Documentation for Model Processing

*Paul Kim*

*Nicholas Patterson*

*Devin Vandenburg*

Contents

[Workflow 3](#_gjdgxs)

[Setup 5](#_30j0zll)

[Model Processor Tool 12](#_3znysh7)

[Controls 15](#_2et92p0)

[Door Processing 16](#_tyjcwt)

[Door Rotation 17](#_3dy6vkm)

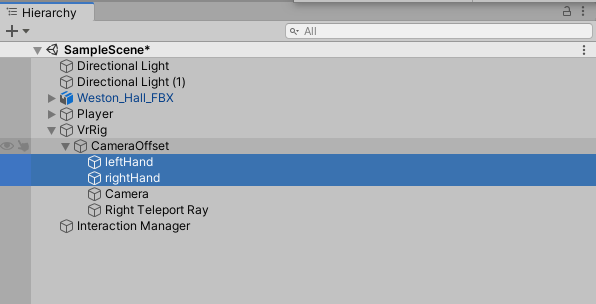
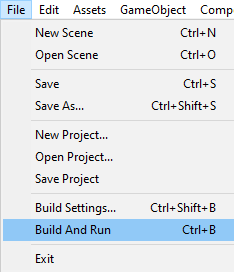
[Movement and Camera 18](#_1t3h5sf)

[Elevators 21](#_4d34og8)

[Windows 22](#_2s8eyo1)

