UNREAL LYRA PROCESSES

EMPTY LYRA WORLD

A — Create Plugin

Create/Refresh Visual Studion solution files. Enable VS plugins.
Edit ▸ Plugins ▸ + Add ▸ Game Feature (with C++) ▸ Name your Plugin ▸ click **Create**.
The Plugin is now available via Edit ▸ Plugins ▸ Game Features. Your plugin Game Feature Data Asset will open. Save it.
For the moment, it will say „Error registering" - that's fine for the moment

B — Preparation

- Make sure to enable "Show Plugin Content" on your Content Browser Settings
- Enter folder Plugins ▸ Your Plugin Name and create root folder **„Maps"**
- Click menu ▸ File ▸ New Level and select a new „Open World Level"
- Click menu ▸ File ▸ Save Current Level As... into **Maps** folder
- Click menu ▸ Edit ▸ Project Settings ▸ Maps&Modes and set your new map as „Editor Startup" and „Game Default Map"

C — Character BluePrint

- Right click in plugin root ▸ Create Folder ▸ "Characters". Enter.
- Right click ▸ Blueprint ▸ base on B_Hero_Default ▸ name BP_XX_Hero, open and click „Event Graph"
- Add a *B_MannequinPawnCosmetics component*
- Copy code from https://blueprintue.com/blueprint/r5mkcql7/
- Insert into graph and compile:

- We use the tag on the Character Parts Actor to decide a feminine or masculine character. This defines the skeleton mesh to be used.
- Click „Class Defaults":
  - **Animation Mode**: Use Animation BP
  - **Anim Class**: ABP_Mannequin_Base
- Click Component ▸ Mesh, set Transform ▸ Location ▸ Z to -88
- Compile, save and close BP

D — LyraPawnData

Right click in plugin root ▸ Miscellaneous ▸ DataAsset ▸ base on **LyraPawnData**, name **DA_XX_PawnData (XX is your project short)**

Open and change

- **Pawn ▸ Pawn Class**: Your new BP_XX_Hero
- **Input ▸ Input Config**: InputData_Hero
- **Camera ▸ Default Camera Mode**: CM_ThirdPerson. Save.

E — Lyra Controller Component - Character Parts

- Right click in Characters folder ▸ Create BluePrint ▸ base on LyraControllerComponent_CharacterParts, name BP_XX_CharParts
- Open, enter Event Graph and add Node „Add Character Part" to Begin Play (or https://blueprintue.com/blueprint/cl1jfncc/ _)
- Right click on „New Part" and choose „Split Struct Pin"
- Set „New Part Part Class" to „B_Manny". Compile and save.

F — LyraExperienceDefinition

Right click in plugin root ▸ Create BluePrint ▸ base on **LyraExperienceDefinition**, name BP_XX_ExpDef. Open.

- Gameplay ▸ Game Features To Enable: Add one element and type in your Plugin Name (same as plugin folder, minus „Content")
- Gameplay ▸ Default Pawn Data: Select above Pawn Data Asset
- Add one „Actions" and select „Add Components"
- Add one items to Component List, with
  - **Actor Class**: Controller
  - **Component Class**: your BP_WA_CharParts
  - **Client Component**: unchecked, **Server component**: checked
- Save

G — GameFeatureData

Open the automatically created GameFeatureData Asset
Add one action, choose „Adds Input Config"
Add one element to „Input Configs", expand and set to

- Config: IPMI_Default_KBM
- Type: Mouse and Keyboard

Add 1 items to Asset Manager ▸ Primary Assets To Scan:
LyraExperienceDefinition

- Primary Asset Type: Type „LyraExperienceDefinition"
- Asset Base Class: Select LyraExperienceDefinition
- Check „Has Blueprint Classes"
- Directories: Add one entry and select your plugin folder

LyraUserFacingExperienceDefinition

- Primary Asset Type: Type LyraUserFacingExperiencesDefinition
- Asset Base Class: Select LyraUserFacingExperiencesDefinition
- Directories: Add one entry and select your plugin folder

