Developing for Sony PlayStation 4 with Unreal Engine

```
#pragma once

#include "GameFramework/Actor.h"
#include "MyActor.generated.h"


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

public:

    // Sets default values for this actor's properties
    AMyActor();

    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

    // Called every frame
    virtual void Tick( float DeltaSeconds ) override;

};
```

Required Setup

Before you can get started developing for the PlayStation 4, you'll have to install your PS4 SDK, setup a source build of the engine, and install the requisite PS4 support files for integration into your build.

1. Download the Unreal Engine 4 source code version of the engine available via GitHub .

   The Binary version of the engine available in the Epic Games Launcher does not support console development.

2. Install the PS4 SDK and make sure that the SDK version matches the value assigned to **PS4PlatformSDK::ExpectedSDKVersion** that is located in:

   `[UE4Directory]/Engine/Source/Programs/UnrealBuildTool/PS4/UEBuildPS4.cs`

   If you upgrade your SDK, you will need to be sure to update this value as well.

   In order to see which version of the PS4 SDK Unreal Engine is expecting, you can look in the `HasRequiredSDKsInstalled()` function in the file below:

   `[UE4Directory]/Engine/Source/Programs/UnrealBuildTool/PS4/UEBuildPS4.cs`

   Refer to your project version Release and HotFix notes for latest supported or recommended Platform SDKs.

3. Download the available PS4 Support files by logging into http://ftp.epicgames.com/ with the FTP Credentials you received from Epic Games when you registered as a PS4 developer and navigating to the FTP's **UE4PS4** directory and download the **PS4-xxxx.zip** where "xxxx" matches the GitHub branch you are synced with.

4. Run the `setup.bat` and extract the **PS4-xxxx.zip** into your `[UE4Directory]` root folder.

5. Run `[UE4Directory]/GenerateProjectFiles.bat`. This step should be performed after installing the PS4 SDK so that it will know you have the necessary PS4 Support files installed.

6. Compile the Editor, the related tools, and your project. The following targets are a good starting point.

Target	Config	Platform

[ProjectName]	Development Editor	Win64
PS4DevKitUtil	Development Editor	Win64
ShaderCompileWorker	Development Editor	Win64
UnrealFrontend	Development Editor	Win64
[ProjectName]	Development	PS4

Alternatively, to make building these projects easier, you can use the **Batch Build** functionality that is available in Visual Studio by performing the following steps:

1. In the **Build** menu, choose **Build > Batch Build**.
2. Setup the following options:

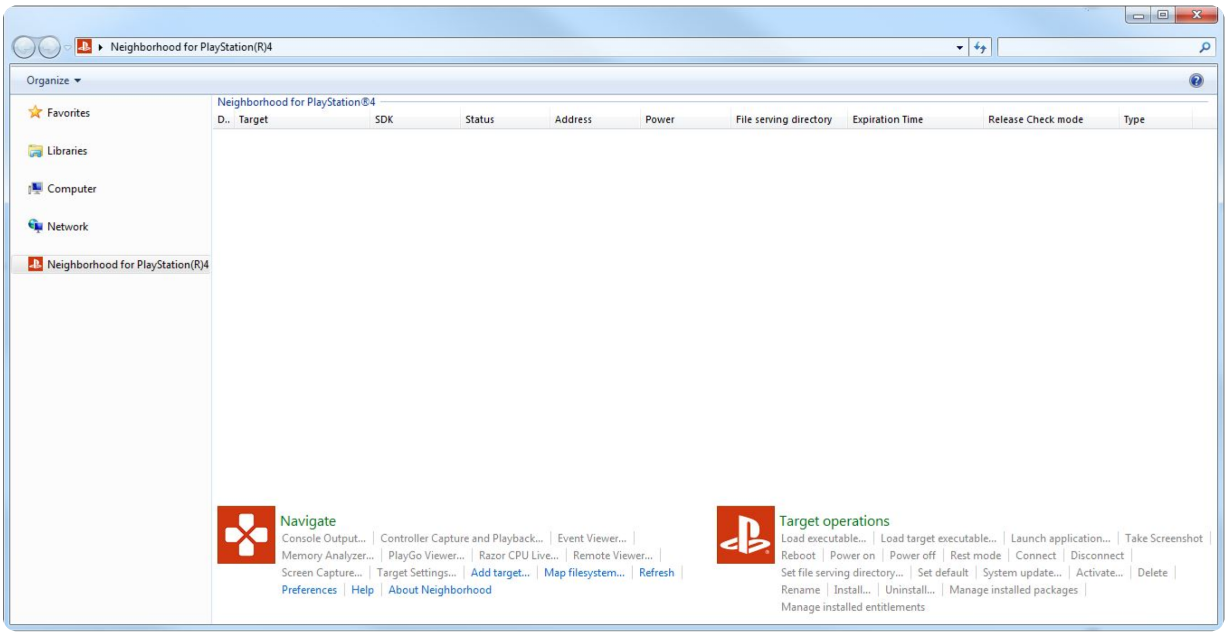
Target	Config	Platform
[ProjectName]	Development_Editor	x64
PS4DevKitUtil	Development Editor	Win64
ShaderCompileWorker	Development_Editor	x64
UnrealFrontend	Development Editor	x64
[ProjectName]	Development	ORBIS

3. Click the **Build** or **Rebuild** button to compile the Editor, related tools, and your project.

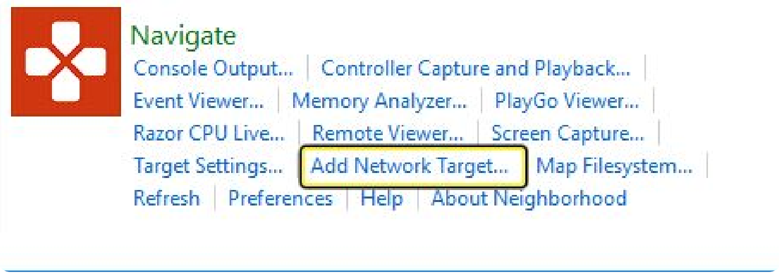
Connecting to your PS4 Dev Kit

Before you can use the PS4 Neighborhood or Device Manager to connect to your PS4 you will first need to add your Dev Kit to the list of available targets, which has to initially be set up from Neighborhood. After which, you can choose the connection method that best fits your workflow.

