

v2.0 Installation Instructions

This guide will walk you through adding the required components to a scene to enable your Unity project for Igloo compatibility.

Refer to the Reference for a more in-depth look at the components that make up the Igloo camera toolkit, and how it can be customised for certain use cases.

If you are new to Unity, we recommend exploring some of [Unity's interactive tutorials](#) to familiarise yourself with the Unity interface and concepts before continuing. More information on Unity can be found at their [Learn Site](#).

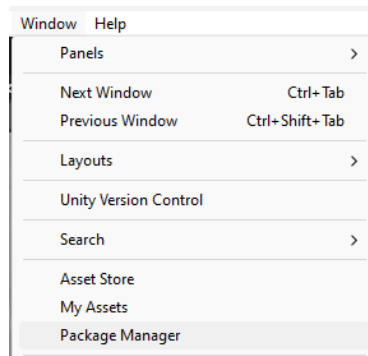
- [Adding the Igloo Manager](#)
- [Customising the Igloo Camera System](#)
- [Adding the Igloo Player](#)
- [Advance: Adjustment of JSON files](#)

Project Setup

Package Import

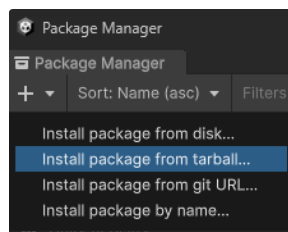
To import the v2 version of the Igloo Toolkit, please use Unity's Package Manager.

You can find it under Window → Package Manager



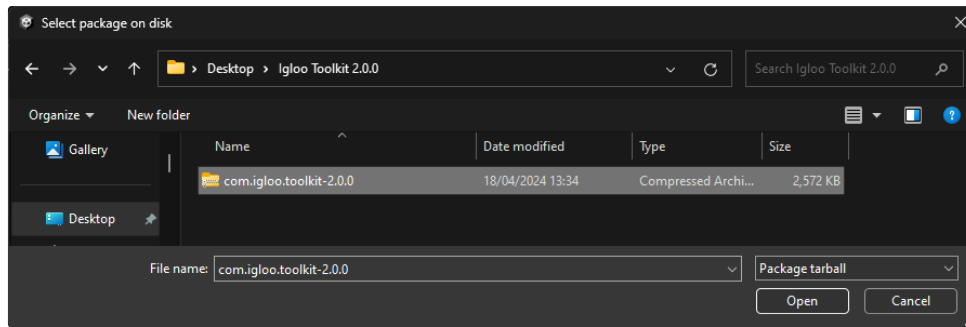
Location of Package Manager in Unity Editor

Open the Package Manager, and click the Plus button in the top left hand corner of the window,, then select *Install Package from tarball...*



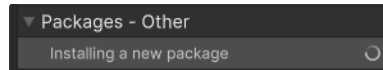
Install from tarball location

A browser window will open. Navigate to your Igloo Toolkit downloaded files which have been obtained from our Downloads page. Select the **com.igloo.toolkit-2.x.x.tgz** file, and press the **Open** button



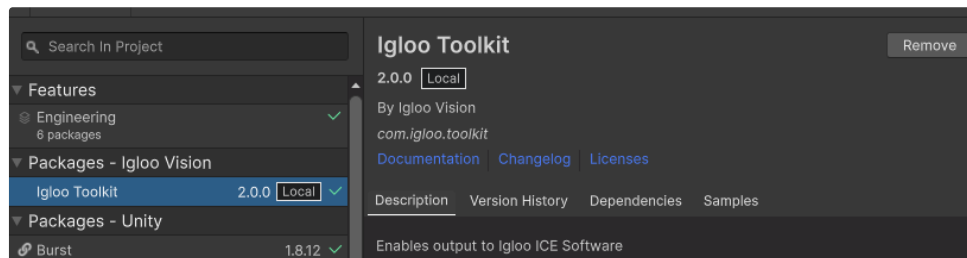
Selecting com.igloo.toolkit-2.0.0.tgz from the desktop

The package will then begin to import and unity will start to compile.



Tarball package importing

Once installed, the package manager will look like this.



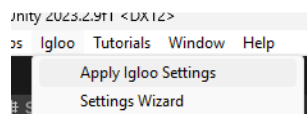
Input Settings

The Igloo camera package requires some custom Input settings to be compatible with an Xbox controller. This can be done one of two ways;

Note - The next step is important, you will encounter errors in play mode if you do not apply the following settings.

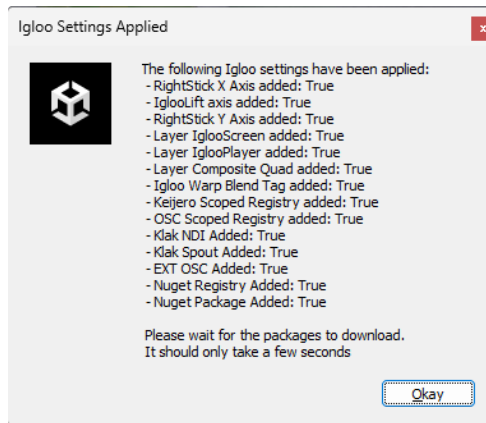
The IglooPlayer system requires two custom Input Axes, a custom tag, and 2 custom layers to be defined in the Unity project settings.


These can be appended to your existing settings automatically by selecting *Igloo > Apply Igloo Settings* from the Unity Editor.



This will also add three packages to package manager, along with some additional player settings, and input settings.