Map

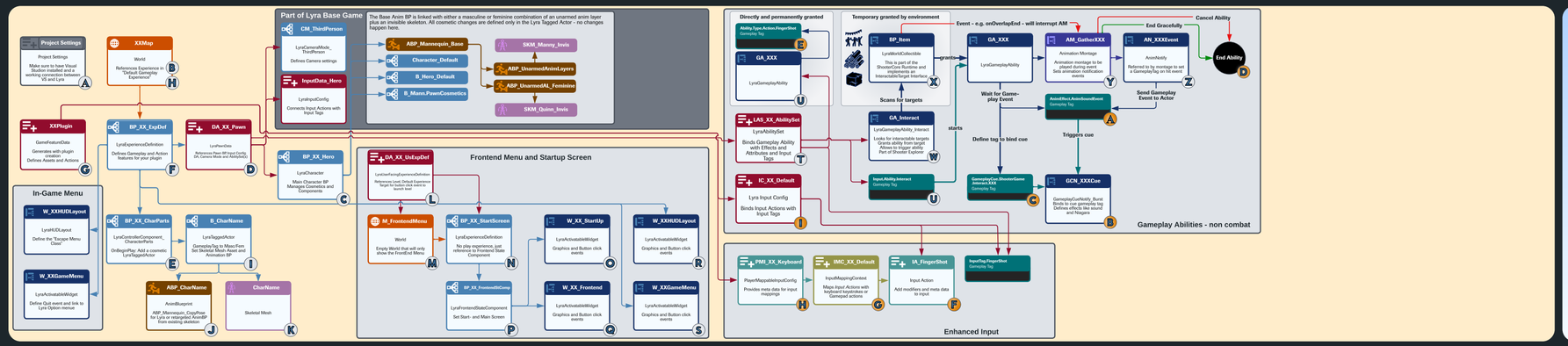
- Primary Asset Type: Type „Map"
- Asset Base Class: Select World
- Directories: Add one entry and select your Maps folder

Set to „Active" and save

H — Set Experience for Map

Open Edit ▸ Project Settings ▸ Game ▸ Asset Manager
Expand „Primary Asset Types to Scan". Expand
„LyraExperienceDefinition"
Expand „Directories". Add your plugin directory there. Restart editor.
Go to World Settings ▸ Default Gameplay Experience ▸ your BP_XX_ExpDef. Play.

REFERENCES



APPEARANCES

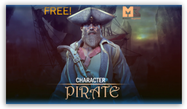
E — Change from Manny to Quinn

- Open BP_XX_CharParts. Change node „Link Anim Class Layers", pin „New Part Part Class" to „B_Quinn"

Use a Marketplace Character: Paragon Feng Mao

- Search the marketplace for „Paragon" - all these characters are free. Get one and add to an existing 4.27 or 5.0 project. Use Windows Explorer to just copy this newly created Paragon folder from that project into your Lyra Content folder. You cannot directly copy this to your plugins content folder as this will mess up the path structure, but you can migrate within the project to it if you want to keep Lyra clean: Just drag it from the Lyra Game folder to your Plgin and choose "Move". Then right click on /Content folder and choose "Fix up Redirectors".
- Open Lyra and search main folder for „ABP_UE4_Mannequin_Retarget". Right click and select „Retarget Animation Assets ▸ "Duplicate and retarget animation BP"
- In the window popping up, select top right your IK Retargeter as „RTG_UE4Manny_UE5Manny". Point „Target Skeletal Mesh" to one of your Paragon Skeletal Meshes. On the lower right, change the folder point to a folder in your plugin. Click „Retarget" - this will create a ABP file in the selected folder. Rename to ABP_PARAGONNAME.
- Now we create a TaggedActor file: Search in Lyra folder for B_Manny if you selected a male, or B_Quinn if you selected a female Paragon. This is just because we are too lazy to set the Static Gameplay Tag manually to *AnimationStyle.Masculine / .Feminine*
- Copy that file into your Paragon folder, rename to B_PARAGON_NAME and open it.
- Click left on the "MeshComponent". On the right in the details panel, set
  - Anim Class: ABP_PARAGON_NAME
  - Skeletal Mesh Asset: One of your Paragons assets from the dropdown
- Compile and save.
- Now open your BP_XX_CharParts and set the tagged actor to your B_PARAGON_NAME. Compile and start!