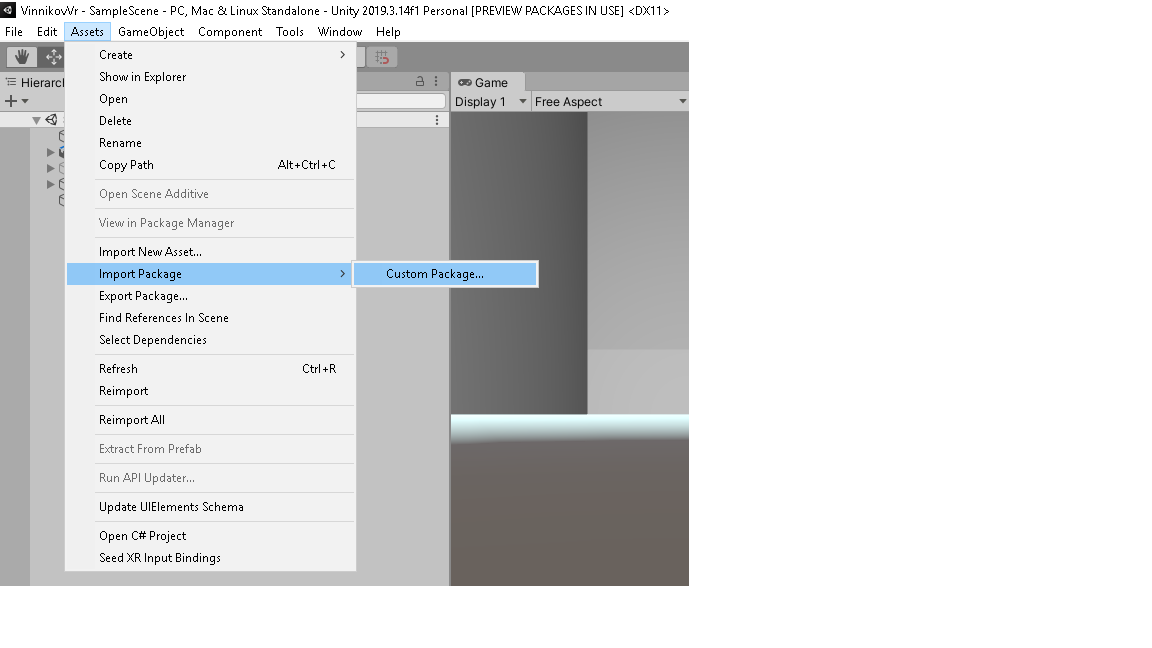
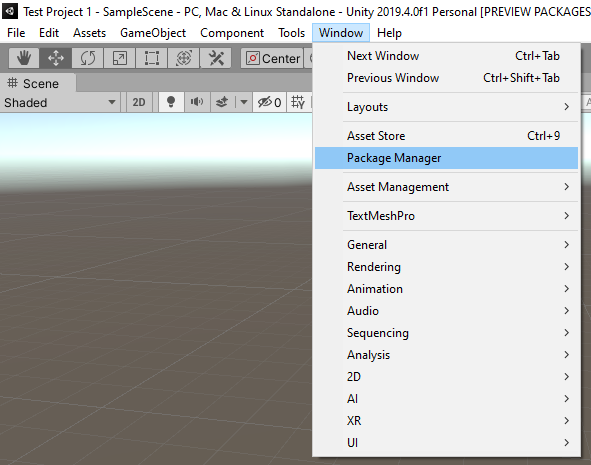
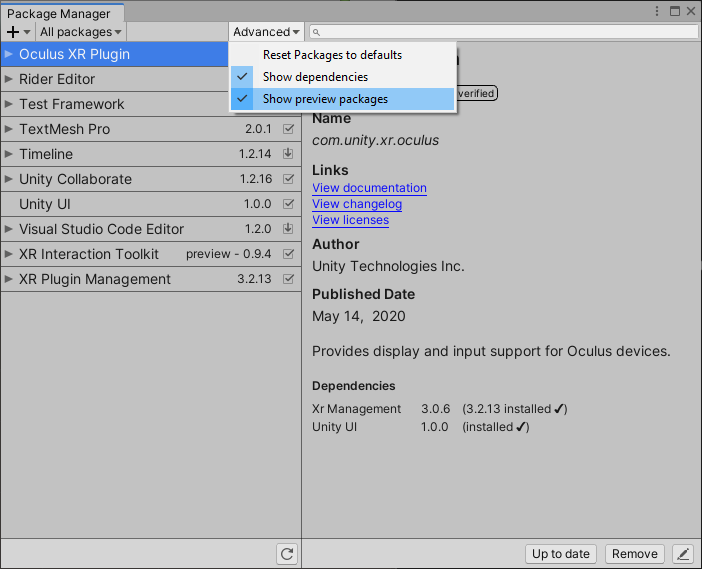
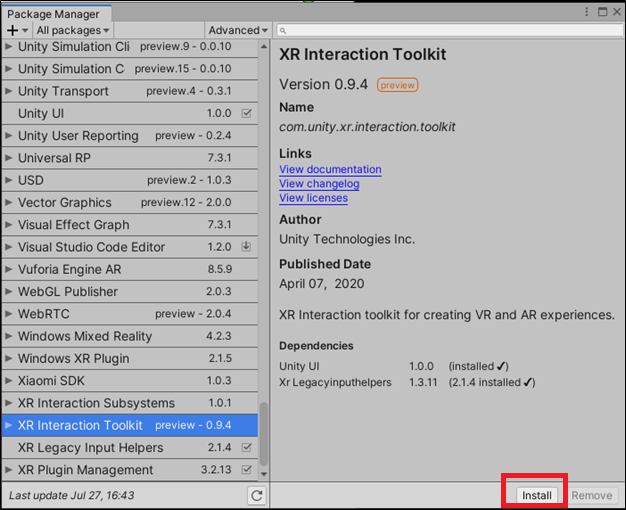
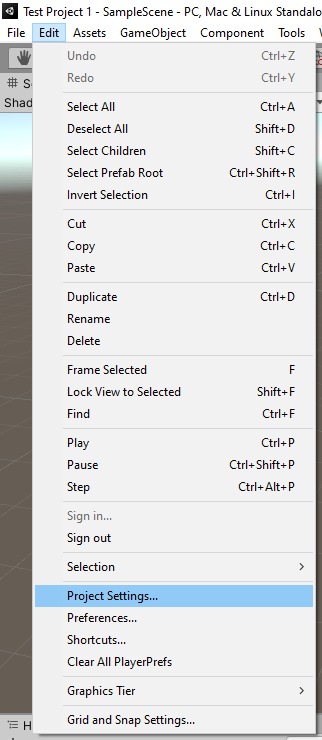
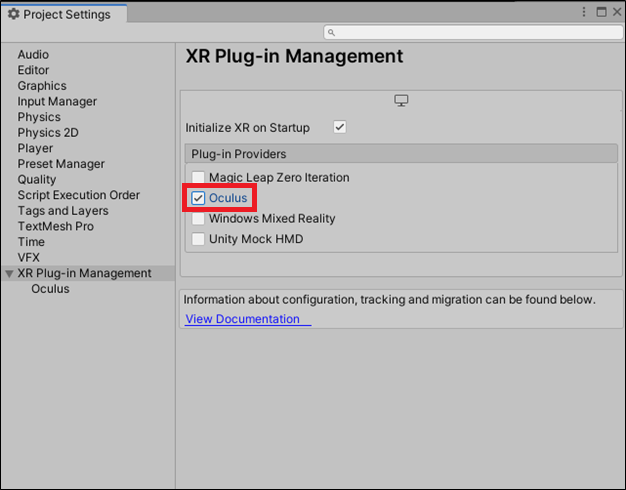
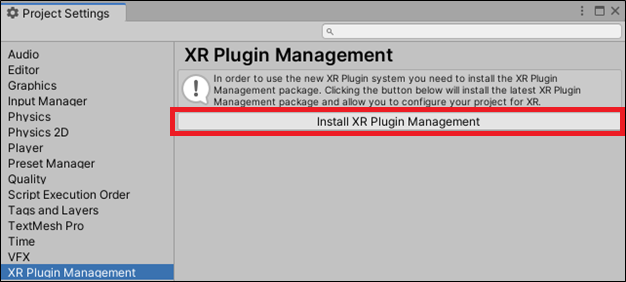
[VR 24](#_17dp8vu)

