The velocity applied to the character when jumping	float	500.0f
AirControl	The amount of lateral movement control available to the character while in the air	float	1000.0f
FallingFriction	The amount of friction to apply to lateral air movement when falling	float	3.5f
Gravity	The amount of gravity applied to the character	float	2.0f
CanDoubleJump	Can the character perform a double jump?	bool	true
CanCoyoteJump	Does the character support coyote time when trying to jump?	bool	true
CoyoteJumpTime	The amount of coyote time for the character	float	0.375f
CoyoteJumpVelocity	The velocity applied to the character when performing a coyote jump	float	700.0f
BrakingFriction	Friction coefficient applied when braking	float	10.0f
MaxAcceleration	The rate of change of velocity	float	2500.0f
DefaultWeaponClass	The default weapon the character will equip on spawn	TSubclassOf<AWeapon>	

Functions

Name	Description	Params	Return
GetCameraComponent	Get the camera used by the character		**UCameraComponent*** The camera used by the character
SetWeapon	Set the current weapon used by the character	**NewWeapon (AWeapon*)** The new weapon	
GetWeapon	Get the current weapon used by the character		**AWeapon*** The current weapon used by the character
StartFiringWeapon	Start firing the current weapon		
StopFiringWeapon	Stop firing the current weapon		

11.2.5 Blueprint Usage

There is no additional functions exposed to Blueprints. Just create the character and add it to your level

11.3 FPS Character

11.3.1 Introduction

Base character that can be used for first person shooter (FPS) games

11.3.2 Features

The following features are included in the FPS character class:

1. **Double Jumping**: Additional jump with a different jump animation
2. **Variable Jump Height**: Adjust jump height based on how long the jump button is pressed
3. **Coyote Time**: Allow the character to jump for a short time after falling off the platform
4. **Shadow Decal**: A decal used as a shadow to indicate where the character will land
5. **Jump & Land effects**: Sound effects and particles when jumping and landing

All these features can be configured to meet your needs and can also be disabled

11.3.3 API Reference

11.3.4 Blueprint Usage

There is no additional functions exposed to Blueprints. Just create the character and add it to your level

11.4 3D Platformer Character

11.4.1 Introduction

Base character that can be used for 3D platformer games

11.4.2 Dependencies

The `PlatformerCharacter` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

11.4.3 Features

The following features are included in the 3D platformer character class:

1. **Double Jumping**: Additional jump with a different jump animation
2. **Variable Jump Height**: Adjust jump height based on how long the jump button is pressed
3. **Coyote Time**: Allow the character to jump for a short time after falling off the platform
4. **Shadow Decal**: A decal used as a shadow to indicate where the character will land
5. **Adjustable Camera Distance**: The camera automatically zooms in on the character while idle and zooms out as soon as the character starts moving
6. **Jump & Land effects**: Sound effects and particles when jumping and landing

All these features can be configured to meet your needs and can also be disabled

11.4.4 Components

The `PlatformerCharacter` uses the following components:

Name	Description	Type
SpringArmComponent	Base character that can be used for 3D platformer games	USpringArmComponent*

11.4.5 API Reference

Properties

Property	Description	Type	Default Value
TargetArmLength	Length of the spring arm component	float	350.0f
ArmLengthMultiplier	The multiplier applied to the spring arm component when the character is moving	float	0.4f
CameraAdjustmentSpeed	The speed used when adjusting the camera distance	float	3.0f

Functions

Name	Description	Params	Return
GetSpringArmComponent	Get the spring arm component of the character		**USpringArmComponent*** The spring arm component responsible for controlling the distance of the camera

11.4.6 Blueprint Usage

There is no additional functions exposed to Blueprints. Just create the character and add it to your level

11.5 Animation Instance

11.5.1 Introduction

Base animation instance for USK characters

11.5.2 Dependencies

The `USKCharacterAnimationInstance` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

11.5.3 Animation Montages

The animation blueprint contains a `USK` slot that can be used to play animation montages

11.5.4 API Reference

Properties

Property	Description	Type	Default Value
IdleAnimation	The animation used when the character is in the idle state while unarmed	UAnimSequence*	`nullptr`
WalkAnimation	The animation used when the character is walking while unarmed	UAnimSequence*	`nullptr`
RunAnimation	The animation used when the character is running while unarmed	UAnimSequence*	`nullptr`
JumpAnimation	The animation used when the character is jumping while unarmed	UAnimSequence*	`nullptr`
DoubleJumpAnimation	The animation used when the character is double jumping while unarmed	UAnimSequence*	`nullptr`
FallAnimation	The animation used when the character is falling while unarmed	UAnimSequence*	`nullptr`
LandAnimation	The animation used when the character is landing while unarmed	UAnimSequence*	`nullptr`
IdleWeaponOneHandedAnimation	The animation used when the character is in the idle state with a one handed weapon	UAnimSequence*	`nullptr`
WalkWeaponOneHandedAnimation	The animation used when the character is walking with a one handed weapon	UAnimSequence*	`nullptr`
RunWeaponOneHandedAnimation	The animation used when the character is running with a one handed weapon	UAnimSequence*	`nullptr`
JumpWeaponOneHandedAnimation	The animation used when the character is jumping with a one handed weapon	UAnimSequence*	`nullptr`
DoubleJumpWeaponOneHandedAnimation	The animation used when the character is double jumping with a one handed weapon	UAnimSequence*	`nullptr`
FallWeaponOneHandedAnimation	The animation used when the character is falling with a one handed weapon	UAnimSequence*	`nullptr`
LandOneHandedAnimation	The animation used when the character is landing with a one handed weapon	UAnimSequence*	`nullptr`
IdleWeaponTwoHandedAnimation	The animation used when the character is in the idle state with a two handed weapon	UAnimSequence*	`nullptr`
WalkWeaponTwoHandedAnimation	The animation used when the character is walking with a two handed weapon	UAnimSequence*	`nullptr`
RunWeaponTwoHandedAnimation	The animation used when the character is running with a two handed weapon	UAnimSequence*	`nullptr`
JumpWeaponTwoHandedAnimation	The animation used when the character is jumping with a two handed weapon	UAnimSequence*	`nullptr`
DoubleJumpWeaponTwoHandedAnimation	The animation used when the character is double jumping with a two handed weapon	UAnimSequence*	`nullptr`
FallWeaponTwoHandedAnimation	The animation used when the character is falling with a two handed weapon	UAnimSequence*	`nullptr`
LandWeaponTwoHandedAnimation	The animation used when the character is landing with a two handed weapon	UAnimSequence*	`nullptr`
MovementSpeed	The movement speed fo the character	float	

IsInAir	Is the character currently in the air?	bool	
IsDoubleJumping	Is the character double jumping?	bool	

Functions

Name	Description	Params	Return
GetIdleAnimation	Get the idle animation based on the current armed state		**UAnimSequence*** The idle animation to play
GetWalkAnimation	Get the walk animation based on the current armed state		**UAnimSequence*** The walk animation to play
GetRunAnimation	Get the run animation based on the current armed state		**UAnimSequence*** The run animation to play
GetJumpAnimation	Get the jump animation based on the current armed state		**UAnimSequence*** The jump animation to play
GetDoubleJumpAnimation	Get the double jump animation based on the current armed state		**UAnimSequence*** The double jump animation to play
GetFallAnimation	Get the fall animation based on the current armed state		**UAnimSequence*** The fall animation to play
GetLandAnimation	Get the land animation based on the current armed state		**UAnimSequence*** The land animation to play

11.5.5 Blueprint Usage

You can use this template by creating your own animation blueprint and selecting `UPlatformerAnimationInstance` as the parent class. Set your animations and use this for your 3D platformer character

11.6 Shadow Decal

11.6.1 Introduction

Decal used to draw a shadow beneath a character when the character is in the air

11.6.2 Dependencies

The `ShadowDecal` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

11.6.3 API Reference

Functions

Name	Description	Params	Return
Initialize	Initialize the shadow decal	**OwnerCharacter (ACharacter*)** The character owning this shadow decal	

11.6.4 Blueprint Usage

You can use the `ShadowDecal` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Shadow Decal > Initialize

11.6.5 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `ShadowDecal` can now be used in any of your C++ files:

```
#include "USK/Character/ShadowDecal.h"

void ATestActor::Test()
{
    // ShadowDecal is a pointer to the AShadowDecal
    ShadowDecal->Initialize(OwnerCharacter);
}
```

12. Weapons

12.1 Weapon

12.1.1 Introduction

The weapon attached to characters

12.1.2 Dependencies

The `Weapon` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Audio: Used to play sound effects either 2D or at a specified location

12.1.3 Components

The `Weapon` uses the following components:

Name	Description	Type
MuzzleFlash	The muzzle flash of the weapon	USceneComponent*