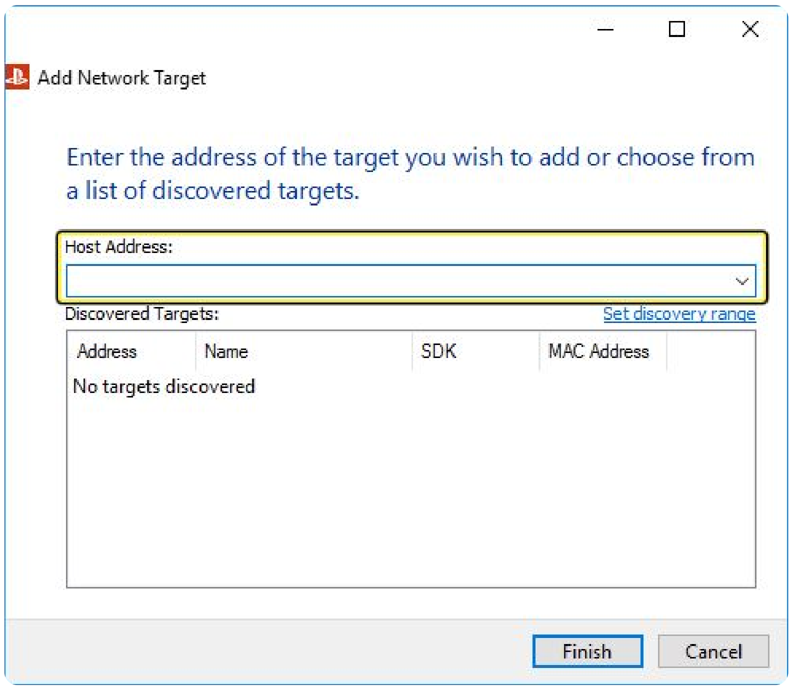
Adding your PS4 to Neighborhood


Neighborhood for PlayStation(R)4
Organize
Favorites
Libraries
Computer
Network
Neighborhood for PlayStation(R)4
Neighborhood for PlayStation®4
D.. Target SDK Status Address Power File serving directory Expiration Time Release Check mode Type
Navigate
Console Output... | Controller Capture and Playback... | Event Viewer... |
Memory Analyzer... | PlayGo Viewer... | Razor CPU Live... | Remote Viewer... |
Screen Capture... | Target Settings... | Add target... | Map filesystem... | Refresh |
Preferences | Help | About Neighborhood
Target operations
Load executable... | Load target executable... | Launch application... | Take Screenshot |
Reboot | Power on | Power off | Rest mode | Connect | Disconnect |
Set file serving directory... | Set default | System update... | Activate... | Delete |
Rename | Install... | Uninstall... | Manage installed packages |
Manage installed entitlements

1. Start by opening up the **Neighborhood for PlayStation 4** application and then in the **Navigate** section, click on **Add Network Target**.


Navigate
Console Output... | Controller Capture and Playback... |
Event Viewer... | Memory Analyzer... | PlayGo Viewer... |
Razor CPU Live... | Remote Viewer... | Screen Capture... |
Target Settings... | Add Network Target... | Map Filesystem... |
Refresh | Preferences | Help | About Neighborhood

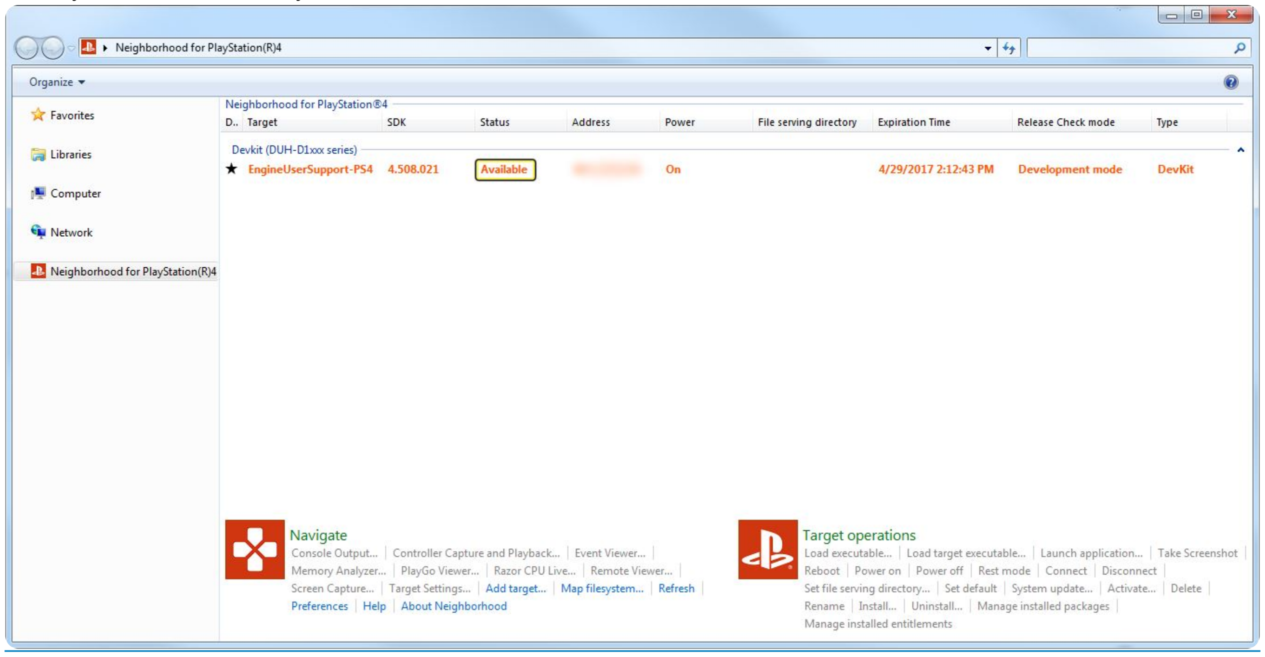
2. Now, the Network Target window will open. Here you can select any discovered targets or specify a **Host Address** to "discover" your Dev Kit on the network.



Enter the address of the target you wish to add or choose from a list of discovered targets.

Address	Name	SDK	MAC Address
No targets discovered			

3. Once you've "discovered" your PS4 Dev Kit, the **Status** column should list whether it's "available", "unavailable", or "connected".


Neighborhood for PlayStation(R)4
Organize
Favorites
Libraries
Computer
Network
Neighborhood for PlayStation(R)4
Navigate
Console Output... | Controller Capture and Playback... | Event Viewer... |
Memory Analyzer... | PlayGo Viewer... | Razor CPU Live... | Remote Viewer... |
Screen Capture... | Target Settings... | Add target... | Map filesystem... | Refresh |
Preferences | Help | About Neighborhood
Target operations
Load executable... | Load target executable... | Launch application... | Take Screenshot |
Reboot | Power on | Power off | Rest mode | Connect | Disconnect |
Set file serving directory... | Set default | System update... | Activate... | Delete |
Rename | Install... | Uninstall... | Manage installed packages |
Manage installed entitlements

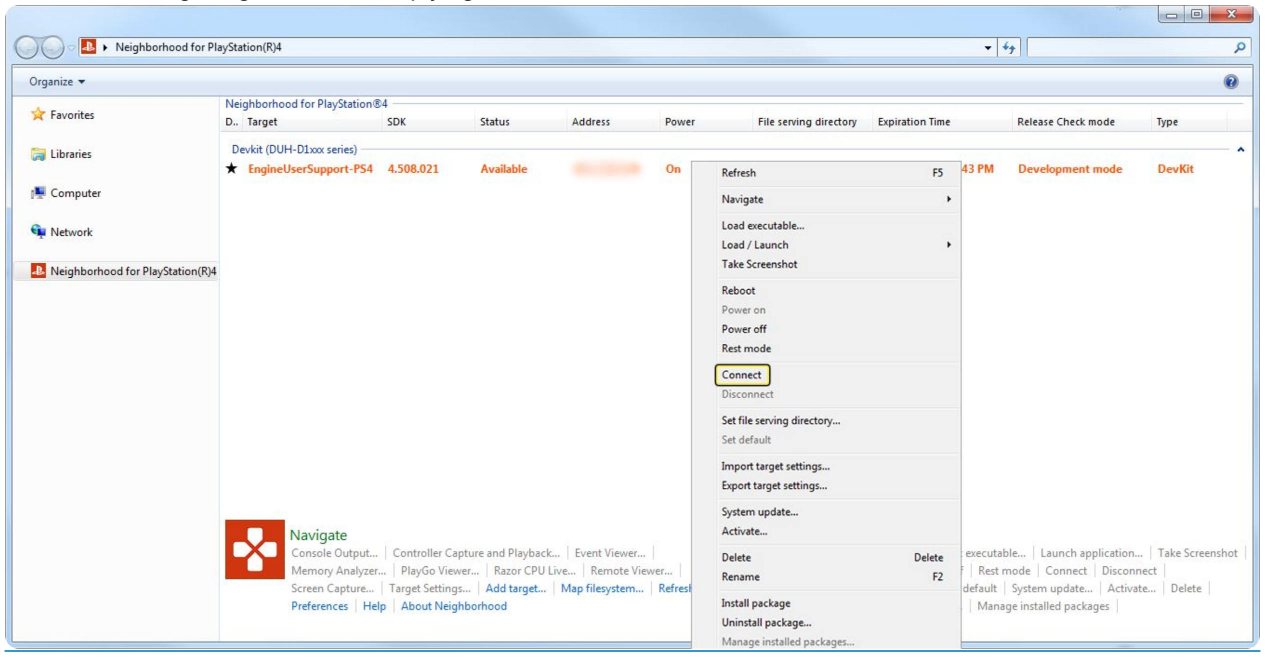
Neighborhood for PlayStation®4

D..	Target	SDK	Status	Address	Power	File serving directory	Expiration Time	Release Check mode	Type
Devkit (DUH-D1xxx series)									
★	EngineUserSupport-PS4	4.508.021	Available	[redacted]	On		4/29/2017 2:12:43 PM	Development mode	DevKit

Click for full image.