# Workflow

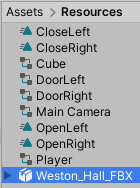
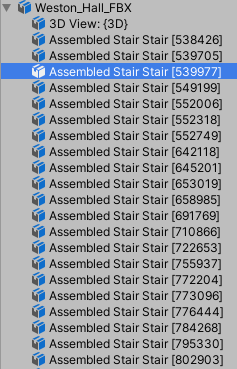
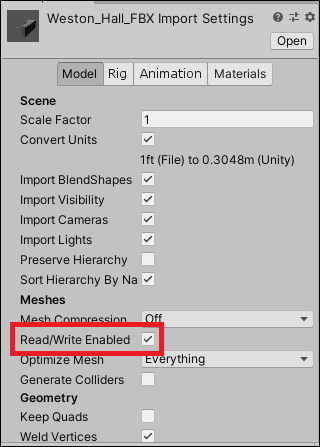
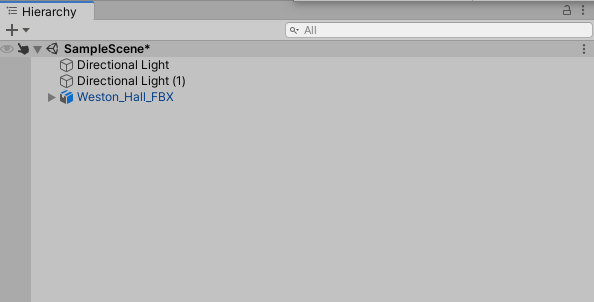
1. Setup environment (Refer to the [Setup](#_gjdgxs) Section for information)
2. Use **Model Processor Tool** (Refer to the [Model Processor Tool](#_3znysh7) Section for information)
3. The following steps can only be done after the **Model Processor Tool** has finished processing.
4. If importing hand models for VR: Locate the rightHand and leftHand objects under **VrRig > CameraOffset**. Here you will add any handle model that you like. Just drag and drop them from assets onto the Controller Prefabs slot. Controller Characteristics should be left for leftHand and right for rightHand.  
   