12.1.4 API Reference

Properties

Property	Description	Type	Default Value
WeaponType	The type of weapon	EWeaponType	
WeaponFireMode	The fire mode of weapon	EWeaponFireMode	
FireRate	The fire rate of the weapon (amount of seconds between each shot)	float	0.2f
MaxShotsPerFireEvent	The amount of shots fired per fire event	int	3
bInfiniteAmmo	Does the weapon have an infinite amount of ammo?	bool	
Ammo	The amount of ammo for the weapon	int	50
WeaponAttachPoint	The attach point used by all weapons	FName	
WeaponTransform	The relative transform of the weapon after it is attached to a character	FTransform	
Projectiles	The projectiles spawned by the weapon	TArray<FWeaponProjectileData>	
MuzzleFlashParticleFx	The muzzle flash particle effects	UNiagaraSystem*	`nullptr`
FireSound	The sound played each time the weapon is fired	USoundBase*	`nullptr`
EmptyClipFireSound	The sound played each time the weapon is fired with an empty clip	USoundBase*	`nullptr`
FireAnimation	The animation played when the weapon is fired	UAnimMontage*	`nullptr`
EmptyClipFireAnimation	The animation played when the weapon is fired with an empty clip	UAnimMontage*	`nullptr`

Events

Name	Description	Params
OnWeaponEquipped	Event used to notify other classes when the weapon is equipped	
OnWeaponUnequipped	Event used to notify other classes when the weapon is unequipped	
OnWeaponFired	Event used to notify other classes when the weapon is fired	
OnWeaponFiredEmptyClip	Event used to notify other classes when the weapon is fired with an empty clip	
OnWeaponAmmoUpdated	Event used to notify other classes when the ammo is updated	**RemainingAmmo (int)** The amount of ammo remaining
OnWeaponAmmoEmpty	Event used to notify other classes when the ammo is empty	

Functions

Name	Description	Params	Return
Equip	Equip the weapon	**TargetCharacter (AUSKCharacter*)** The character that will use the weapon	
Unequip	Unequip the weapon		
StartFiring	Start firing the weapon		
StopFiring	Stop firing the weapon		
AddAmmo	Add more ammo to the weapon	**Amount (int)** The amount of ammo to add	
RemoveAmmo	Remove ammo from the weapon	**Amount (int)** The amount of ammo to remove	
GetAmmoRemaining	Get the amount of ammo remaining		**int** The amount of ammo remaining

12.1.5 Blueprint Usage

You can use the `Weapon` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Weapon > Equip
- Ultimate Starter Kit > Weapon > Unequip
- Ultimate Starter Kit > Weapon > Start Firing
- Ultimate Starter Kit > Weapon > Stop Firing
- Ultimate Starter Kit > Weapon > Add Ammo
- Ultimate Starter Kit > Weapon > Remove Ammo
- Ultimate Starter Kit > Weapon > Get Ammo Remaining

12.1.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `Weapon` can now be used in any of your C++ files:

```
#include "USK/Weapons/Weapon.h"
```

```
void ATestActor::Test()
{
    // Weapon is a pointer to the AWeapon
    Weapon->Equip(TargetCharacter);
    Weapon->Unequip();
    Weapon->StartFiring();
    Weapon->StopFiring();
    Weapon->AddAmmo(Amount);
    Weapon->RemoveAmmo(Amount);
    int AmmoRemaining = Weapon->GetAmmoRemaining();
}
```

12.2 Weapon Type

12.2.1 Introduction

The types of weapons

12.2.2 Values

Value	Description
WeaponOneHanded	A one handed weapon
WeaponTwoHanded	A two handed weapon

12.3 Weapon Fire Mode

12.3.1 Introduction

The types of weapon fire modes

12.3.2 Values

Value	Description
SingleShot	Fires a single shot each time the trigger is pulled
SemiAuto	Fires multiple projectiles each time the trigger is pulled
FullAuto	Continuously fires projectiles while the trigger is held

12.4 Weapon Item

12.4.1 Introduction

A weapon item that can be picked up by a character

12.4.2 API Reference

Properties

Property	Description	Type	Default Value
WeaponClass	The weapon assigned to this item	TSubclassOf<AWeapon>	

12.5 Weapon Ammo Item

12.5.1 Introduction

A weapon ammo item that can be picked up by a character

12.5.2 Dependencies

The `WeaponAmmoItem` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

12.5.3 API Reference

Properties

Property	Description	Type	Default Value
Ammo	The amount of ammo to add to the weapon	int	10
bAddToAnyWeapon	Should the ammo be added to any weapon?	bool	false
AllowedWeapons	The weapons that can use this ammo	TArray<TSubclassOf<AWeapon>>	

12.6 Weapon Projectile

12.6.1 Introduction

The projectile spawned by weapons

12.6.2 Dependencies

The `WeaponProjectile` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

12.6.3 Components

The `WeaponProjectile` uses the following components:

Name	Description	Type
CollisionComponent	The projectile spawned by weapons	USphereComponent*
ProjectileMovementComponent	The projectile movement component used to move the projectile	UProjectileMovementComponent*

12.6.4 API Reference

Properties

Property	Description	Type	Default Value
bDestroyOnHit	Should the projectile be destroyed after hitting something?	bool	true
HitImpulse	The impulse applied to the component that was hit	float	

Functions

Name	Description	Params	Return
GetCollisionComponent	Get the collision component used by the projectile		**USphereComponent*** The collision component used by the projectile
GetProjectileMovementComponent	Get the projectile movement component used to move the projectile		**UProjectileMovementComponent*** The projectile movement component used to move the projectile
NormalImpulse, const FHitResult& HitResult);	Called after the projectile hits something	**HitComponent (HitResult);)** The component responsible for the hit **OtherActor (HitResult);)** The actor that was hit **OtherComponent (HitResult);)** The component that was hit **NormalImpulse (FVector)** The normal impulse of the hit **HitResult (FHitResult&)** Result describing the hit	

12.6.5 Blueprint Usage

You can use the `WeaponProjectile` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Weapon Projectile > Get Collision Component
- Ultimate Starter Kit > Weapon Projectile > Get Projectile Movement Component
- Ultimate Starter Kit > Weapon Projectile > Normal Impulse, const FHit Result& Hit Result);

12.6.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `WeaponProjectile` can now be used in any of your C++ files:

```
#include "USK/Weapons/WeaponProjectile.h"

void ATestActor::Test()
{
    // WeaponProjectile is a pointer to the AWeaponProjectile
    USphereComponent* CollisionComponent = WeaponProjectile->GetCollisionComponent();
    UProjectileMovementComponent* ProjectileMovementComponent = WeaponProjectile->GetProjectileMovementComponent();
    WeaponProjectile->NormalImpulse, const FHitResult& HitResult);(HitComponent, OtherActor, OtherComponent, NormalImpulse, HitResult);
}
```

12.7 Weapon Projectile Data

12.7.1 Introduction

Structure describing a projectile spawned by a weapon

12.7.2 Properties

Property	Description	Type	Default Value
ProjectileClass	The class of the projectile to spawn	TSubclassOf<AWeaponProjectile>	
SpawnTransform	The relative transform of the projectile	FTransform	

12.8 Weapon Utils

12.8.1 Introduction

A Blueprint Function Library class used to manage weapons

12.8.2 Dependencies

The `WeaponUtils` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

12.8.3 API Reference

Functions

Name	Description	Params	Return
EquipWeapon	Equip a weapon to a character	**Owner (AUSKCharacter*)** The owner character **WeaponClass (TSubclassOf)** The weapon class to equip	

12.8.4 Blueprint Usage

You can use the `WeaponUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Weapons > Equip Weapon

12.8.5 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `WeaponUtils` can now be used in any of your C++ files:

```
#include "USK/Weapons/WeaponUtils.h"

void ATestActor::Test()
{
    UWeaponUtils::EquipWeapon(Owner, WeaponClass);
}
```

13. Items

13.1 Allowed Collector

13.1.1 Introduction

The types of actor(s) that can collect an item

13.1.2 Values

Value	Description
AnyActor	Any actor can collect the item
AnyPawn	Any pawn can collect the item
AnyCharacter	Any character can collect the item
AnyPlatformerCharacter	Any platformer character can collect the item
AnyFpsCharacter	Any FPS character can collect the item
PossessedPawn	Only the possessed pawn can collect the item
Custom	A custom array of actor types can collect the item

13.2 Collectable Item

13.2.1 Introduction

An item that can be collected by an actor

13.2.2 Dependencies

The `CollectableItem` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Audio: Used to play sound effects either 2D or at a specified location

13.2.3 Collision

The item requires an actor to overlap with the item before it can be collected. Make sure you have some collider on the actor and that the intended collector can overlap with the item/collider

13.2.4 API Reference

Properties

Property	Description	Type	Default Value
DestroyOnCollected	Should the item be destroyed after it has been collected	bool	true
AllowedCollector	The type of actor that can collect the item	EAllowedCollector	EAllowedCollector::PossessedPawn
AllowedCollectorTypes	The array of actor types that can collect the item	TArray<TSubclassOf<AActor>>	
CollectedSoundEffects	An array of sound effects played when collecting the item	TArray<USoundBase*>	
CollectedParticleFx	The particle effects spawned when collecting the item	UNiagaraSystem*	`nullptr`
CollectedParticleFxSpawnOffset	The offset applied to the location of the collected particles when spawning	FVector	

Functions

Name	Description	Params	Return
CollectItem	Collect the item	**Collector (AActor*)** A pointer to the actor that collected the item	
OnItemCollected	Called after the item is collected	**Collector (AActor*)** A pointer to the actor that collected the item	