Another example: Pirate by Maksim Bugrimov



- For this example, I use a free example of Maksim Bugrimov's impressive work - the pirate. Its exactly the same process as above with a Paragon character, but the character is just to great to miss out!
- Add it to your project, then let's keep the Lyra game folder clean, so drag&drop to your plugin folder and fix up redirects via rightclick afterwards.
- Again:
  - „ABP_UE4_Mannequin_Retarget" ▸ Right click ▸ „Retarget Animation Assets ▸ "Duplicate and retarget animation BP" ▸ IK Retargeter as „RTG_UE4Manny_UE5Manny" ▸ Point „Target Skeletal Mesh" to a pirate SKM - e.g. SKM_Pirate_Full01. Retarget and rename to ABP_Pirate
  - Copy B_Manny into your folder, change:
    - "MeshComponent" ▸ Anim Class: ABP_Pirate
  - Skeletal Mesh Asset: One of your Pirate skeleton assets from the dropdown
- Compile and save.
- Now open your BP_XX_CharParts and set the tagged actor to your B_Pirate. Compile and start!

Meta Human

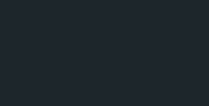
- Open Bridge inside Unreal, select a MetaHuman and import. Download and Import. Enable all missing plugins that are requested during import and restart Unreal.
- Open the your MH folder and open the BP_Name there
- Adjust:
  - Click File ▸ Reparent BP ▸ Lyra Tagged Actor
  - Click the Component root ("self") and in the details panel:
    - Static Gameplay Tags ▸ add Cosmetic.AnimationStyle.Masc or Fem.
    - Static Gameplay Tags ▸ add Cosmetic.BodyStyle.Medium
    - *Use Live Retarget Mode*: Checked
  - Save & Compile BP.
- Open your BP_XX_CharParts and select the modified BP. Run game.

MENUS

L — Frontend Menu and Startup Screen

- First, we create an asset to define our level to launch:
  - In your Plugin root, right click ▸ Misc ▸ Data Asset ▸ Choose type "LyraUserFacingExperienceDefinition", name "DA_UxExpDef"
  - Open that asset and set
    - Map ID: Starter map from "Empty Lyra World", step B
    - Experience ID: Exp Def from "Empty Lyra World", step F
    - Tile title, subtitle and icon as you wish - we actually will not use tiles here in the first step - but Lyra will actually find this asset and will display it already
    - Is Default Exp: Show in Frontend: Checked
    - Loading Screen: Leave empty. Save.
- Both Startup and Frontend screen are of type "LyraActivatableWidget". We use copies of Lyras here:
  - Create a /Plugin/UI folder
  - Copy into it from Lyra game directory
    - W_LyraStartup ▸ Rename W_XX_Startup
    - W_LyraFrontEnd ▸ Rename W_XX_Frontend
- (Optional) Edits in **W_XX_Startup**:
  - copy T_UI_LyraLogo_2K and modify to new mask
  - copy MI_UI_Logo_Glow_DefaultLoadingScreen and point "LogoMaske" to modified mask
  - point W_XX_Startup ▸ Logo ▸ Image to new texture
- Edits in **W_XX_Frontend**:
  - Open and click on "Play Lyra" button. Change "Button Text" to "Play XX", then open the Graph (top right)
  - Add a new variable of type "Lyra User Facing Experience Definition" variable called uExExp:

- Drag into the graph, choose "Set" and link to "Event On Activated"
- Choose the above created Data Asset as "UExExp"



- Add the below blueprint to the On Base Button Clicked event of your StartGameButton (http://blueprintue.com/blueprint/ljmtlbbj/)
  - Note: You have to drag the "UExExp" variable to GET It, then drag from output pin to "Get Map ID", then right click on that output pin and select "Split Struct Pin":