5. Click Play to test the model
   1. If when hitting play you receive the error "Unable to start oculus XR plugin," "Failed to load display subsystem," and/or "Failed to load input subsystem," that means that either the XR Interaction Toolkit or XR Plug-in Management needs to be updated, the Oculus program itself needs to be updated, or the Oculus isn’t connected. **You may ignore these messages if not using VR.**
6. Click **File** > **Build and Run**   
   ****

# Setup

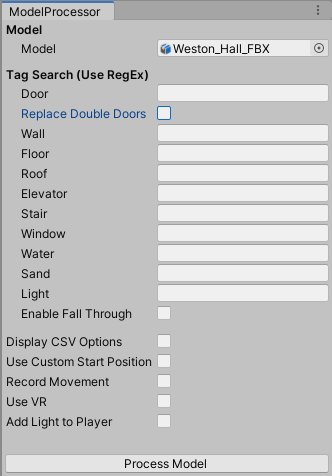
(Make sure to follow all steps or the plugin may not function properly

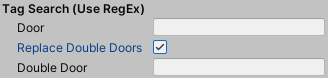
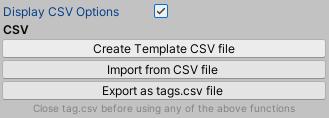
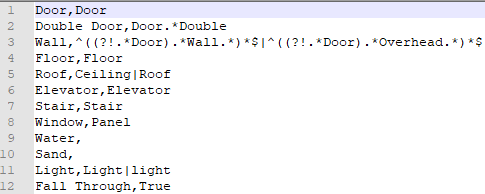
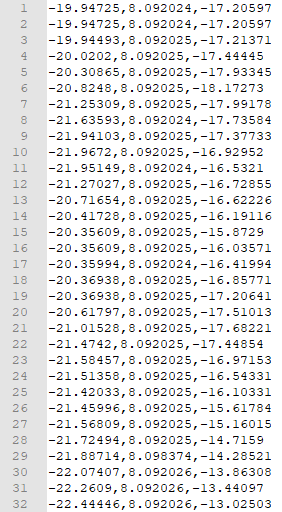
1. Click on **Assets** > **Import Package** > **Custom Package** > Select the package. **
2. Click on **Window** > **Package Manager****
3. Click on the **Advanced** tab and enable **"Show Previewed packages"****
4. Scroll down until you find **XR Interaction Toolkit** and install****
5. Install the **XR Interaction Toolkit**
6. Click on **Edit** > P**roject Settings  
   **
7. Click **XR Plug-in Management** > Install **XR Plug-in Management** > Check **Oculus**  
   ****
8. **(VR Only)** Import models for the user’s hands

Before doing anything, appropriate models for the unity controllers should be downloaded and added to the Assets folder if desired. Standard hands + controllers can be found at <https://developer.oculus.com/downloads/package/oculus-controller-art/>

1. Move the desired Model into the **Assets** folder. Use a model that has objects labeled. A demo model is provided in the **Resources** folder.  
      
   
2. Click on the Model and make sure “Read/Write Enabled” is checked. Apply changes.
3. Drag the Model into the Scene. 

# Model Processor Tool

This tool can be accessed in the Unity Editor by going to **Tools** > **Model Processor**.   
  


1. **Model**
   1. **Model** – After importing the Model into the project, drag and drop the model into this field.
2. **Tag** **Search** – All text fields will take Regex expressions and tag objects in the model based on their names. The objects are checked against Regex strings in the order they appear in the tool.
   1. **Replace Double Doors** – When checked, this option will enable the **Double Doors** text field.  
      