13.2.5 Blueprint Usage

You can use the `CollectableItem` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Item > Collect Item
- Ultimate Starter Kit > Item > On Item Collected

13.2.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `CollectableItem` can now be used in any of your C++ files:

```
#include "USK/Items/CollectableItem.h"

void ATestActor::Test()
{
    // CollectableItem is a pointer to the ACollectableItem
    CollectableItem->CollectItem(Collector);
    CollectableItem->OnItemCollected(Collector);
}
```

14. UI & Widgets

14.1 Collectable Item Icon

14.1.1 Introduction

A widget used to display the collection state of an item by showing/hiding an image

14.1.2 Dependencies

The `CollectableItemIcon` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

14.1.3 Required Widgets

You need to add the following before you can compile the `CollectableItemIcon` widget:

Name	Description	Type
Icon	The Icon displayed if the item is collected	UImage*

14.1.4 API Reference

Properties

Property	Description	Type	Default Value
RequiredValue	The required value before the item is considered collected	float	

Functions

Name	Description	Params	Return
UpdateState	Update the collected state (and visibility) of the icon	**IsCollected (bool)** Is the item collected?	
UpdateValue	Update the value of the item and adjust the collected state if necessary	**Value (float)** The current value of the item	
MonitorTrackableValue	Monitor the trackable data and automatically update the icon state whenever the value changes	**TrackableDataComponent (UTrackableDataComponent*)** A reference to the TrackableDataComponent **DataName (FName)** The name of the data item to monitor	

14.1.5 Blueprint Usage

You can use the `CollectableItemIcon` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Update State
- Ultimate Starter Kit > UI > Update Value
- Ultimate Starter Kit > UI > Monitor Trackable Value

14.1.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `CollectableItemIcon` can now be used in any of your C++ files:

```
#include "USK/Widgets/CollectableItemIcon.h"

void ATestActor::Test()
{
    // CollectableItemIcon is a pointer to the UCollectableItemIcon
    CollectableItemIcon->UpdateState(IsCollected);
    CollectableItemIcon->UpdateValue(Value);
    CollectableItemIcon->MonitorTrackableValue(TrackableDataComponent, DataName);
}
```

14.2 Credits

14.2.1 Credits Entry

Introduction

An single credits entry displayed by the credits widget

Properties

Property	Description	Type	Default Value
Title	The title for the credits entry	FText	
Text	The text for the credits entry	FText	
Duration	The duration of the credits entry	float	5.0f
HorizontalAlignment	The horizontal alignment applied to the credits widget when displaying this entry	TEnumAsByte<EHorizontalAlignment>	HAlign_Center
VerticalAlignment	The vertical alignment applied to the credits widget when displaying this entry	TEnumAsByte<EVerticalAlignment>	VAlign_Center

14.2.2 Credits Widget

Introduction

Widget used to display multiple animated credits entries using different alignment options and durations

Dependencies

The `CreditsWidget` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

Required Widgets

You need to add the following before you can compile the `CreditsWidget` widget:

Name	Description	Type
Root	The root container of the widget	UPanelWidget*
Container	The container used to display the credits entries	UPanelWidget*
Text	The text block used to display the text of the credits entry	UTextBlock*

Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
Title	The text block used to display the title of the credits entry	UTextBlock*

Optional Animations

You can add the following widgets to enable extra functionality:

Name	Description
ShowAnimation	The animation played each time a new entry is shown
HideAnimation	The animation played at the end of each entry

API Reference

PROPERTIES

Property	Description	Type	Default Value
ShowAnimation	The animation played each time a new entry is shown	UWidgetAnimation*	`nullptr`
HideAnimation	The animation played at the end of each entry	UWidgetAnimation*	`nullptr`
AutoStart	Should the credits automatically be started when the widget is constructed?	bool	true
RemoveOnCompletion	Should the widget automatically be removed from the viewport when the credits are finished?	bool	true
StartDelay	The delay in seconds before the first credits entry is shown after starting	float	1.0f
DelayBetweenEntries	The delay in seconds between the previous hide animation and the next show animation	float	3.0f
Credits	The array of credits entries to be displayed	TArray<FCreditsEntry>	

EVENTS

Name	Description	Params
OnCreditsFinished	Event used to notify other classes when the credits are finished	
OnCreditsNextEntryStarted	Event used to notify other classes every time a next entry is started	**Name (int)** The index of the entry that was started

FUNCTIONS

Name	Description	Params	Return
Start	Start showing the credits		

Blueprint Usage

You can use the `CreditsWidget` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Start

C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `CreditsWidget` can now be used in any of your C++ files:

```
#include "USK/Widgets/CreditsWidget.h"

void ATestActor::Test()
{
    // CreditsWidget is a pointer to the UCreditsWidget
    CreditsWidget->Start();
}
```

14.3 FPS Counter

14.3.1 Introduction

A widget used to display the current framerate

14.3.2 Dependencies

The `FpsCounter` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data

14.3.3 Required Widgets

There is already a `FpsCounter_Implementation` that you can use in your projects. But if you create your own instance of this widget, you need to add the following before you can compile:

Name	Description	Type
FramerateText	The text block used to display the framerate	UTextBlock*

14.3.4 API Reference

Properties

Property	Description	Type	Default Value
UpdateDelay	The delay in seconds between each update	float	0.125f
HighFramerate	A framerate that is considered high and will use the high color	int	60
MediumFramerate	A framerate that is considered medium and will use the medium color	int	30
HighColor	The color used to display high framerates	FLinearColor	FLinearColor::Green
MediumColor	The color used to display medium framerates	FLinearColor	FLinearColor::Yellow
LowColor	The color used to display low framerates	FLinearColor	FLinearColor::Red

Functions

Name	Description	Params	Return
UpdateVisibility	Update the visibility of the widget	**IsVisible (bool)** Is the widget visible?	

14.3.5 Blueprint Usage

You can use the `FpsCounter` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Update Visibility

14.3.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `FpsCounter` can now be used in any of your C++ files:

```
#include "USK/Widgets/FpsCounter.h"

void ATestActor::Test()
{
    // FpsCounter is a pointer to the UFpsCounter
    FpsCounter->UpdateVisibility(IsVisible);
}
```

14.4 Input Indicator

14.4.1 Introduction

A widget used to display input indicators based on the current input device and input action

14.4.2 Dependencies

The `InputIndicator` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data

14.4.3 Required Widgets

There is already a `InputIndicator_Implementation` that you can use in your projects. But if you create your own instance of this widget, you need to add the following before you can compile:

Name	Description	Type
Container	The container used to display multiple images	UHorizontalBox*

14.4.4 API Reference

Properties

Property	Description	Type	Default Value
InputIndicatorIconClass	The input indicator icon class	TSubclassOf<UInputIndicatorIcon>	
Action	The input action displayed by widget	UInputAction*	`nullptr`
Size	The size of the image	float	50.0f
Amount	The amount of images to display for the input action	int	1

Functions

Name	Description	Params	Return
UpdateAction	Update the input action displayed by the widget	**NewAction (UInputAction*)** The new action **NewAmount (int)** The new amount of images to display	

14.4.5 Blueprint Usage

You can use the `InputIndicator` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Update Action

14.4.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `InputIndicator` can now be used in any of your C++ files:

```
#include "USK/Widgets/InputIndicator.h"

void ATestActor::Test()
{
    // InputIndicator is a pointer to the UInputIndicator
    InputIndicator->UpdateAction(NewAction, NewAmount);
}
```

14.5 Input Indicator Icon

14.5.1 Introduction

A widget used to display a single input indicator image

14.5.2 Dependencies

The `InputIndicatorIcon` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

14.5.3 Required Widgets

There is already a `InputIndicatorIcon_Implementation` that you can use in your projects. But if you create your own instance of this widget, you need to add the following before you can compile:

Name	Description	Type
Container	The size box container used to resize the image	USizeBox*
Image	The image used to display the input indicator	UImage*

14.5.4 API Reference

Properties

Property	Description	Type	Default Value

Functions

Name	Description	Params	Return
UpdateIcon	Update the icon	**Size (float)** The size of the image **Icon (UTexture2D*)** The new icon	

14.5.5 Blueprint Usage

You can use the `InputIndicatorIcon` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Update Icon

14.5.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `InputIndicatorIcon` can now be used in any of your C++ files:

```
#include "USK/Widgets/InputIndicatorIcon.h"

void ATestActor::Test()
{
    // InputIndicatorIcon is a pointer to the UInputIndicatorIcon
```

```
    InputIndicatorIcon->UpdateIcon(Size, Icon);
}
```

14.6 Menu

14.6.1 Introduction

A widget used to display menu items and handle navigation between the items

14.6.2 Dependencies

The `Menu` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data
- Audio: Used to play sound effects either 2D or at a specified location

14.6.3 Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
ScrollContainer	Scroll container used for large menus with many items	UScrollBox*
Container	The container used to display the menu items	UPanelWidget*