- Normally, the event would just use a "Push Content To Layer" node to switch to the W_ExperienceSelectionScreen where you can define multiple experiences - we replace that here to directly launch the level
- Now we have a startscreen and a frontend - let connect it to the game!
- Enter /Plugin/Maps folder, rightclick and choose "Level", name it "M_FrontendMenu". Open and make sure you did activate the "World Settings" window from the menu to see level details
- Now create a BP of LyraFrontendStateComponent, name "BP_XX_FrontendStComp"
  - Save and open
  - UI ▸ Press Start Screen Class: W_XX_Startup
  - UI ▸ Main Screen Class: W_XX_Frontend
- Enter your Plugin's Experience folder (the one you set in step H of "Empty Lyra World")
- Create a new BP based on LyraExperienceDefinition, name BP_XX_StartScreen. Save it and set is as "Default Gameplay Experience" for the opened map. Open it.
  - Gameplay ▸ Game Features To Enable: Add one element and type in your Plugin Name (same as plugin folder, minus „Content")
  - Actions ▸ Add one element ▸ Add Component
    - Component List ▸ Add one element
      - Actor Class: LyraGameState
      - Component Class: your BP_XX_FrontendStComp
      - Client Comp.: Checked, Server Comp.: Unchecked
- Now play - you should see your startscreen, and an entry menu.
- The options and credits are coming directly from Lyra - maybe copy and modify the credit screen:
  - Copy W_Credits to your Plugin folder, point the Widget Class of yours "On Base Clicked (Credit Button)" event from W_XX_Frontend to this copy
  - Open the credits copy, set the for-loop last index to 1 and set your name into the Append Node

Ingame Menu

- Copy into your UI folder:
  - W_DefaultHUDLayout - rename to W_XXHUDLayout
  - W_LyraGameMenu - rename to W_XXGameMenu
- Open your main levels user experience - created in step F
  - Actions ▸ Add Item ▸ Add Widgets
    - UILayoutsAdd item
      - Layout class: W_XXHUDLayout
      - Layer ID: Select UI.Layer.Menu
- Open W_XXHUDLayout. Set "Escape Menu Class" to W_XXGameMenu
- Open W_XXGameMenu. Reconnect both the "Return Button" and the "Quit Button" to a quit game routine as shown below (https://blueprintue.com/blueprint/xxb44kkv/). Save and run!

ABILITIES & INPUT

An ability pattern promoted here

The Gameplay Ability System decouples an ability - gathering, jumping, shooting, ... - from the actor that is using it.
This decoupling of a complex task into mostly independant services is the mandatory basis for all modern (especially cloud) development, and should also be a mandatory pattern in all game development.

Unfortunately, Epics does not embrace that concept yet: The marketplace has no option to just sell an ability, there is no clear abstraction framework etc. That leads to the current situation, that marketplace assets are overwriting such large parts of your existing processes, that its often easier to reimplement instead of using it.

Lyra is a step in the right direction, so let's hope Epics continues on that path.

The way we use abilities here will embrace the situational context:

- The situation owns and grants the ability, its not permanent part of the character class
- This leads to an abstraction concept that bundles abilities and all of their assets (cues, animation, effects, ...) to make them exchangable and easily integrable.

Directly grant an ability

Let us start with the less recommended way: Directly and permanently granting a character an ability.
There are multiple options to grant abilities via data assets:

- Creating a Lyra Ability Set to connect to the Pawn (LyraPawnData asset)
- Creating a Lyra Experience Action Set to connect to your Gameplay Experience

I will show no. 1 and grant an emote Gameplay Ability to the Pawn:

- Create a Gameplay Tag "Ability.Type.Action.FingerShot"
- Create the Gameplay Ability "GA_FingerShot" (https://blueprintue.com/blueprint/8o6nxfnd/)
- Connect both within the new Lyra Ability Set "LAS_XX_Default"
- Open your Pawn class, add one item to Abilities ▸ Ability Set and add the above Ability Set

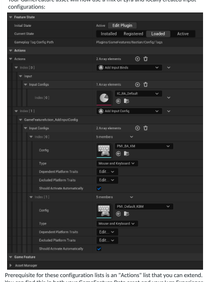
Sparta test blood on gun test Fatality etc

First we trigger the new ability exclusively onSpawn, then we add an extension to the default input mappings exclusively to our new plugin:

- Create an Input Action "IA_FingerShot"
- Create a Input Mapping Context "IMC_XX_Default" to map above IA to a key pressed
- Create a Player Mappable Input Config "PMI_BA_KB", open it and add your new "IMC_XX_Default" to the Contexts list
- Create a new Lyra Input Config "IC_XX_Default", open it and add one Ability Input Action mapping: IA_FingerShot ▸ InputTag.FingerShot
- Reopen your "LAS_XX_Default" and add the missing InputTag.FingerShot