Using Neighborhood to Connect

1. To connect using Neighborhood, simply right-click on the listed Dev Kit and select **Connect** from the menu.



Neighborhood for PlayStation(R)4
Organize
Favorites
Libraries
Computer
Network
Neighborhood for PlayStation(R)4
Refresh F5
Navigate
Load executable...
Load / Launch
Take Screenshot
Reboot
Power on
Power off
Rest mode
Connect
Disconnect
Set file serving directory...
Set default
Import target settings...
Export target settings...
System update...
Activate...
Delete Delete
Rename F2
Install package
Uninstall package...
Manage installed packages...
Navigate
Console Output... | Controller Capture and Playback... | Event Viewer...
Memory Analyzer... | PlayGo Viewer... | Razor CPU Live... | Remote Viewer...
Screen Capture... | Target Settings... | Add target... | Map filesystem... | Refresh
Preferences | Help | About Neighborhood
executable... | Launch application... | Take Screenshot
Rest mode | Connect | Disconnect
default | System update... | Activate... | Delete
Manage installed packages

Neighborhood for PlayStation®4

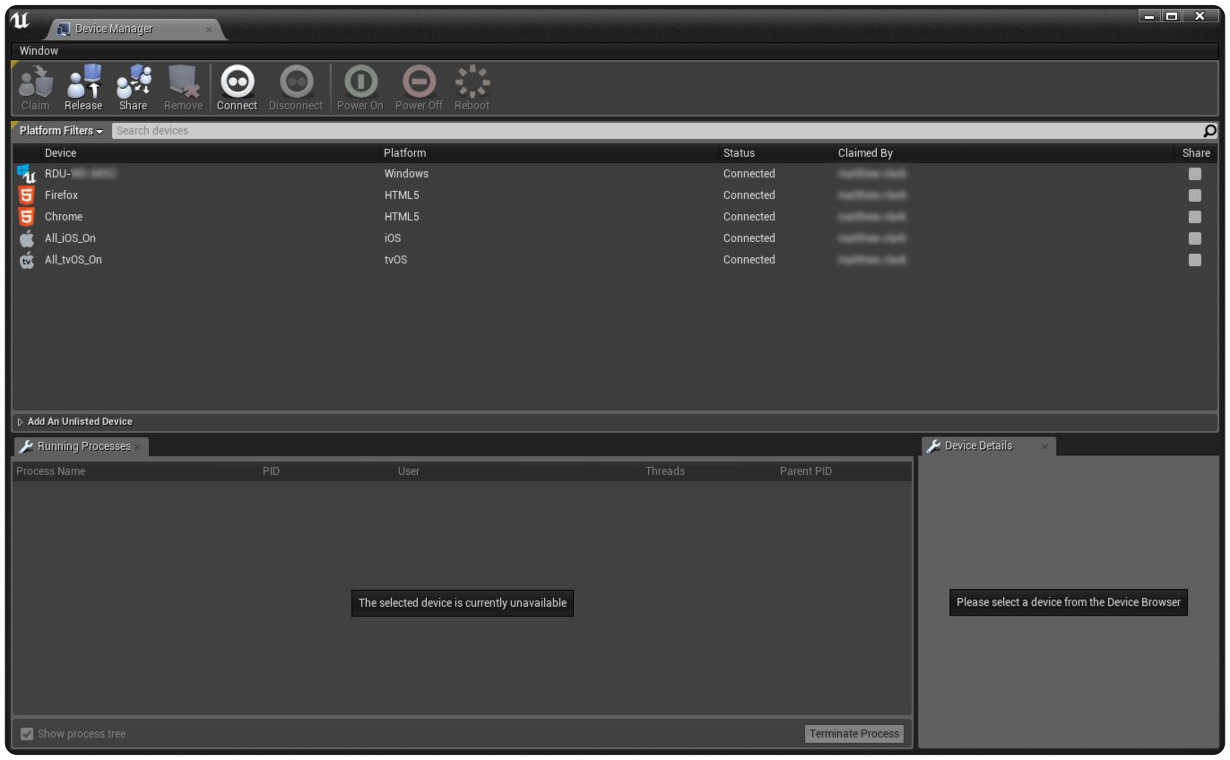
Devkit (DUH-D1xxx series)

D..	Target	SDK	Status	Address	Power	File serving directory	Expiration Time	Release Check mode	Type
★	EngineUserSupport-PS4	4.508.021	Available	[redacted]	On		43 PM	Development mode	DevKit

Click for full image.

For additional information about Neighborhood tool, please consult Sony Documentation .

Using UE4's Device Manager to Connect



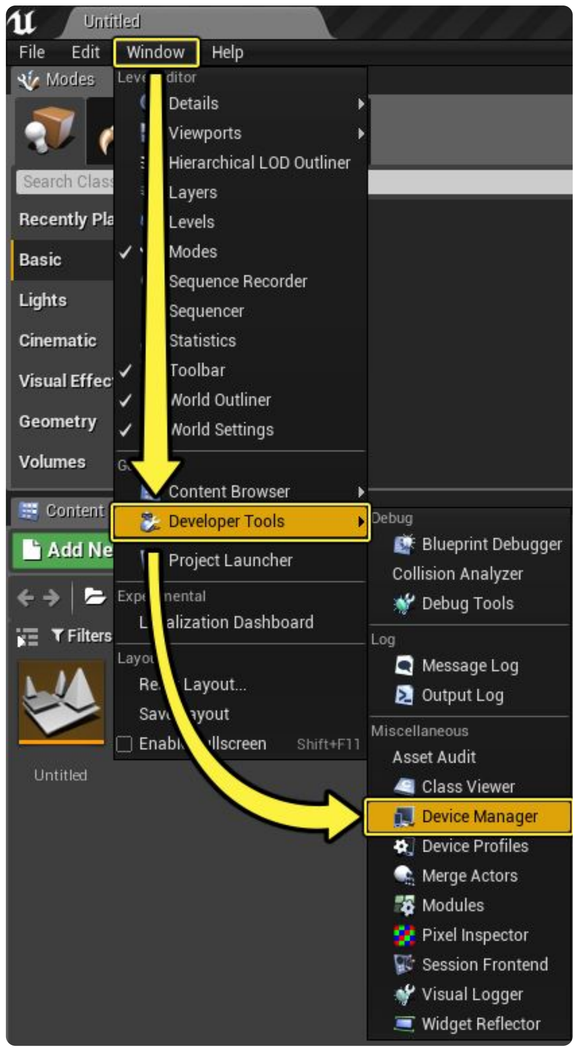
Device	Platform	Status	Claimed By	Share
RDU-[redacted]	Windows	Connected	[redacted]	
Firefox	HTML5	Connected	[redacted]	
Chrome	HTML5	Connected	[redacted]	
All_iOS_On	iOS	Connected	[redacted]	
All_tvOS_On	tvOS	Connected	[redacted]	