14.6.4 API Reference

Properties

Property	Description	Type	Default Value
AddInputBindingOnLoad	Should the input binding automatically be added as soon as the widget is loaded?	bool	
PauseGameWhileVisible	Should the game automatically be paused/resumed based on the visibility of the menu?	bool	
DisableWhilePaused	Should the menu be disabled while the game is paused?	bool	
SelectedSFX	The sound effect played when a menu item is selected	USoundBase*	`nullptr`
BackSFX	The sound effect played when trying to go back to a previous menu or closing the menu through the back button	USoundBase*	`nullptr`
InputMappingContext	The input mapping context used to navigate the menu	UInputMappingContext*	`nullptr`
MenuUpInputAction	The input action used to navigate up	UInputAction*	`nullptr`
MenuDownInputAction	The input action used to navigate down	UInputAction*	`nullptr`
MenuLeftInputAction	The input action used to navigate left	UInputAction*	`nullptr`
MenuRightInputAction	The input action used to navigate right	UInputAction*	`nullptr`
MenuSelectInputAction	The input action used to select a menu item	UInputAction*	`nullptr`
MenuBackInputAction	The input action used to go back to a previous menu or close the menu	UInputAction*	`nullptr`

Events

Name	Description	Params
OnBackEvent	Event used to handle the back/close action of the menu	

Functions

Name	Description	Params	Return
OnMenuUp	Navigate up or increase the value		
OnMenuUpHold	Increase the value while holding the menu up key		
OnMenuDown	Navigate down or decrease the value		
OnMenuDownHold	Decrease the value while holding the menu down key		
OnMenuLeft	Navigate left or decrease the value		
OnMenuLeftHold	Decrease the value while holding the menu left key		
OnMenuRight	Navigate right or increase the value		
OnMenuRightHold	Increase the value while holding the menu right key		
OnMenuSelected	Select the current menu item		
OnMenuBack	Go back to a previous menu or close the menu		
RequestHighlight	Request to highlight a specific menu item	**MenuItem (UMenuItem*)** The menu item to highlight	
RemoveHighlight	Request to remove the highlighted state from a specific menu item	**MenuItem (UMenuItem*)** The menu item to remove the highlighted state from	
AddMenuItem	Add a menu item to the container	**MenuItem (UMenuItem*)** The menu item to add	

14.6.5 Blueprint Usage

You can use the `Menu` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > On Menu Up
- Ultimate Starter Kit > UI > On Menu Up Hold
- Ultimate Starter Kit > UI > On Menu Down
- Ultimate Starter Kit > UI > On Menu Down Hold
- Ultimate Starter Kit > UI > On Menu Left
- Ultimate Starter Kit > UI > On Menu Left Hold
- Ultimate Starter Kit > UI > On Menu Right
- Ultimate Starter Kit > UI > On Menu Right Hold
- Ultimate Starter Kit > UI > On Menu Selected
- Ultimate Starter Kit > UI > On Menu Back
- Ultimate Starter Kit > UI > Request Highlight
- Ultimate Starter Kit > UI > Remove Highlight
- Ultimate Starter Kit > UI > Add Menu Item

14.6.6 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `Menu` can now be used in any of your C++ files:

```
#include "USK/Widgets/Menu.h"

void ATestActor::Test()
{
    // Menu is a pointer to the UMenu
    Menu->OnMenuUp();
    Menu->OnMenuUpHold();
    Menu->OnMenuDown();
    Menu->OnMenuDownHold();
    Menu->OnMenuLeft();
    Menu->OnMenuLeftHold();
    Menu->OnMenuRight();
    Menu->OnMenuRightHold();
    Menu->OnMenuSelected();
    Menu->OnMenuBack();
    Menu->RequestHighlight(MenuItem);
    Menu->RemoveHighlight(MenuItem);
    Menu->AddMenuItem(MenuItem);
}
```

14.7 Menu Item

14.7.1 Navigation

Introduction

All the supported menu navigation types

Values

Value	Description
Disabled	No navigation allowed
HighlightItem	Highlight a different menu item
IncreaseDecreaseValue	Increase or decrease the value

14.7.2 Value Update Method

Introduction

The method used to update the value of a menu item

Values

Value	Description
SinglePress	The value is only updated when the button is pressed
Hold	The value is updated while the button is held down

14.7.3 Menu Item Widget

Introduction

A widget used to display a title, text and value in the form of a menu item

Dependencies

The `MenuItem` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data
- Audio: Used to play sound effects either 2D or at a specified location

Optional Widgets

You can add the following widgets to enable extra functionality:

Name	Description	Type
Title	The TextBlock used to display the title of the menu item	UTextBlock*
NormalText	The TextBlock used to display the text of the menu item while not highlighted	UTextBlock*
HighlightedText	The TextBlock used to display the text of the menu item while highlighted	UTextBlock*
ValueText	The TextBlock used to display the current value of the menu item	UTextBlock*
HighlightedValueText	The TextBlock used to display the current value of the menu item while highlighted	UTextBlock*
SelectButton	The button used to select the menu item	UButton*
ValueSlider	The slider used to display and update the current value of the menu item	USlider*
IncreaseValueButton	The button used to increase the value of the menu item	UButton*
DecreaseValueButton	The button used to decrease the value of the menu item	UButton*
BorderLeft	The border displayed on the left of the menu item	UImage*
BorderRight	The border displayed on the right of the menu item	UImage*
BorderBackground	The background border display in the menu item	UImage*
ButtonLeft	The button displayed on the left of the menu item	UImage*
ButtonRight	The button displayed on the right of the menu item	UImage*
ButtonBackground	The background button display in the menu item	UImage*
InputIndicator	The background button display in the menu item	UImage*

Optional Animations

You can add the following widgets to enable extra functionality:

Name	Description
HighlightedAnimation	The animation played when the menu item is highlighted

API Reference

PROPERTIES

Property	Description	Type	Default Value
HighlightedAnimation	The animation played when the menu item is highlighted	UWidgetAnimation*	`nullptr`
FocusByDefault	Should the menu item be focused by default?	bool	
HideOnConsoles	Should the menu item be hidden on consoles?	bool	
TitleText	The title text displayed in the menu item	FText	
MenuItemText	The text displayed in the menu item	FText	
HighlightedSFX	The sound effect played when the menu item is highlighted	USoundBase*	`nullptr`
BorderNormalColor	The color of the border when not highlighted	FLinearColor	
BorderHighlightedColor	The color of the border when highlighted	FLinearColor	
BorderNormalImage	The image of the border when not highlighted	UTexture2D*	`nullptr`
BorderHighlightedImage	The image of the border when highlighted	UTexture2D*	`nullptr`
BorderLeftNormalImage	The image of the left border when not highlighted	UTexture2D*	`nullptr`
BorderLeftHighlightedImage	The image of the left border when highlighted	UTexture2D*	`nullptr`
BorderRightNormalImage	The image of the right border	UTexture2D*	`nullptr`

	when not highlighted		
BorderRightHighlightedImage	The image of the right border when highlighted	UTexture2D*	`nullptr`
BackgroundNormalColor	The color of the button when not highlighted	FLinearColor	
BackgroundHighlightedColor	The color of the button when highlighted	FLinearColor	
BackgroundNormalImage	The image of the button when not highlighted	UTexture2D*	`nullptr`
BackgroundHighlightedImage	The image of the button when highlighted	UTexture2D*	`nullptr`
BackgroundLeftNormalImage	The image of the left button when not highlighted	UTexture2D*	`nullptr`
BackgroundLeftHighlightedImage	The image of the left button when highlighted	UTexture2D*	`nullptr`
BackgroundRightNormalImage	The image of the right button when not highlighted	UTexture2D*	`nullptr`
BackgroundRightHighlightedImage	The image of the right button when highlighted	UTexture2D*	`nullptr`
ValueUpdateMethod	The method used to update the value of the menu item	EMenuItemValueUpdateMethod	EMenuItemValueUpdateMethod::SinglePress
IncrementSinglePress	The increment used when updating the value when the key is pressed	float	1.0f
IncrementHold	The increment used when updating the value when the	float	0.15f

	key is held down		
SettingsItemType	The type of setting item managed by this menu item (changing this will overwrite other settings)	ESettingsItemType	ESettingsItemType::None
AutoSaveSettingsOnValueChanged	Should the settings managed by this menu item automatically be saved when the value is changed?	bool	true
AutoSaveSettingsOnHighlightRemoved	Should the settings managed by this menu item automatically be saved when the highlight state is removed?	bool	true
AutoSaveSettingsOnSelected	Should the settings managed by this menu item automatically be saved when the menu item is selected?	bool	true
InputDevice	The input device associated with the action to rebind	EInputDevice	EInputDevice::Unknown
InputMappingContext	The input mapping context containing the action to rebind	UInputMappingContext*	`nullptr`
InputAction	The input action to rebind	UInputAction*	`nullptr`
MappableName	The player mappable name for the action to rebind	FName	
ShowValueSlider	Should the value slider be	bool	false

	shown for this menu item?		
ShowValueButtons	Should the increase/ decrease value buttons be shown for this menu item?	bool	false
ValueMapping	A mapping of possible values to text	TMap<int, FText>	
DefaultValue	The default value of the menu item	int	100
MinValue	The minimum value of the menu item	int	0
MaxValue	The maximum value of the menu item	int	100
AllowSelection	Can the menu item be selected?	bool	true
VerticalNavigation	The type of navigation used by the menu item when pressing the up or down key	EMenuNavigation	EMenuNavigation::HighlightItem
MenuItemUp	The menu item highlighted when the up key is pressed	UMenuItem*	`nullptr`
MenuItemDown	The menu item highlighted when the down key is pressed	UMenuItem*	`nullptr`
HorizontalNavigation	The type of navigation used by the menu item when pressing the left or right key	EMenuNavigation	EMenuNavigation::HighlightItem
MenuItemLeft	The menu item highlighted when the left key is pressed	UMenuItem*	`nullptr`
MenuItemRight	The menu item highlighted when the right key is pressed	UMenuItem*	`nullptr`