   2. **Enable Fall Through** – When unchecked, if matched will multiple Regex strings, objects will be tagged based on the first Regex string they match. When checked, objects will be tagged based on the last Regex string they match.
3. **Display CSV Options** – When checked, displays all CSV buttons  
   
   1. **Create Template CSV file** – Creates a CSV file that can be filled in.
   2. **Import from CSV file** – Allows the user to import a CSV file to fill out the Tag Search fields. CSV should be formatted with “tag, regexString” on each line. A demo file is provided in the Documentation folder.  
      
   3. **Export as tags.csv file** – Exports the current strings in the Tag Search fields as a CSV file.
4. **Use Custom Start Position** – When checked, allows the user to use their own predefined coordinates as starting spawn positions. If checked but no position.csv has been imported, the default spawning locations will be used.  
   
   1. **Import from CSV file** – Coordinates for each spawn location should be on their own line, with commas separating the x, y, and z coordinates.  
      
5. **Record Movement** – If checked will record player position every x frames to a CSV file. File can be found in **Assets** > **Movement**.  
     
   
   1. **Record every X frames** – Dictates how often a players position will be recorded
6. **Use VR** – Disables the Player object and enables the VrRig object.
   1. If this box is checked, the player object will be disbabled and the VRRig object will be enabled. This will remove the option of using keyboard and mouse
   2. If this box is not checked, the player object will be enabled and the VrRig object will be disabled. This will remove the option of using VR.
   3. If you want to swap controls without running the model processor again, you can enable and disable objects in the unity editor.
7. **Add Light to Player** – Adds a Point Light to the Player and VrRig objects.
8. **Process Model** – Process the Model based off the inputs above

# Controls

Keyboard and Mouse

Movement – WASD Keys

Camera – Mouse

Elevator Up – U key (Press again to stop)

Elevator Down – D key (Press again to stop)

VR

Movement – Left Controller Joystick

Camera – Headset

Teleportation – Right Controller Grip button

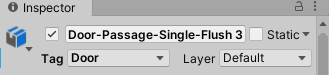
Elevator Up – Right Controller A button (Press again to stop)

Elevator Down – Right Controller B button (Press again to stop)

Keyboard and Mouse + VR

When playing, the mouse cursor will be disabled by default. To have it show up again, press the Esc key.

# Door Processing

1. Ensure the **ObjectComponentProvider** script is inside the **Navigation Scripts** folder in the project.
2. Run the **Model Processor** tool (See Model Processor Tool section)
3. Once the tool has completed, all doors should be processed with their tags. All double doors should be similarly tagged.  
   
4. Run the project with the play button at the top of Unity, this button can be found next to the pause button. These buttons share a similar resemblance to that of a video’s pause and play button.  
   
5. If a door is a double door, the script would have split these doors into two separate doors automatically, these should end with “\_1” and “\_2”. Regular doors will have “\_1” added to the end.
6. Once the project has built and is running, all the doors will have their designated components attached on them.

# Door Rotation

1. At the beginning of Unity, there will be a **Resources** folder filled with animations and animator controllers, ensure this folder is in the Unity project.
2. At this step all doors should be tagged and be given an **Animator** component, this component will have the **DoorRight** or **DoorLeft** animator controllers attached onto it depending on if the name ends with \_1 or \_2.
3. The **Model Processor** automatically adds the **Open** and **Close** animations on the animator controllers, so the creator does not have to worry about it.
4. While the game is running, simply walk up to a door and it will open automatically either towards or away from the player.
5. This is completed because the **ObjectComponentProvider** script spawns in a **DoorHolder** object at the hinge of a door to allow all doors to open and close off the edge of the wall smoothly.

# Movement and Camera

The following sections will explain how Keyboard/VR controls are manifested if you were seeing it in Unity (In case the user wants to change any parameters before building).