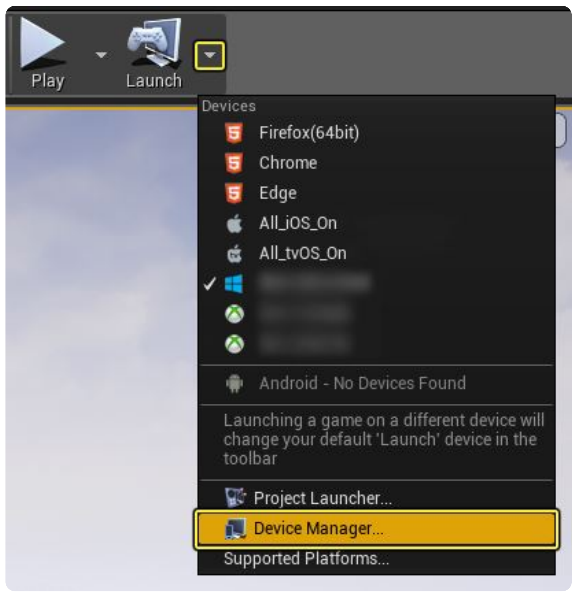
Process Name	PID	User	Threads	Parent PID

The selected device is currently unavailable

Please select a device from the Device Browser

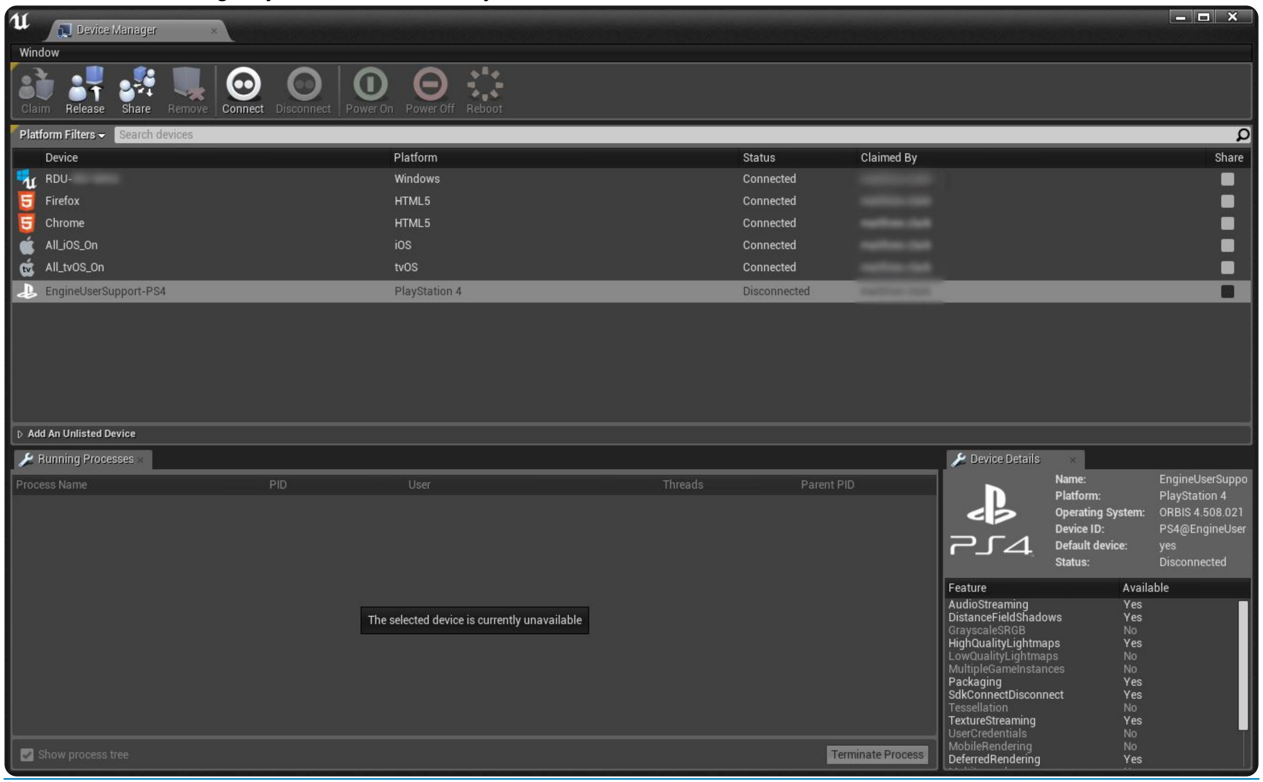
1. With the UE4 Editor opened you can access the Device manager in one of two ways; navigate to the main menu > **Window** > **Developer Tools** > and select **Device Manager**, or you can use the main toolbar > click the arrow next to **Launch** > and select **Device Manager** from the menu list.




Play
Launch
Devices
Firefox(64bit)
Chrome
Edge
All_iOS_On
All_tvOS_On
Android - No Devices Found
Launching a game on a different device will change your default 'Launch' device in the toolbar
Project Launcher...
Device Manager...
Supported Platforms...

Main Menu **Main Toolbar**

2. In the Device Manager, you will see a list of your available Platforms. Your PS4 should be listed in the device list.



Device	Platform	Status	Claimed By	Share
RDU-[redacted]	Windows	Connected	[redacted]	
Firefox	HTML5	Connected	[redacted]	
Chrome	HTML5	Connected	[redacted]	
All_iOS_On	iOS	Connected	[redacted]	
All_tvOS_On	tvOS	Connected	[redacted]	
EngineUserSupport-PS4	PlayStation 4	Disconnected	[redacted]	

Process Name	PID	User	Threads	Parent PID

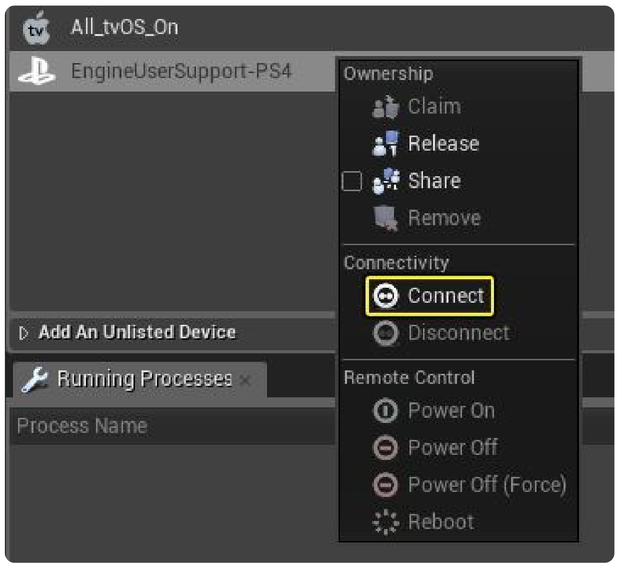
The selected device is currently unavailable

Name:	EngineUserSuppo
Platform:	PlayStation 4
Operating System:	ORBIS 4.508.021
Device ID:	PS4@EngineUser
Default device:	yes
Status:	Disconnected

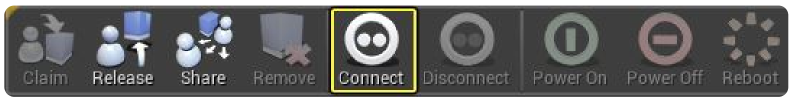
Feature	Available
AudioStreaming	Yes
DistanceFieldShadows	Yes
GrayscaleSRGB	No
HighQualityLightmaps	Yes
LowQualityLightmaps	No
MultipleGameInstances	No
Packaging	Yes
SdkConnectDisconnect	Yes
Tessellation	No
TextureStreaming	Yes
UserCredentials	No
MobileRendering	No
DeferredRendering	Yes

Click for full image.

3. Select the PS4 device you want to connect to, then from the main toolbar you can select **Connect**, or you can right-click the device and use the context menu to select **Connect**.