Menu	A reference to the menu that contains this menu item	UMenu*	`nullptr`

EVENTS

Name	Description	Params
OnSelected	Event used to notify other classes that the menu item was selected	
OnSelectedInContainer	Event used to notify other classes that a specific menu item in the container was selected	**Index (int)** The index of the menu item that was selected
OnHighlighted	Event used to notify other classes that the menu item was highlighted	
OnHighlightRemoved	Event used to notify other classes that the menu item's highlighted state was removed	
OnValueChanged	Event used to notify other classes that the menu item's value was updated	**Value (int)** The new value of the menu item

FUNCTIONS

Name	Description	Params	Return
SetText	Set the text displayed in the menu item	**Text (FText&)** The new text displayed in the menu item	
SetTitle	Set the title displayed in the menu item	**Text (FText&)** The new title displayed in the menu item	
SetHighlightedState	Set the highlighted state of the menu item	**IsHighlighted (bool)** Is the menu item highlighted? **PlayHighlightedAnimation (bool)** Should the highlighted animation be played? **PlayHighlightedSound (bool)** Should the highlighted sound effect be played?	
IsHighlighted	Check if the menu item is highlighted		**bool** A boolean value indicating if the menu item is highlighted
GetValue	Get the current value of the menu item		**int** The current value of the menu item
UpdateValue	Update the value of the menu item	**Increment (float)** The amount added to the current value of the menu item	
SelectItem	Select the menu item		
SaveSettings	Save the settings managed by this menu item		
ApplySettings	Apply the settings managed by this menu item		
GetInputActionKey	Get the key used by the specified input action		**FKey** The key used by the specified input action
OnMenuBack	Called when trying to go back in the menu		**bool** A boolean value indicating if the back event was handled
AnyKeyPressed	Called after any key is pressed by the player (used to remap controls)	**Key (FKey)** The key pressed by the player	
ApplyKeyBinding	Apply the key binding for the input action		
IsWaitingForKeyPress	Is the menu item waiting for a key press?		**bool** A boolean value indicating if the menu item is waiting for a key press

Blueprint Usage

You can use the `MenuItem` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > UI > Set Text
- Ultimate Starter Kit > UI > Set Title
- Ultimate Starter Kit > UI > Set Highlighted State
- Ultimate Starter Kit > UI > Is Highlighted
- Ultimate Starter Kit > UI > Get Value
- Ultimate Starter Kit > UI > Update Value
- Ultimate Starter Kit > UI > Select Item
- Ultimate Starter Kit > UI > Save Settings
- Ultimate Starter Kit > UI > Apply Settings
- Ultimate Starter Kit > UI > Get Input Action Key
- Ultimate Starter Kit > UI > On Menu Back
- Ultimate Starter Kit > UI > Any Key Pressed
- Ultimate Starter Kit > UI > Apply Key Binding
- Ultimate Starter Kit > UI > Is Waiting For Key Press

C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `MenuItem` can now be used in any of your C++ files:

```
#include "USK/Widgets/MenuItem.h"

void ATestActor::Test()
{
    // MenuItem is a pointer to the UMenuItem
    MenuItem->SetText(Text);
    MenuItem->SetTitle(Text);
    MenuItem->SetHighlightedState(IsHighlighted, PlayHighlightedAnimation, PlayHighlightedSound);
    bool IsHighlightedValue = MenuItem->IsHighlighted();
    int Value = MenuItem->GetValue();
    MenuItem->UpdateValue(Increment);
    MenuItem->SelectItem();
    MenuItem->SaveSettings();
    MenuItem->ApplySettings();
    FKey InputActionKey = MenuItem->GetInputActionKey();
    bool OnMenuBackValue = MenuItem->OnMenuBack();
    MenuItem->AnyKeyPressed(Key);
    MenuItem->ApplyKeyBinding();
    bool IsWaitingForKeyPressValue = MenuItem->IsWaitingForKeyPress();
}
```

15. Settings

15.1 Data

15.1.1 Introduction

The settings data that is saved/loaded

15.1.2 API Reference

Properties

Property	Description	Type	Default Value
AudioMasterModified	Was the master audio volume modified?	bool	
AudioMaster	The master audio volume	float	
AudioMusicModified	Was the music volume modified?	bool	
AudioMusic	The music volume	float	
AudioEffectsModified	Was the effects volume modified?	bool	
AudioEffects	The effects volume	float	
AudioUiModified	Was the UI volume modified?	bool	
AudioUi	The UI volume	float	
AudioVoiceModified	Was the voice volume modified?	bool	
AudioVoice	The voice volume	float	
GraphicsResolutionX	The X value of the saved resolution	int	
GraphicsResolutionY	The Y value of the saved resolution	int	
GraphicsFullscreenModified	Was the graphics fullscreen setting modified?	bool	
GraphicsFullscreen	The graphics fullscreen value	bool	
GraphicsViewDistanceModified	Was the graphics view distance setting modified?	bool	
GraphicsViewDistance	The graphics view distance value	int	
GraphicsAntiAliasingModified	Was the graphics anti-aliasing setting modified?	bool	
GraphicsAntiAliasing	The graphics anti-aliasing value	int	
GraphicsPostProcessingModified	Was the graphics post processing setting modified?	bool	
GraphicsPostProcessing	The graphics post processing value	int	
GraphicsShadowQualityModified	Was the graphics shadow quality setting modified?	bool	
GraphicsShadowQuality	The graphics shadow quality value	int	
GraphicsTextureQualityModified	Was the graphics texture quality setting modified?	bool	
GraphicsTextureQuality	The graphics texture quality value	int	
GraphicsVisualEffectsModified	Was the graphics visual effects setting modified?	bool	
GraphicsVisualEffects	The graphics visual effects value	int	
GraphicsShadingQualityModified	Was the graphics shading quality setting modified?	bool	
GraphicsShadingQuality	The graphics shading quality value	int	
GraphicsVsyncModified	Was the graphics vsync setting modified?	bool	
GraphicsVsync	The graphics vsync value	bool	
GraphicsFpsIndicatorModified	Was the graphics FPS indicator setting modified?	bool	
GraphicsFpsIndicator	The graphics FPS indicator value	bool	
AccessibilityColorBlindMode	The color blind mode	int	
AccessibilityColorBlindModeSeverity	The severity of the color blind mode	float	100.0f

KeyBindings	A map of all key bindings changed by the player	TMap<FName, FKey>	

15.2 Config

15.2.1 Introduction

The configuration used for managing settings

15.2.2 API Reference

Properties

Property	Description	Type	Default Value
AudioMasterImplementation	The implementation for the audio master settings item	TSubclassOf<USettingsItem>	USettingsItemAudioMaster::StaticClas
AudioMasterSoundMix	The sound mix used to manage all sound classes	USoundMix*	`nullptr`
AudioMasterText	The text displayed in the master audio settings item	FText	
AudioMasterMin	The minimum value for the master audio settings item	float	0.0f
AudioMasterMax	The maximum value for the master audio settings item	float	100.0f
AudioMasterDefault	The default value for the master audio settings item	float	100.0f
AudioMusicImplementation	The implementation for the audio music settings item	TSubclassOf<USettingsItem>	USettingsItemAudioMusic::StaticClas
AudioMusicSoundClass	The sound class used by all music	USoundClass*	`nullptr`
AudioMusicText	The text displayed in the music audio settings item	FText	
AudioMusicMin	The minimum value for the music audio settings item	float	0.0f
AudioMusicMax	The maximum value for the music audio settings item	float	100.0f
AudioMusicDefault	The default value for the	float	100.0f

	music audio settings item		
AudioEffectsImplementation	The implementation for the audio effects settings item	TSubclassOf<USettingsItem>	USettingsItemAudioEffects::StaticClas
AudioEffectsSoundClass	The sound class used by all effects	USoundClass*	`nullptr`
AudioEffectsText	The text displayed in the effects audio settings item	FText	
AudioEffectsMin	The minimum value for the effects audio settings item	float	0.0f
AudioEffectsMax	The maximum value for the effects audio settings item	float	100.0f
AudioEffectsDefault	The default value for the effects audio settings item	float	100.0f
AudioUiImplementation	The implementation for the audio UI settings item	TSubclassOf<USettingsItem>	USettingsItemAudioUi::StaticClas
AudioUiSoundClass	The sound class used by all UI	USoundClass*	`nullptr`
AudioUiText	The text displayed in the UI audio settings item	FText	
AudioUiMin	The minimum value for the UI audio settings item	float	0.0f
AudioUiMax	The maximum value for the UI audio settings item	float	100.0f
AudioUiDefault	The default value for the UI audio settings item	float	100.0f