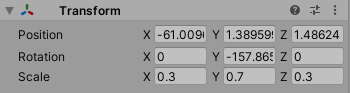
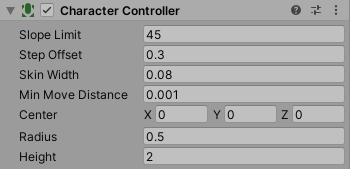
**Keyboard and Mouse controls - W key goes forward, S key goes backwards, A key goes to the left, and D key goes to the right.**

**- Camera can be rotated using the mouse**

Everything you need to know about Keyboard/Mouse Movement

The KeyBoard and Mouse controls will be attached to an object called "Player" when the model is processed through the tool.

The **player** object contains 8 components:

1. **Transform** - The characters scale should automatically be set to 0.3, 0.7, and 0.3 to allow for movement through all doors. However, if it is not set to that, manually do so before building. These values can be modified depending on the model used.  
   
2. **Capsule (Mesh Filter) and Mesh Renderer** - These allow the player model to been; nothing further needs to be modified.
3. **Character Controller** - This enables gravity as well as colliders for the player; nothing further needs to be modified. If there any steps or obstacles in the model that the character cannot get over that they are supposed to, modify the step offset parameter to fit the needs of the model. 
4. **Player Movement** - The enables to the character to able to move using the W key as forward movement, the S key as backwards movement, the A Key as left movement, and the D key as right movement.



**Player Movement** contains 4 relevant parameters:

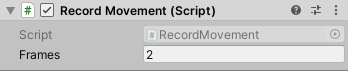
**Forward Force** - The rate at which the player moves. It is set to 3 as a default, but can be changed before building.

**Controller** - This is where the character controller component will go, but it is assigned by a script.

**Gravity** - Works with Character Controllers gravity. Is defaulted to 10 but can be changed before building.

**Kb Player** - This will activate when the "Use VR" option is not selected in the model processor, this lets unity know that only keyboard movement should be used. The parameter will automatically be assigned.

**VR player, Input Axis, and Input Source** are only modified for the VRPlayer and will be explained more in the VR section of documentation.

1. **Record Movement** - This will record the players movement every 'X' frames. It contains the frame parameter which can override the default option of 2 frames.  
   

The **player** object also contains 2 children objects:

**Main Camera** - This is the camera that will allow the user to see what’s around them

**Pivot** - This helps with turning the camera properly

**Pivot** contains 1 component:

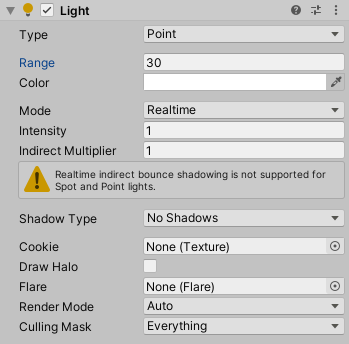
**Transform** - Make sure the position and rotation are set to 0,0,0. Nothing further has to be modified

The **Main Camera** has 3 components:

**Transform** - Make sure the position and rotation is set to 0,0,0. Nothing further has to be modified

**Camera** - Enables to functionality of a camera; nothing further has to be modified

**Camera Control** - This script will allow for the camera to change angles depending on movement of the mouse. It will also turn the character in the direction that the camera is turning in. Camera Control control contains a 'Mouse Sensitivity' parameter that can changed to fit the user's comfort.

1. **Light** - This will only be active if the "add light to player" option in the model processor is selected. This will attach a light to the player, illuminating everything around them. Light contains two parameters called Range and Intensity that can be modified for the personal preference depending on the model used.  
   

# Elevators

1. At the beginning of Unity, ensure all **Elevator** scripts are in **Navigation** folder, this includes **ElevatorSpawn, ElevatorMotion, and ElevatorCommand**.
2. The **ElevatorCommand** script will be placed on the player while the **ElevatorSpawn** is placed on the Elevator Door object. This will spawn in an elevator in the elevator shaft that has the **ElevatorMotion** script on it.
3. When the player walks on the elevator, the elevator connects to the player object through the Collider to link with the **ElevatorCommand** script.
4. At this point, simply press the U button to move the elevator up, or press the J button to move the elevator down.
5. There are two ways of proceeding with this on the scripts, the first is simply using the MeshCollider to make the elevator move the player upwards by pushing.
6. The second way is commented out, but it makes the Player a child of the elevator, so any motion the elevator does makes the player smoothly follow the elevator. However, this causes a lot of bugs. The two bugs that occurs with this method include either sending the player into space away from the model entirely, or the player could end up at 0, 0, 0 once it leaves the Elevator.