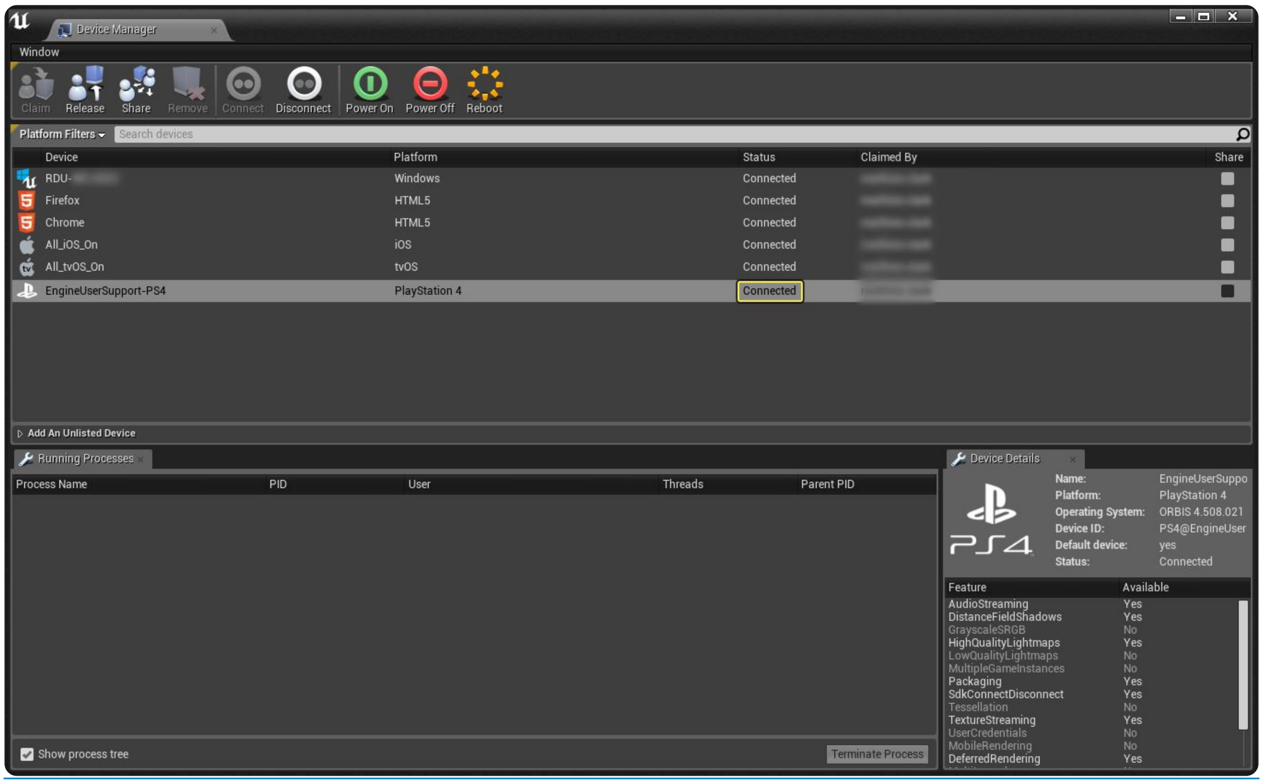


Right-Click Context Menu



Main Toolbar

Now you should see "Connected" listed in the **Status** column next to PS4 in the device list.



Device	Platform	Status	Claimed By	Share
RDU-[redacted]	Windows	Connected	[redacted]	
Firefox	HTML5	Connected	[redacted]	
Chrome	HTML5	Connected	[redacted]	
All_iOS_On	iOS	Connected	[redacted]	
All_tvOS_On	tvOS	Connected	[redacted]	
EngineUserSupport-PS4	PlayStation 4	Connected	[redacted]	

Process Name	PID	User	Threads	Parent PID

Name:	EngineUserSuppo
Platform:	PlayStation 4
Operating System:	ORBIS 4.508.021
Device ID:	PS4@EngineUser
Default device:	yes
Status:	Connected

Feature	Available
AudioStreaming	Yes
DistanceFieldShadows	Yes
GrayscaleSRGB	No
HighQualityLightmaps	Yes
LowQualityLightmaps	No
MultipleGameInstances	No
Packaging	Yes
SdkConnectDisconnect	Yes
Tessellation	No
TextureStreaming	Yes
UserCredentials	No
MobileRendering	No
DeferredRendering	Yes

Click for full image.

Using Build Operations and Cooking Content

All Build Operations (build, cook, package, deploy, and run) are performed using Unreal Automation Tool . When using valid arguments this tool can be run on the command line by using the following:

```
[UE4Directory]/Engine/Build/BatchFiles/RunUAT.bat
```

When you use the Project Launcher (**Window** > **Project Launcher**) from the Editor, it will generate the command line on your behalf which it then automatically passes to `RunUAT.bat`.

Running a profile from the Project Launcher in the Editor will output the generated command line in the log. The text from **BuildCookRun** onward can be passed to `RunUAT.bat`.

Project Launcher Generated Log Output

```
Automation.ParseCommandLine: Parsing commandline: -ScriptsForProject=MyProject.uproject BuildCookRun -
project=MyProject.uproject -clientconfig=Development ...
```

Equivalent Command Line

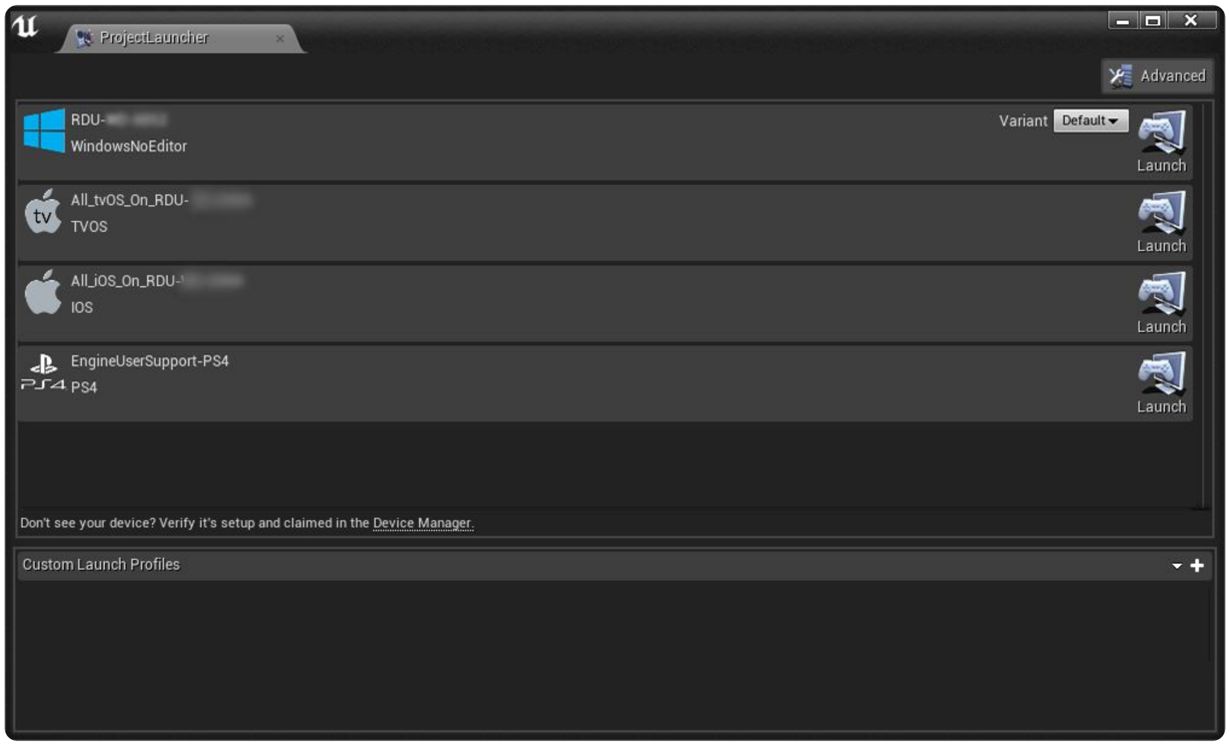
```
[UE4Directory]/Engine/Build/BatchFiles/RunUAT.bat BuildCookRun -project=MyProject.uproject -
clientconfig=Development
```

As a "Best Practice", the arguments supplied to UAT can be quite involved, therefore we recommend that you use a Custom Profile in the Project Launcher to make your builds. Even if you prefer to run UAT from a command line, the best route is to use the Project Launcher at least to generate the proper command line rather than attempting to author it by hand.