AudioVoiceImplementation	The implementation for the audio voice settings item	TSubclassOf<USettingsItem>	USettingsItemAudioVoice::StaticClas
AudioVoiceSoundClass	The sound class used by all voice	USoundClass*	`nullptr`
AudioVoiceText	The text displayed in the voice audio settings item	FText	
AudioVoiceMin	The minimum value for the voice audio settings item	float	0.0f
AudioVoiceMax	The maximum value for the voice audio settings item	float	100.0f
AudioVoiceDefault	The default value for the voice audio settings item	float	100.0f
GraphicsResolutionImplementation	The implementation for the graphics resolution settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsResolution::StaticClas
GraphicsResolutionText	The text displayed in the graphics resolution settings item	FText	
GraphicsFullscreenImplementation	The implementation for the graphics fullscreen settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsFullscreen::StaticClas
GraphicsFullscreenText	The text displayed in the graphics fullscreen settings item	FText	
GraphicsFullscreenEnabledText	The text displayed when fullscreen is enabled	FText	
GraphicsFullscreenDisabledText	The text displayed when	FText	

	fullscreen is disabled		
GraphicsFullscreenDefault	The default value of the fullscreen setting	bool	true
GraphicsViewDistanceImplementation	The implementation for the graphics view distance settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsViewDistance::StaticClas
GraphicsViewDistanceText	The text displayed in the graphics view distance settings item	FText	
GraphicsViewDistanceNearValueText	The text displayed when the near value is used for the view distance setting	FText	
GraphicsViewDistanceMediumValueText	The text displayed when the medium value is used for the view distance setting	FText	
GraphicsViewDistanceFarValueText	The text displayed when the far value is used for the view distance setting	FText	
GraphicsViewDistanceEpicValueText	The text displayed when the epic value is used for the view distance setting	FText	
GraphicsViewDistanceCinematicValueText	The text displayed when the cinematic value is used for the view distance setting	FText	
GraphicsViewDistanceDefault	The default value of the view distance setting	int	2
GraphicsAntiAliasingImplementation	The implementation for the graphics	TSubclassOf<USettingsItem>	USettingsItemGraphicsAntiAliasing::StaticClas

	anti-aliasing settings item		
GraphicsAntiAliasingText	The text displayed in the graphics anti-aliasing settings item	FText	
GraphicsAntiAliasingLowValueText	The text displayed when the low value is used for the anti-aliasing setting	FText	
GraphicsAntiAliasingMediumValueText	The text displayed when the medium value is used for the anti-aliasing setting	FText	
GraphicsAntiAliasingHighValueText	The text displayed when the high value is used for the anti-aliasing setting	FText	
GraphicsAntiAliasingEpicValueText	The text displayed when the epic value is used for the anti-aliasing setting	FText	
GraphicsAntiAliasingCinematicValueText	The text displayed when the cinematic value is used for the anti-aliasing setting	FText	
GraphicsAntiAliasingDefault	The default value of the anti-aliasing setting	int	2
GraphicsPostProcessingImplementation	The implementation for the graphics post processing settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsPostProcessing::StaticClas
GraphicsPostProcessingText	The text displayed in the graphics post processing settings item	FText	
GraphicsPostProcessingLowValueText	The text displayed when	FText	

	the low value is used for the post processing setting		
GraphicsPostProcessingMediumValueText	The text displayed when the medium value is used for the post processing setting	FText	
GraphicsPostProcessingHighValueText	The text displayed when the high value is used for the post processing setting	FText	
GraphicsPostProcessingEpicValueText	The text displayed when the epic value is used for the post processing setting	FText	
GraphicsPostProcessingCinematicValueText	The text displayed when the cinematic value is used for the post processing setting	FText	
GraphicsPostProcessingDefault	The default value of the post processing setting	int	2
GraphicsShadowQualityImplementation	The implementation for the graphics shadow quality settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsShadowQuality::StaticClas
GraphicsShadowQualityText	The text displayed in the graphics shadow quality settings item	FText	
GraphicsShadowQualityLowValueText	The text displayed when the low value is used for the shadow quality setting	FText	
GraphicsShadowQualityMediumValueText	The text displayed when the medium	FText	

	value is used for the shadow quality setting		
GraphicsShadowQualityHighValueText	The text displayed when the high value is used for the shadow quality setting	FText	
GraphicsShadowQualityEpicValueText	The text displayed when the epic value is used for the shadow quality setting	FText	
GraphicsShadowQualityCinematicValueText	The text displayed when the cinematic value is used for the shadow quality setting	FText	
GraphicsShadowQualityDefault	The default value of the shadow quality setting	int	2
GraphicsTextureQualityImplementation	The implementation for the graphics texture quality settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsTextureQuality::StaticClas
GraphicsTextureQualityText	The text displayed in the graphics texture quality settings item	FText	
GraphicsTextureQualityLowValueText	The text displayed when the low value is used for the texture quality setting	FText	
GraphicsTextureQualityMediumValueText	The text displayed when the medium value is used for the texture quality setting	FText	
GraphicsTextureQualityHighValueText	The text displayed when the high value is used for the texture quality setting	FText	

GraphicsTextureQualityEpicValueText	The text displayed when the epic value is used for the texture quality setting	FText	
GraphicsTextureQualityCinematicValueText	The text displayed when the cinematic value is used for the texture quality setting	FText	
GraphicsTextureQualityDefault	The default value of the texture quality setting	int	2
GraphicsVisualEffectsImplementation	The implementation for the graphics visual effects settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsVisualEffects::StaticClas
GraphicsVisualEffectsText	The text displayed in the graphics visual effects settings item	FText	
GraphicsVisualEffectsLowValueText	The text displayed when the low value is used for the visual effects setting	FText	
GraphicsVisualEffectsMediumValueText	The text displayed when the medium value is used for the visual effects setting	FText	
GraphicsVisualEffectsHighValueText	The text displayed when the high value is used for the visual effects setting	FText	
GraphicsVisualEffectsEpicValueText	The text displayed when the epic value is used for the visual effects setting	FText	
GraphicsVisualEffectsCinematicValueText	The text displayed when the cinematic	FText	

	value is used for the visual effects setting		
GraphicsVisualEffectsDefault	The default value of the visual effects setting	int	2
GraphicsShadingQualityImplementation	The implementation for the graphics shading quality settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsShadingQuality::StaticClas
GraphicsShadingQualityText	The text displayed in the graphics shading quality settings item	FText	
GraphicsShadingQualityLowValueText	The text displayed when the low value is used for the shading quality setting	FText	
GraphicsShadingQualityMediumValueText	The text displayed when the medium value is used for the shading quality setting	FText	
GraphicsShadingQualityHighValueText	The text displayed when the high value is used for the shading quality setting	FText	
GraphicsShadingQualityEpicValueText	The text displayed when the epic value is used for the shading quality setting	FText	
GraphicsShadingQualityCinematicValueText	The text displayed when the cinematic value is used for the shading quality setting	FText	
GraphicsShadingQualityDefault	The default value of the shading quality setting	int	2
GraphicsVsyncImplementation	The implementation	TSubclassOf<USettingsItem>	USettingsItemGraphicsVsync::StaticClas

	for the graphics vsync settings item		
GraphicsVsyncText	The text displayed in the graphics vsync settings item	FText	
GraphicsVsyncEnabledText	The text displayed when the vsync setting is enabled	FText	
GraphicsVsyncDisabledText	The text displayed when the vsync setting is disabled	FText	
GraphicsVsyncDefault	The default value of the vsync setting	bool	
GraphicsFpsIndicatorImplementation	The implementation for the graphics FPS indicator settings item	TSubclassOf<USettingsItem>	USettingsItemGraphicsFpsIndicator::StaticClas
GraphicsFpsIndicatorText	The text displayed in the graphics FPS indicator settings item	FText	
GraphicsFpsIndicatorEnabledText	The text displayed when the FPS indicator setting is enabled	FText	
GraphicsFpsIndicatorDisabledText	The text displayed when the FPS indicator setting is disabled	FText	
GraphicsFpsIndicatorDefault	The default value of the FPS indicator setting	bool	
AccessibilityColorBlindModeImplementation	The implementation for the accessibility color blind	TSubclassOf<USettingsItem>	USettingsItemAccessibilityColorBlindMode::Static

	mode settings item		
AccessibilityColorBlindModeText	The text displayed in the accessibility color blind mode settings item	FText	
AccessibilityColorBlindModeNormalVisionText	The text displayed when the normal vision value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeDeuteranopiaText	The text displayed when the deuteranopia value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeDeuteranomalyText	The text displayed when the deuteranomaly value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeProtanopiaText	The text displayed when the protanopia value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeProtanomalyText	The text displayed when the protanomaly value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeTritanopiaText	The text displayed when the tritanopia	FText	

	value is used for the accessibility color blind mode setting		
AccessibilityColorBlindModeTritanomalyText	The text displayed when the tritanomaly value is used for the accessibility color blind mode setting	FText	
AccessibilityColorBlindModeSeverityImplementation	The implementation for the accessibility color blind mode severity settings item	TSubclassOf<USettingsItem>	USettingsItemAccessibilityColorBlindModeSeverity
AccessibilityColorBlindModeSeverityText	The text displayed in the accessibility color blind mode severity settings item	FText	
ControlsRemapImplementation	The implementation for the controls remap settings item	TSubclassOf<USettingsItem>	USettingsItemControlsRemap::StaticClas
ControlsWaitingForKeyPressText	The text displayed in the menu item while waiting for the user to press a new key	FText	