# Windows

1. While Windows are not yet implemented, I have left the components to easily allow Windows to work.
2. Windows require an animation to open them which could either be completed by using the DoorRight or DoorLeft animations to open the windows or close them.
3. Using the same animation controller idea the Doors had to open and close them will allow for windows to open or close.
4. You could use a script that each time you enter the area of a window it opens or you could make it if you’re in the area of a window and press a button then it will open, this can be accomplished by copying a similar idea to the **ElevatorCommand** and **ElevatorMotion** scripts.

Lights

1. At the beginning of Unity, there will be **Light** scripts in the **Navigation** folder that will react to the **ObjectComponentProvider** to place them on **Light** objects.
2. Simply run the project and this should be done.
3. Lights should spawn in a game object to act as a trigger for when the player for when the player walks under a light, the light will activate and when they walk away, it will deactivate.
4. The trigger allows this to be completed.

# VR

Everything you need to know about vr controls

The VR controls will be attached to an object called **"VrRig"** when the model is processed through the tool.

The "VrRig" object contains 7 components:

1. **Transform** - The characters scale should automatically be set to 0.3, 0.7, and 0.3 to allow for movement through all doors. However, if it is not set to that, manually do so before building. These values can be modified depending on the model used.
2. **XRRig** - This represents the VR player. All values will be added automatically by the object component provider script. If you feel like your character is too tall or too short, you can adjust the camera Y offset for personal preference.  
   
3. **Character controller** - This enables gravity as well as colliders for the player. Nothing further needs to be modified.
4. **Locomotion System** - This is one of the components necessary for the player to move. Nothing further needs to be modified.
5. **Teleportation Provider** - This will allow the player to teleport. Nothing further needs to be modified.
6. **Player Movement** - This allows the player to move in VR using the joystick on the left controller.  
   

Player Movement contains 6 relevant parameters:

**Forward Force** - The rate at which the player moves. It is set to 3 as a default, but can be changed before building.

**Controller** - This is where the character controller component will go, but it is assigned by a script.

**Gravity** - Works with Character Controllers gravity. Is defaulted to 10 but can be changed before building.

**Input Axis** - This catches the coordinates from the left controller's joystick to use for movement. Nothing further needs to be modifed.

**VR Player** - This parameter will automatically get assigned if the "Use VR" option is selected in the model processor.

**Input Source** - This parameter will be what is used to capture input used towards movement. This will automatically be set to left hand when the program starts.

\*KbPlayer is not relevant as keyboard controls will be turned off at this point.

1. **Light** - This will only be active if the "add light to player" option in the model processor is selected. This will attach a light to player, illuminating everything around them. Light contains two parameters called Range and Intensity that can be modified for the personal preference depending on the model used.

The **"VrRig"** object contains 1 child:

**CameraOffset** - This helps with the camera positioning. Nothing further needs to be modified.

The **"CameraOffset"** object contains 4 children:

**leftHand/rightHand** - These objects represent the hands of the player. They both contain 2 components:

**Transform** - Nothing here needs to be modified

**XR controller** - This links the objects to the unity controllers the ControllerNode parameter will automatically be set to LeftHand for the leftHand object and RightHand for the rightHand object. Their select usage will automatically be set to the grip.

**Camera** - This handles the camera in unity. All added components are modified automatically and will work for any model

**Right Teleport Ray** - This produces a beam from your right hand that will be used to teleport the player to the beams location. All components and parameters are automatically added and will work for any model.

The **"Interaction Manager"** object will also be automatically added to the scene. This is crucial for making sure that the VRRig works. Nothing there needs to be modified.