Cooking Content

Before you can use your content on your PS4 it must be cooked. Content Cooking refers to the files which have been compiled and converted to the final format which is most optimized for the target platform. You can do this right from the **Project Launcher** where you have two ways to go about cooking your content; **By the Book** or **on the fly**.

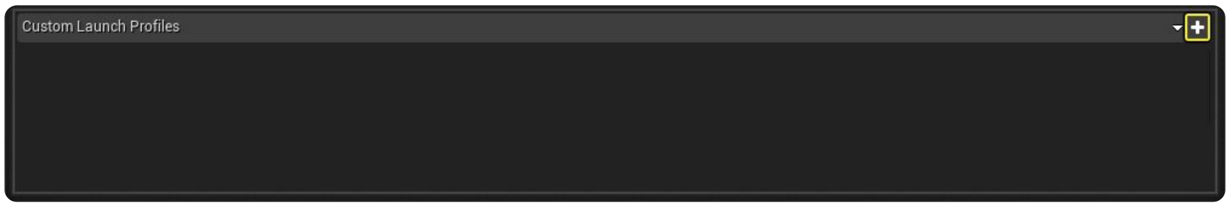
To open the Project Launcher, go to the main menu > **Window** > and select **Project Launcher**.



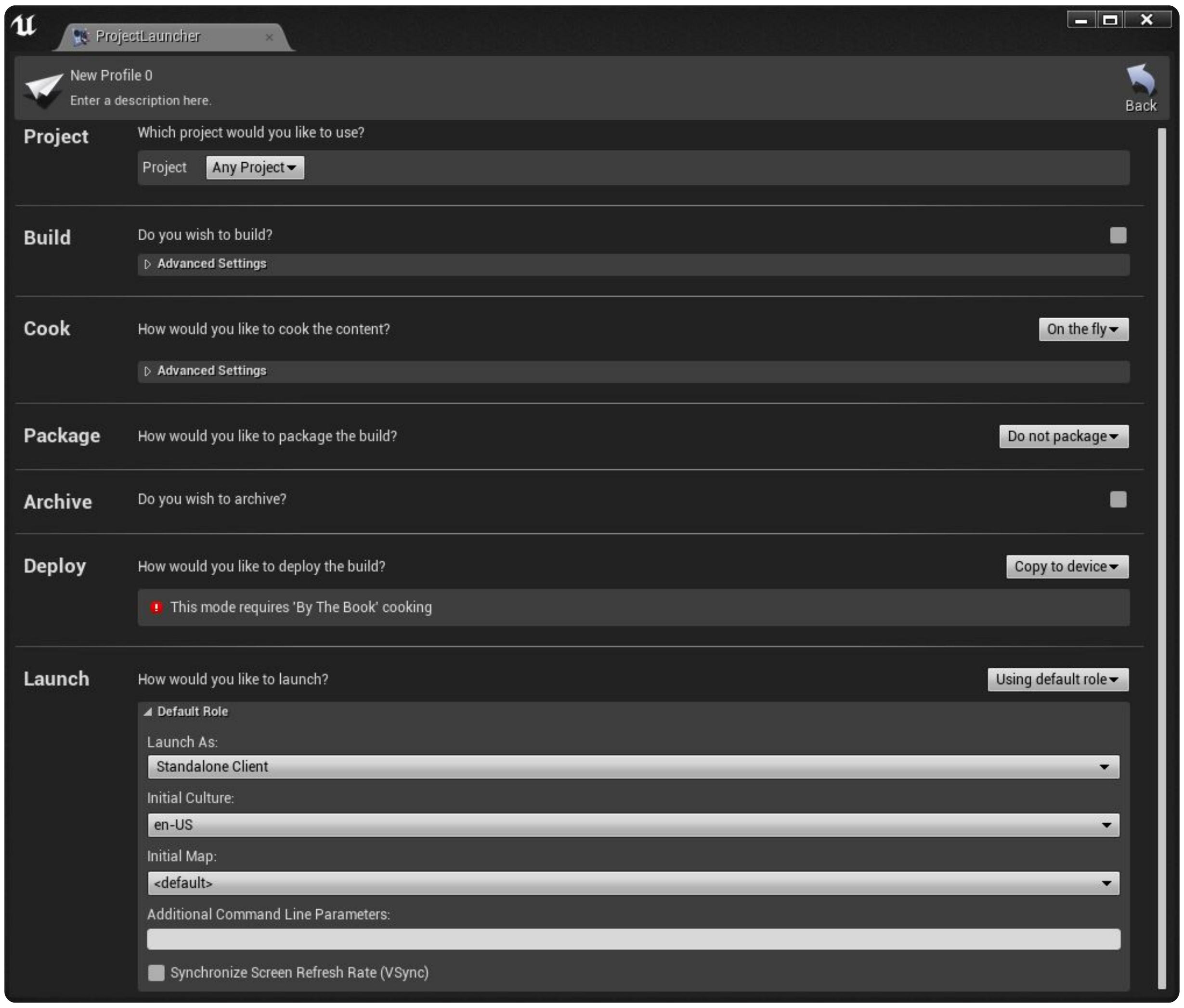
Before you can use these cook methods, you will first need to create a **Custom Launch Profile**.

Creating a Custom Launch Profile

1. In the Project Launcher window, click the **add** (**+**) sign in the **Custom Launch Profile** section.



2. In the same window, it will now display the options available for your new "Custom Launch Profile." Make sure to give your profile a **name** and assign a specific **project**, if necessary.



New Profile 0
Enter a description here.

Project — Which project would you like to use? Project: Any Project

Build — Do you wish to build? Advanced Settings

Cook — How would you like to cook the content? On the fly. Advanced Settings

Package — How would you like to package the build? Do not package

Archive — Do you wish to archive?

Deploy — How would you like to deploy the build? Copy to device. This mode requires 'By The Book' cooking