15.3 Items

15.3.1 Types

Introduction

An enum of all supported settings items

Values

Value	Description
None	A setting not managed by the game instance
AudioMaster	The master audio volume setting
AudioMusic	The music audio volume setting
AudioEffects	The effects audio volume setting
AudioUi	The UI audio volume setting
AudioVoice	The voice audio volume setting
GraphicsResolution	The graphics resolution setting
GraphicsFullscreen	The graphics fullscreen setting
GraphicsViewDistance	The graphics view distance setting
GraphicsAntiAliasing	The graphics anti-aliasing setting
GraphicsPostProcessing	The graphics post processing setting
GraphicsShadowQuality	The graphics shadow quality setting
GraphicsTextureQuality	The graphics texture quality setting
GraphicsVisualEffects	The graphics visual effects setting
GraphicsShadingQuality	The graphics shading quality setting
GraphicsVSync	The graphics vsync setting
GraphicsFpsIndicator	The graphics FPS indicator setting
AccessibilityColorBlindMode	The accessibility color blind mode setting
AccessibilityColorBlindModeSeverity	The accessibility color blind mode severity setting
ControlsRemap	Remap the controls

15.3.2 Logic

Introduction

An implementation for a settings item controlling how the setting is configured, saved and applied

Dependencies

The `SettingsItem` relies on other components of this plugin to work:

- Game Instance: Used to monitor for input device changes and handle saving/loading game data

Implementations

There is already implementations for all settings items. But you can expand this if needed

Category	Name	Description
Audio	SettingsItemAudioMaster	An implementation for the audio master settings item
Audio	SettingsItemAudioMusic	An implementation for the audio music settings item
Audio	SettingsItemAudioEffects	An implementation for the audio effects settings item
Audio	SettingsItemAudioUi	An implementation for the audio UI settings item
Audio	SettingsItemAudioVoice	An implementation for the audio voice settings item
Graphics	SettingsItemGraphicsResolution	An implementation for the graphics resolution settings item
Graphics	SettingsItemGraphicsFullscreen	An implementation for the graphics fullscreen settings item
Graphics	SettingsItemGraphicsViewDistance	An implementation for the graphics view distance settings item
Graphics	SettingsItemGraphicsAntiAliasing	An implementation for the graphics anti-aliasing settings item
Graphics	SettingsItemGraphicsPostProcessing	An implementation for the graphics post processing settings item
Graphics	SettingsItemGraphicsShadowQuality	An implementation for the graphics shadow quality settings item
Graphics	SettingsItemGraphicsTextureQuality	An implementation for the graphics texture quality settings item
Graphics	SettingsItemGraphicsVisualEffects	An implementation for the graphics visual effects settings item
Graphics	SettingsItemGraphicsShadingQuality	An implementation for the graphics shading quality settings item
Graphics	SettingsItemGraphicsVsync	An implementation for the graphics vsync settings item
Graphics	SettingsItemGraphicsFpsIndicator	An implementation for the graphics FPS indicator settings item
Accessibility	SettingsItemAccessibilityColorBlindMode	An implementation for the accessibility color blind mode settings item
Accessibility	SettingsItemAccessibilityColorBlindModeSeverity	An implementation for the accessibility color blind mode severity settings item
Controls	SettingsItemControlsRemap	An implementation for the controls settings item

API Reference

FUNCTIONS

Name	Description	Params	Return
ConfigureMenuItem	Configure the menu item	**Config (void)** The settings config specified in the game instance **Settings (USettingsData*)** The current settings data **MenuItem (UMenuItem*)** The menu item to configure	
SaveSettings	Save the settings managed by a menu item	**Settings (USettingsData*)** The current settings data **MenuItem (UMenuItem*)** The menu item containing the updated settings	**USettingsData*** The updated settings data
ApplySettings	Apply the settings	**World (UObject*)** The world context **Config (USettingsConfig*)** The settings config specified in the game instance **Settings (USettingsData*)** The current settings data	

Blueprint Usage

You can use the `SettingsItem` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Settings > Configure Menu Item
- Ultimate Starter Kit > Settings > Save Settings
- Ultimate Starter Kit > Settings > Apply Settings

C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `SettingsItem` can now be used in any of your C++ files:

```
#include "USK/Settings/SettingsItem.h"

void ATestActor::Test()
{
    // SettingsItem is a pointer to the USettingsItem
    SettingsItem->ConfigureMenuItem(Config, Settings, MenuItem);
    USettingsData* SaveSettingsValue = SettingsItem->SaveSettings(Settings, MenuItem);
    SettingsItem->ApplySettings(World, Config, Settings);
}
```

15.4 Utils

15.4.1 Introduction

A Blueprint Function Library class used to load, save and apply all settings

15.4.2 Dependencies

The `SettingsUtils` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience
- Game Instance: Used to monitor for input device changes and handle saving/loading game data

15.4.3 Requirements

It's important that you configure the settings in the Game Instance before you can use the settings feature

15.4.4 Controls Settings

Before you can use the plugin to automatically handle the controls settings, you need to configure your Input Mapping Context. You are required to specify the `Name` for each key that can be changed through the plugin. This `Name` should match the value you specify in your `Menu Item`

NB: This feature is only available on Unreal Engine 5 and newer

15.4.5 API Reference

Functions

Name	Description	Params	Return
Initialize	Initialize the settings	**GameInstance (UUSKGameInstance*)** A reference to the game instance	
LoadSettings	Load the settings		**USettingsData*** The loaded settings data
SaveSettings	Save the settings	**Settings (USettingsData*)** The updated settings data	
ApplySettingsInWorld	Apply the settings	**World (UObject*)** The world context **Settings (USettingsData*)** The settings data to apply	
ApplySettings	Apply the settings	**GameInstance (UUSKGameInstance*)** A reference to the game instance **Settings (USettingsData*)** The settings data to apply	
ConfigureMenuItem	Configure the menu item to manage the specified settings item	**MenuItem (UMenuItem*)** The menu item to configure	
SaveMenuItemSettings	Save the settings managed by the menu item	**MenuItem (UMenuItem*)** The menu item containing the updated settings **ApplySettings (bool)** Should the settings also be applied?	
ApplyMenuItemSettings	Apply the settings managed by the menu item	**MenuItem (UMenuItem*)** The menu item containing the updated settings	
GetSettingsItemForMenuItem	Get the settings item for the specified menu item	**MenuItem (UMenuItem*)** The menu item to get the settings item for **Config (USettingsConfig*)** The settings config specified in the game instance	**USettingsItem*** The settings item
GetSettingsItem	Get the settings item for the specified settings item type	**SettingsItemType (ESettingsItemType)** The menu item to get the settings item for **Config (USettingsConfig*)** The settings config specified in the game instance	**USettingsItem*** The settings item

15.4.6 Blueprint Usage

You can use the `SettingsUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Settings > Initialize
- Ultimate Starter Kit > Settings > Load Settings
- Ultimate Starter Kit > Settings > Save Settings
- Ultimate Starter Kit > Settings > Apply Settings In World
- Ultimate Starter Kit > Settings > Apply Settings
- Ultimate Starter Kit > Settings > Configure Menu Item
- Ultimate Starter Kit > Settings > Save Menu Item Settings
- Ultimate Starter Kit > Settings > Apply Menu Item Settings
- Ultimate Starter Kit > Settings > Get Settings Item For Menu Item
- Ultimate Starter Kit > Settings > Get Settings Item

15.4.7 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `SettingsUtils` can now be used in any of your C++ files:

```
#include "USK/Settings/SettingsUtils.h"

void ATestActor::Test()
{
    USettingsUtils::Initialize(GameInstance);
    USettingsData* LoadSettingsValue = USettingsUtils::LoadSettings();
    USettingsUtils::SaveSettings(Settings);
    USettingsUtils::ApplySettingsInWorld(World, Settings);
    USettingsUtils::ApplySettings(GameInstance, Settings);
    USettingsUtils::ConfigureMenuItem(MenuItem);
    USettingsUtils::SaveMenuItemSettings(MenuItem, ApplySettings);
    USettingsUtils::ApplyMenuItemSettings(MenuItem);
    USettingsItem* SettingsItemForMenuItem = USettingsUtils::GetSettingsItemForMenuItem(MenuItem, Config);
    USettingsItem* SettingsItem = USettingsUtils::GetSettingsItem(SettingsItemType, Config);
}
```

16. Utils

16.1 Config Utils

16.1.1 Introduction

A Blueprint Function Library class used to extract config values

16.1.2 Dependencies

The `ConfigUtils` relies on other components of this plugin to work:

- Logger: Used to log useful information to help you debug any issues you might experience

16.1.3 API Reference

Functions

Name	Description	Params	Return
GetConfigValue	Extract a config value from a given config file	**Filename (FString)** The name of the config file **Section (FString)** The section in the config file **Key (FString)** The key in the config file **DefaultValue (FString)** The default value to return if the config file can't be read	**FString** The value extracted from the config file
GetGameConfigValue	Extract a config value from the default game config file	**Section (FString)** The section in the config file **Key (FString)** The key in the config file **DefaultValue (FString)** The default value to return if the config file can't be read	**FString** The value extracted from the config file

16.1.4 Blueprint Usage

You can use the `ConfigUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Utils > Config > Get Config Value
- Ultimate Starter Kit > Utils > Config > Get Game Config Value

16.1.5 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `ConfigUtils` can now be used in any of your C++ files:

```
#include "USK/Utils/ConfigUtils.h"

void ATestActor::Test()
{
    FString ConfigValue = UConfigUtils::GetConfigValue(Filename, Section, Key, DefaultValue);
    FString GameConfigValue = UConfigUtils::GetGameConfigValue(Section, Key, DefaultValue);
}
```

16.2 Platform

16.2.1 Platform Type

Introduction

The types of supported platform types

Values

Value	Description
Unknown	An unknown or unsupported platform
Windows	Windows (any architecture)
MacOS	MacOS (any architecture)
Linux	Linux (any architecture)
ConsoleMx	Console MX
ConsoleSp	Console SP
ConsoleNs	Console NS
Android	Android (any architecture)
IOS	iOS

16.2.2 Platform Utils

Introduction

A Blueprint Function Library class used for platform detection

API Reference

FUNCTIONS

Name	Description	Params	Return
GetPlatform	Get the current platform		**EPlatform** The current platform
IsInEditor	Is the build running inside the editor?		**bool** A boolean value indicating if the build is running inside the editor
IsDesktop	Is the build running on a desktop platform?		**bool** A boolean value indicating if the build is running on a desktop platform
IsWindows	Is the build running on Windows?		**bool** A boolean value indicating if the build is running on Windows
IsMacOS	Is the build running on MacOS?		**bool** A boolean value indicating if the build is running on MacOS
IsMacOSx86	Is the build running on MacOS (x86)?		**bool** A boolean value indicating if the build is running on MacOS (x86)
IsMacOSArm	Is the build running on MacOS (ARM)?		**bool** A boolean value indicating if the build is running on MacOS (ARM)
IsLinux	Is the build running on Linux?		**bool** A boolean value indicating if the build is running on Linux
IsConsole	Is the build running on a console platform?		**bool** A boolean value indicating if the build is running on a console platform
IsConsoleMx	Is the build running on Console MX?		**bool** A boolean value indicating if the build is running on Console MX
IsConsoleSp	Is the build running on Console SP?		**bool** A boolean value indicating if the build is running on Console SP
IsConsoleNs	Is the build running on Console NS?		**bool** A boolean value indicating if the build is running on Console NS
IsMobile	Is the build running on a mobile platform?		**bool** A boolean value indicating if the build is running on a mobile platform
IsAndroid	Is the build running on Android?		**bool** A boolean value indicating if the build is running on Android
IsAndroidx86	Is the build running on Android (x86)?		**bool** A boolean value indicating if the build is running on Android (x86)
IsAndroidx64	Is the build running on Android (x64)?		**bool** A boolean value indicating if the build is running on Android (x64)

IsAndroidArm	Is the build running on Android (ARM)?		**bool** A boolean value indicating if the build is running on Android (ARM)
IsAndroidArm64	Is the build running on Android (ARM64)?		**bool** A boolean value indicating if the build is running on Android (ARM64)
IsIOS	Is the build running on iOS?		**bool** A boolean value indicating if the build is running on iOS

Blueprint Usage

You can use the `PlatformUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Utils > Platform > Get Platform
- Ultimate Starter Kit > Utils > Platform > Is In Editor
- Ultimate Starter Kit > Utils > Platform > Is Desktop
- Ultimate Starter Kit > Utils > Platform > Is Windows
- Ultimate Starter Kit > Utils > Platform > Is MacOS
- Ultimate Starter Kit > Utils > Platform > Is MacOS (x86)
- Ultimate Starter Kit > Utils > Platform > Is MacOS (ARM)
- Ultimate Starter Kit > Utils > Platform > Is Linux
- Ultimate Starter Kit > Utils > Platform > Is Console
- Ultimate Starter Kit > Utils > Platform > Is Console MX
- Ultimate Starter Kit > Utils > Platform > Is Console SP
- Ultimate Starter Kit > Utils > Platform > Is Console NS
- Ultimate Starter Kit > Utils > Platform > Is Mobile
- Ultimate Starter Kit > Utils > Platform > Is Android
- Ultimate Starter Kit > Utils > Platform > Is Android (x86)
- Ultimate Starter Kit > Utils > Platform > Is Android (x64)
- Ultimate Starter Kit > Utils > Platform > Is Android (ARM)
- Ultimate Starter Kit > Utils > Platform > Is Android (ARM64)
- Ultimate Starter Kit > Utils > Platform > Is iOS

C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `PlatformUtils` can now be used in any of your C++ files:

```
#include "USK/Utils/PlatformUtils.h"

void ATestActor::Test()
{
    EPlatform Platform = UPlatformUtils::GetPlatform();
    bool IsInEditorValue = UPlatformUtils::IsInEditor();
    bool IsDesktopValue = UPlatformUtils::IsDesktop();
    bool IsWindowsValue = UPlatformUtils::IsWindows();
    bool IsMacOSValue = UPlatformUtils::IsMacOS();
    bool IsMacOSx86Value = UPlatformUtils::IsMacOSx86();
    bool IsMacOSArmValue = UPlatformUtils::IsMacOSArm();
    bool IsLinuxValue = UPlatformUtils::IsLinux();
    bool IsConsoleValue = UPlatformUtils::IsConsole();
    bool IsConsoleMxValue = UPlatformUtils::IsConsoleMx();
    bool IsConsoleSpValue = UPlatformUtils::IsConsoleSp();
    bool IsConsoleNsValue = UPlatformUtils::IsConsoleNs();
```

```
    bool IsMobileValue = UPlatformUtils::IsMobile();
    bool IsAndroidValue = UPlatformUtils::IsAndroid();
    bool IsAndroidx86Value = UPlatformUtils::IsAndroidx86();
    bool IsAndroidx64Value = UPlatformUtils::IsAndroidx64();
    bool IsAndroidArmValue = UPlatformUtils::IsAndroidArm();
    bool IsAndroidArm64Value = UPlatformUtils::IsAndroidArm64();
    bool IsIOSValue = UPlatformUtils::IsIOS();
}
```

16.3 Project Utils

16.3.1 Introduction

A Blueprint Function Library class used to extract project values

16.3.2 API Reference

Functions

Name	Description	Params	Return
GetProjectId	Get the project ID from the game config file		**FString** The project ID
GetProjectName	Get the project name from the game config file		**FString** The project name
GetProjectDescription	Get the project description from the game config file		**FString** The project description
GetProjectVersion	Get the project version from the game config file		**FString** The project version
GetProjectCompanyName	Get the project company name from the game config file		**FString** The project company name
GetProjectCopyrightNotice	Get the project copyright notice from the game config file		**FString** The project copyright notice
GetProjectLicensingTerms	Get the project licensing terms from the game config file		**FString** The project licensing terms
GetProjectHomepage	Get the project homepage from the game config file		**FString** The project homepage

16.3.3 Blueprint Usage

You can use the `ProjectUtils` using Blueprints by adding one of the following nodes:

- Ultimate Starter Kit > Utils > Project > Get Project Id
- Ultimate Starter Kit > Utils > Project > Get Project Name
- Ultimate Starter Kit > Utils > Project > Get Project Description
- Ultimate Starter Kit > Utils > Project > Get Project Version
- Ultimate Starter Kit > Utils > Project > Get Project Company Name
- Ultimate Starter Kit > Utils > Project > Get Project Copyright Notice
- Ultimate Starter Kit > Utils > Project > Get Project Licensing Terms
- Ultimate Starter Kit > Utils > Project > Get Project Homepage

16.3.4 C++ Usage

Before you can use the plugin, you first need to enable the plugin in your `Build.cs` file:

```
PublicDependencyModuleNames.Add("USK");
```

The `ProjectUtils` can now be used in any of your C++ files:

```
#include "USK/Utils/ProjectUtils.h"

void ATestActor::Test()
{
    FString ProjectId = UProjectUtils::GetProjectId();
    FString ProjectName = UProjectUtils::GetProjectName();
    FString ProjectDescription = UProjectUtils::GetProjectDescription();
    FString ProjectVersion = UProjectUtils::GetProjectVersion();
    FString ProjectCompanyName = UProjectUtils::GetProjectCompanyName();
    FString ProjectCopyrightNotice = UProjectUtils::GetProjectCopyrightNotice();
    FString ProjectLicensingTerms = UProjectUtils::GetProjectLicensingTerms();
    FString ProjectHomepage = UProjectUtils::GetProjectHomepage();
}
```