- Open your Plugin GameFeatureData:
  - Add your new "PMI_XX_KB" to Actions ▸ GameFeatureAction_AddInputConfig
  - Add Actions ▸ Add Input Binds, extend the Input Config list and add your new "IC_XX_Default"
- Reopen the ability, make sure the Activation Policy is "Input Triggered"
- Play the game - both keybindings from Lyra - e.g. move and your new fingershot ability should be active now

Your GameFeature asset will now use a mix of Lyra and locally created input configurations:



Prerequisite for these configuration lists is an "Actions" list that you can extend. You can find this in both your GameFeature Data asset and your Lyra Experience Definition, allowing you to add different configurations either on a plugin wide, or map specific level.

The Actions that are available are:

- "Add Abilities"
  - Let you directly add Gameplay Abilities to Actors
- "Add Cheats"
  - Let you add LyraCheatManager derived blueprints
- "Add Components"
  - Let you bind actors to components - we used it to bind the BP for our CharParts to our Controller Actor
- "Add Data Registry" and "Add Data Registry Source"
  - Let you add USTRUCT tagged data structure - details here and add Data Tables and Curve Data to an existing Data Registry
- "Add Gameplay Cue Path"
  - Path to find your Gameplay Cues - these are visual or sound effects used within your abilities
- "Add Input Binds, - Configs and - Mappings": Add local input assets
- "Add Widgets"
  - Add UI components - but you normally would add these on an Lyra Gameplay Experience level
- "Splitscreen Config": Enable Splitscreen (default is disabled)
- "Add World Partition Content" and "Frontend Perf Settings" are quite unclear at the moment - might be work in progress

ABILITIES USAGE

Get an ability from the environment

MULTIPLAYER 01

Spawn your map as multiplayer

To simulate multiplayer in our editor, we can define some parameters to make sure our final game will work. It is recommended to always develop in multiplayer/single user to recognize bugs early:

- Open the options to start your game in the editor - click the 3 vertical dots right of the green start arrow:

Select "Net Mode" / "Play as client". This will spawn an invisible dedicated server and one client window to simulate a remote connection:

If you now start the project with two players, you will most probably be stick on the loading screen - Lyra is waiting for a triggered respawn command to show any characters.
Let's fix that:

- Open your character blueprint - BP_XX_Hero in our case
- Add a BeginPlay event and Trigger an "Request Player Restart Next Frame": https://blueprintue.com/blueprint/7yxzh1wb/

Add map path

- Make sure to add the following paths to your Project OpenSettings / Asset Manager:
  - Primary Asset Type: Maps - your plugin's map directory

INSTALLATION

Youtube video

Install Visual Studio

- Search for Visual Studio Community. Download. Open Install.
- Select as components:
  - .NET Desktop Development
    - Universal Windows Platform Development
    - Desktop Development with C++
  - Game Development with C++
    - Make sure to manually select "Unreal-Engine Installer" on the details panel here
  - Linux and embedded development with C++
- Install and do all updates that might be asked for

- Open Epic Games launcher
- Go to Unreal Engine/Library and install latest Unreal Engine
- Get Lyra from Marketplace and Create Lyra Project from you Library
- Install UnrealVS (Engine Path/Engine/Extras/UnrealVS)

- Open Lyra
- Go to Tools/Create Visual Studio Solution

- Close Lyra, open the newly created solution from Lyras install directory
- Right top you find the "Solution Explorer", right click on "Solution 'LyraStarterGame', and choose
  - Clean Solution - should take some seconds and deletes all binaries
  - Rebuild Solution - takes a couple of minutes while recompiling
- Start Lyra via the green arrow "Local Windows Debugger" (or F5)

Links

- https://developerbastian.github.io/www
- https://www.youtube.com/@bastiandev/videos
- https://github.com/DeveloperBastian
- Unreal Forum: http://bit.ly/3LDHnJ
- https://twitter.com/BastianDevelop

This sign shows that the tutorial is released and will link to the regarding video. All unsigned tutorials are work in progress and most probably will not work. Yet

This is a private tutorial series that is inspired by the official Epics Games Infographics, but completely unaligned with Epic Games. The Unreal logo is used within the Epic Games Branding guidelines