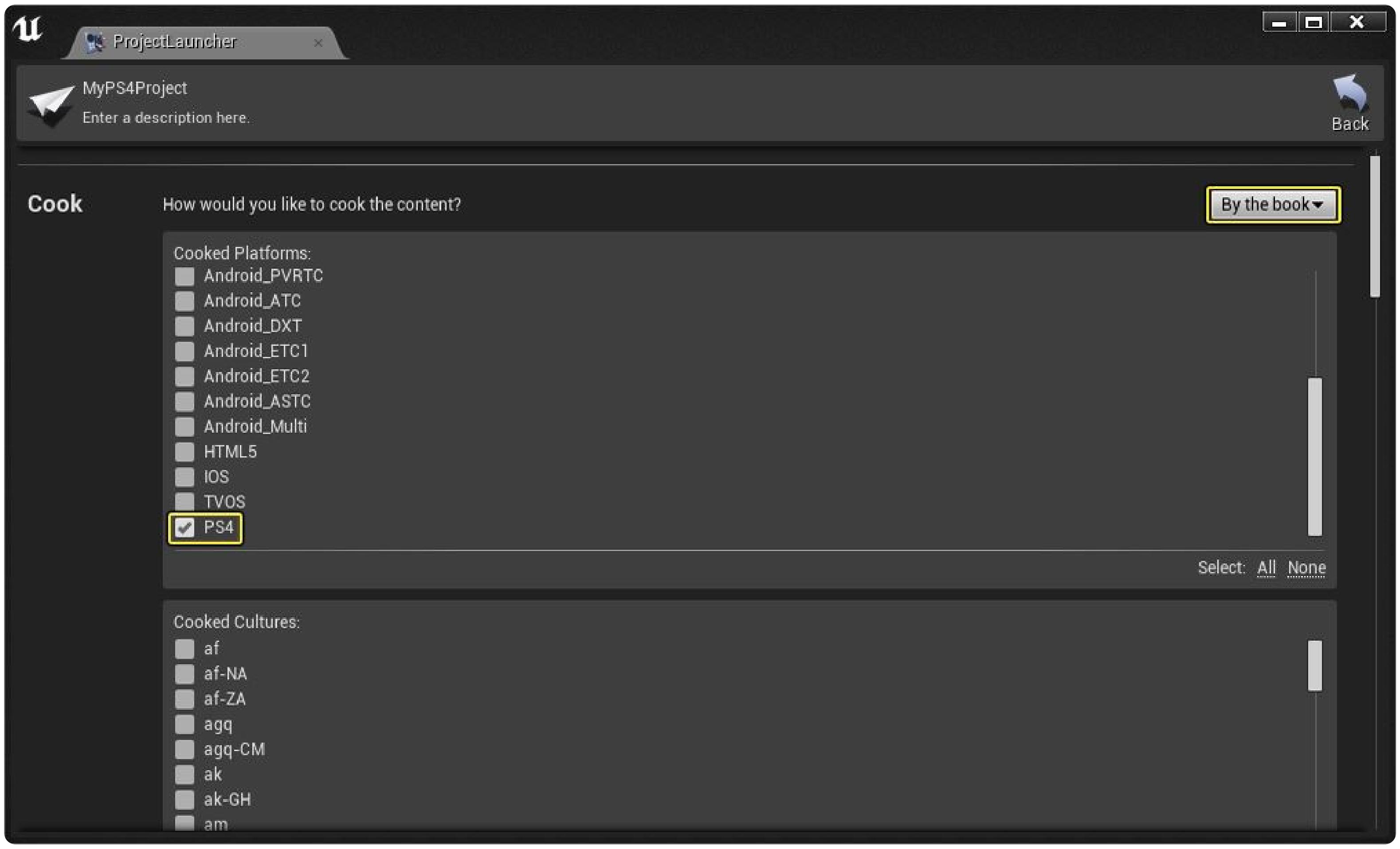
Launch — How would you like to launch? Using default role
Default Role
Launch As: Standalone Client
Initial Culture: en-US
Initial Map: <default>
Additional Command Line Parameters:
Synchronize Screen Refresh Rate (VSync)

3. Now that you've got your own Custom Launch Profile, choose between **Cook By the Book** and **Cook on the fly** below to determine which method best fits your work environment and deployment needs.

Cook By The Book

When you choose to cook content **By the Book** it means performing the entirety of the cook ahead of time, allowing the build to deploy the cooked assets all at once. This is useful if you're not iterating on individual assets or for those who want the game to perform at full speed without waiting for a server to deliver the cooked content. Typically performance testing and play tests will want to use this method.

There's no extra setup required to perform a cook by the book build. In the Project Launcher under your **Custom Launch Profile**, choose the **Cook** method from the drop-down selection to set **By The Book**.



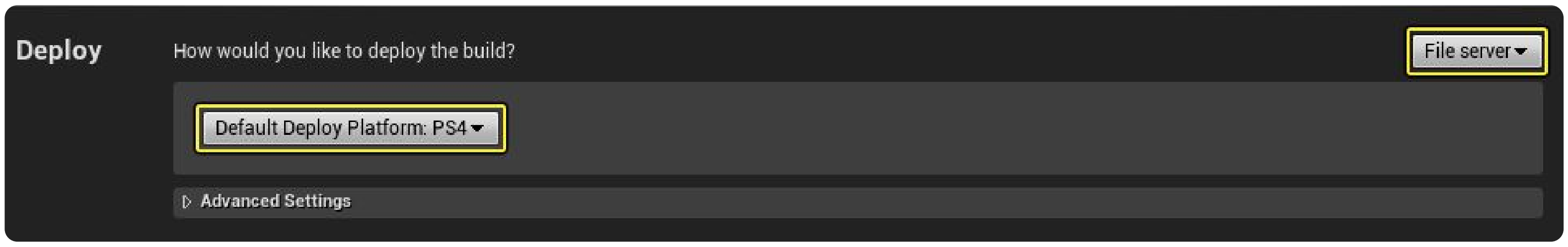
Cook On The Fly

When you choose to cook content **on the fly** it will delay cooking until after the game has been deployed. Only the executable and some other basic files are installed which use network communication with a Cook Server to make requests on-demand. Cook on the fly enables faster iteration for developers who will be making changes to content regularly or those who will be exploring sections of the game (which means they don't need to be cooked at all).

In order to cook on the fly, you will need to start a Cook Server on a machine which has the full project. This can be either the local machine, or a remote server which performs the cook. The Cook Server can be run by starting the UE4 Editor in command line mode as follows:

```
UE4Editor-cmd.exe [fullAbsolutePathToProject.uproject] -run=cook -targetplatform=PS4 -cookonthefly
```

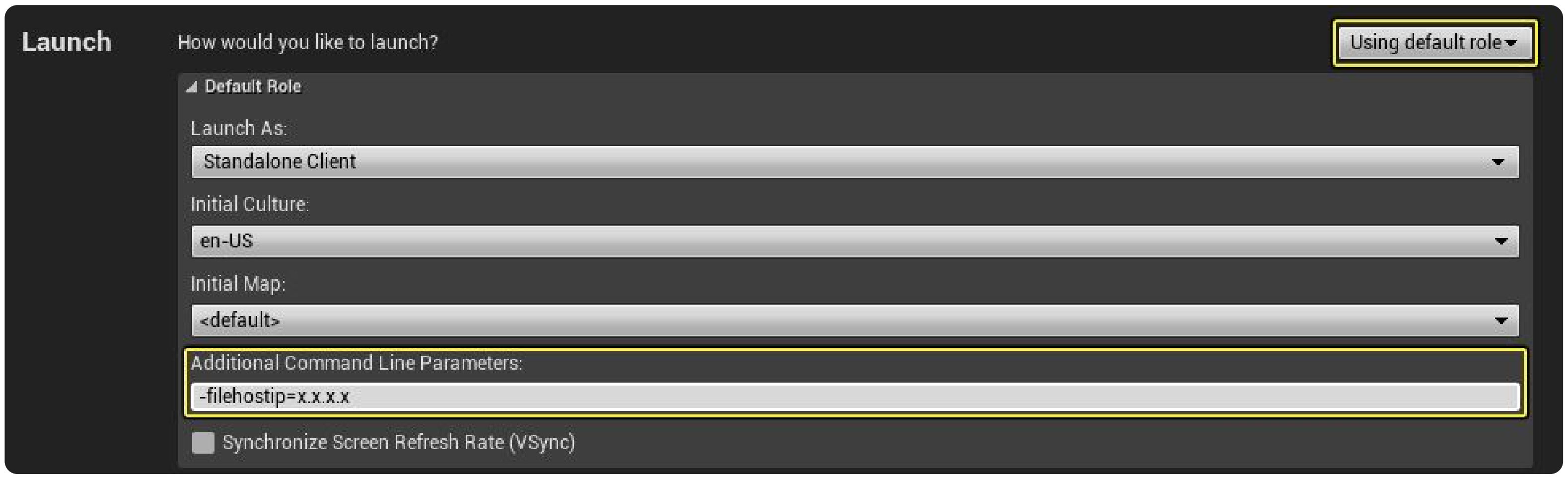
On the Developer's machine, in the Project Launcher under your Custom Launch Profile, choose the **Deploy** method from the drop-down selection to set **File Server**. Use the drop-down selection to set the target to **Default Deploy Platform: PS4**.


Deploy
How would you like to deploy the build?
File server
Default Deploy Platform: PS4
Advanced Settings

With the executable running on the PS4, it will need to be made aware of the IP address from the machine which is running the Cook Server. To do this pass the following command line argument on the client command line (where x.x.x.x represents your host's IP):

```
-filehostip=x.x.x.x
```

This can be specified in your Custom Launch Profile's **Launch** section under the **Additional Command Line Parameters**. The default is 0.0.0.0 (local host)



Launch As: Standalone Client
Initial Culture: en-US
Initial Map: <default>
Additional Command Line Parameters: -filehostip=x.x.x.x
Synchronize Screen Refresh Rate (VSync)

Using Visual Studio for Debugging

When you are debugging your game with Visual Studio there are two real variations to this; you can either be running your game or running the Editor, however, it is not recommended to do both at the same time. A game launched from an Editor which is already running within Visual Studio will be launched outside of the Visual Studio context, meaning that it will not have any debugger attached to it. You can have any number of instances of Visual Studio open however, which can be attached to any number of running processes. This makes it possible to launch the game from the Editor and then attach another instance of Visual Studio to the kit through the "Attach to Process" dialog.

Running the Game from Visual Studio

Running your game from Visual Studio will require that the assets be **cooked and deployed outside of Visual Studio**. The process will depend on whether you want to Cook By The Book, Cook On The Fly, or if you are skipping the cooking process altogether with an existing game package file.

Click to expand the method that best fits your workflow and follow the steps.

Run the Game with Cook by the Book

This method will cook the whole game on the developer's machine and deploy it to the console before beginning the debugging session. If you do not yet have cooked content and want to ensure the game runs at the best rate this is the method you want to choose.

1. Open the Editor, build, cook, and deploy the content onto the PS4 using a **Cook by the Book** profile in the Project Launcher.
2. Choose a build configuration such as **Development | PS4** or **Debug Game | PS4**.
3. In Visual Studio under the **Solution Explorer** window, right-click on the game project and select **Properties** > **Debugging**.
4. Next, set the Executable's Arguments to:
    - [RelativePathFromElfToProject.uproject] [MapName]
    - ../../../MyProject/MyProject.uproject myMap
5. Finally, **Build** and **Run** the project.

Things to keep in mind:

- The .uproject specified is relative to the .elf / .self file on the PS4 and not relative to the Visual Studio directory.

- Visual Studio does not deploy your game assets. They must already be in place by a previous cook and deployment.

Run the Game with Cook on the fly

This method will cook the whole game on the designated "Cook on the Fly" machine (either local or remote) and start the game up quickly. Runtime performance will suffer some as the assets are cooked on demand. This method is ideal when you do not have an existing cooked build and you only want to cook the minimum number of assets needed to run the game.

1. Ensure that the Cook on the Fly server is running.
2. Choose a build configuration such as **Development | PS4** or **Debug Game | PS4**.
3. In Visual Studio under the **Solution Explorer** window, right-click on the game project and select **Properties** > **Debugging**.
4. Set Executable Arguments to:
   - [RelativePathFromElfToProject.uproject] [MapName] [-filehostip=COTFServerAddress]
   - ../../../MyProject/MyProject.uproject myMap -filehostip=192.168.0.1

   The `-filehostip` should specify the Cook Server's IP address as seen by the PS4 hardware running the game.
5. Finally, **Build** and **Run** the project.

Run the Game with a Pre-built Package

This method installs a pre-existing fully packaged build to the PS4 and then runs debugging on top of that.

You can use this to debug binaries that are already in the package (.pkg) or to make code changes and then run those on top of the package. This workflow enables teams to debug a common build or to make code changes without having to re-cook and re-package the game.

1. Install the desired base package to the PS4.
2. Choose a build configuration such as **Development | PS4** or **Debug Game | PS4**.
3. In Visual Studio under the **Solution Explorer** window, right-click on the game project and select **Properties** > **Debugging**.
4. Change the **Debugger to launch** drop-down selection from "PS4 Standard Debugger" to **PS4 Application Debugger** and input values for the following fields.
   - **TitleID** - Change this to your correct TitleID.
   - **Executable Load Location** - Enter one of the following:
     - **Auto** - Specify to use the .self defined within the deployed package. Visual Studio will attach to the binary built into the package.
     - **Host** - Specify this to have Visual Studio load a .self binary file directly from your host PC. Enter the path to the .self in the **Host Executable** field which appears. To use a locally built binary for example you would use: `MyGameProject\Binaries\PS4\MyGameProject.self` Visual Studio would attach to the binary compiled on the local machine running on top of the deployed package.
     - **PS4** - This is the same as the "Host" option, except the path you specify in "PS4 Executable" is relative to the **/data/** directory on the PS4's hard disk. This folder is accessible via the network mapped dev kit folder in Windows Explorer. Visual Studio will launch and attach to the binary specified here running on top of the deployed package.
   - Optionally, add additional game command line arguments in the **Executable Arguments**.
5. Change "PS4 Standard Debugger" to **PS4 Application Debugger** in Visual Studio next to the **Play** (green) arrow.

6. To begin debugging you can use the shortcut **F5**.

> There are a few important things to remember when the "Running the Game from Visual Studio" method.
>
> - The .uproject file specified is relative to the .self file on the PS4 and not relative to a Visual Studio directory.
> - `-filehostip` is only used if you are using **Cook On The Fly** and should specify the Cook Server's IP address as seen by the hardware running the game.
> - Visual Studio does not deploy your game assets, so they must already be in place or a Cook On The Fly server must be used.

Running the Editor from Visual Studio

To run the Editor, itself, inside of Visual Studio, just build your game project with the desired "Editor" build configuration for Win64. For example **Development Editor** or **Debug Editor**.

You will not be able to use the Project Launcher from the Editor to run your game if that Editor is currently running in Visual Studio unless you disable **Edit and Continue**. To do this you can go to **Tools** > **Options** > **Debugging** > **General** > and uncheck the option for **Enable Edit and Continue**.

Additional Information

In this page you'll find additional information and quick references mentioned in the previous pages for setting up your PlayStation 4 Developer Kit for use with Unreal Engine 4.

Accessing the PlayStation 4 Support Files

1. Log into http://ftp.epicgames.com/ with the FTP Credentials you received from Epic Games when you registered as a PlayStation 4 developer.
2. Navigate to the FTP's **UE4PS4** directory and download the **PS4-xxxx.zip** where "xxxx" matches the GitHub branch you are synced with.

Setup Reminders

- If you're developing for other platforms, be sure to install all prerequisites for those platforms before generating your project files.
- Platform support is based on which SDKs are installed at the time the project files are generated. So, anytime you make an update to the SDK version, you'll need to run `[UE4Directory]/GenerateProjectFiles.bat`.
- When installing your SDK, make sure the SDK version matches the value assigned in **PS4PlatformSDK::ExpectedSDKVersion** located in your `[UE4Directory]/Engine/Source/Programs/UnrealBuildTool/PS4/UEBuildPS4.cs` file.

Batch Building with Visual Studio

To make the build process easier on yourself, you can use Batch Building to build multiple configurations of your project by using the **Batch Build** option from the **Build** menu.

You can see an example of how this is used in Step 6 of the Required Setup page.

Additional Resources

- Use the Release and HotFix Notes to which SDK is required for your specific version of Unreal Engine.
- Sony documentation for PlayStation 4's Neighborhood tool