You will see this popup after it has finished loading, just click 'Okay'



 It may take Unity a second or two to figure out what has just happened, so it's best to click onto something else then click back on the Unity Editor (Or press the Windows Key, then select the editor again) which forces Unity to re-compile.

You'll then get a small notification regarding the additional packages that have been added to the package manager. This popup can be closed without issue.

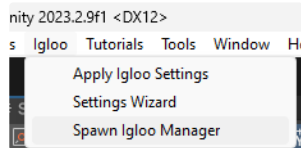
You are now ready to add the Igloo Manager to the scene.

Adding the Igloo Manager

Now the Igloo Toolkit is installed, and the required additional components, adjustments, and settings are applied, it's now possible to add the Igloo Manager to your scene and output to an Igloo Core Engine powered Display System.

Adding the GameObject to your scene.

With your scene open, click Igloo → Spawn Igloo Manager from the top bar



This will add the Igloo Manager GameObject to your scene, and populate it with the NDI and Spout resources that it requires.

It's always a good idea to check that these have been added, you should see them on the Igloo Manager Component as per the image below.



If they are not present, please follow the instructions below.

- ✓ My Igloo Manager doesn't have Spout or NDI resources

Try clicking off the Unity Editor, or pressing the windows key, then clicking back on the editor window again.

The relevant packages might not have installed yet, and are waiting for the next compile cycle. Once complete, two buttons should then appear on the Igloo Manager component to add the resources to it.



Add Resources buttons are visible

If the **Add Resources** section is not visible, please make sure you have clicked Igloo → Apply Igloo Settings from the top bar, and waited for the packages to install. If it still doesn't work, please contact Igloo Support.

You can now press Play and the Igloo Manager will now create the complete camera system, and output a standard equirectanglar of 8000 x 4000 to Igloo's ICE software using the Spout protocol.



Customising the Igloo Camera System

By using the default settings the Igloo Manager will create a standard 2:1 equirectangular output that is sent via Spout to ICE. It will also follow the Main Camera in the scene on the update thread, acting like a 360 degree 'drone' camera rig.

However this may not be what you require and wish to customise how the camera system is both created and interacted with.

Instruct the Camera to Follow a different object

By default the camera system will look for a camera in the scene with the tag 'Main Camera' and it will then follow it. To change this behavior, drag and drop the GameObject you wish the camera system to follow into the Follow Object field within the Igloo Manager component.

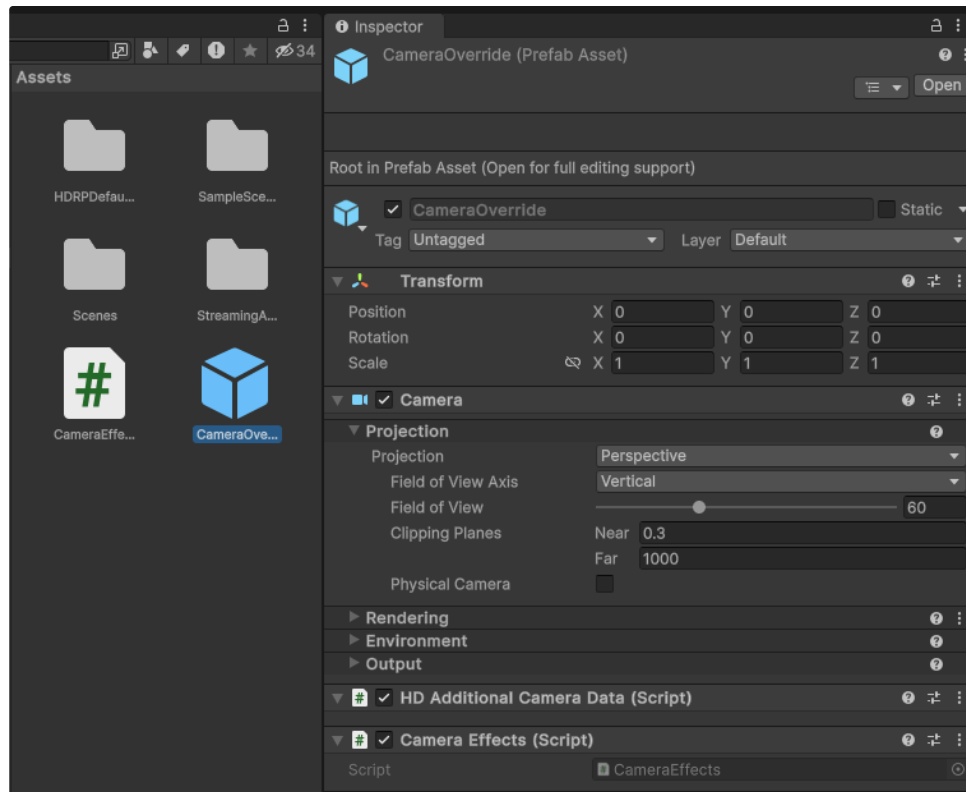


PlayerFollowCamera object added to Follow Object of Igloo Manager

Customising the cameras that are created using a prefab.

When creating complex post processing systems it is often required that additional settings are changed on the cameras that view the game world. As the Igloo Camera System generates its own cameras from scratch, we've made it possible to give the Igloo Manager the ability to use your own Camera, with any additional components or children, as a basis for it to make the entire camera rig.

To achieve this, you need to make a prefab somewhere in your project's assets folder of a GameObject that at the very least contains a camera component. In the image below I've created a prefab called CameraOverride which has a camera component and a script called "CameraEffects.cs", and a child GameObject called "node".

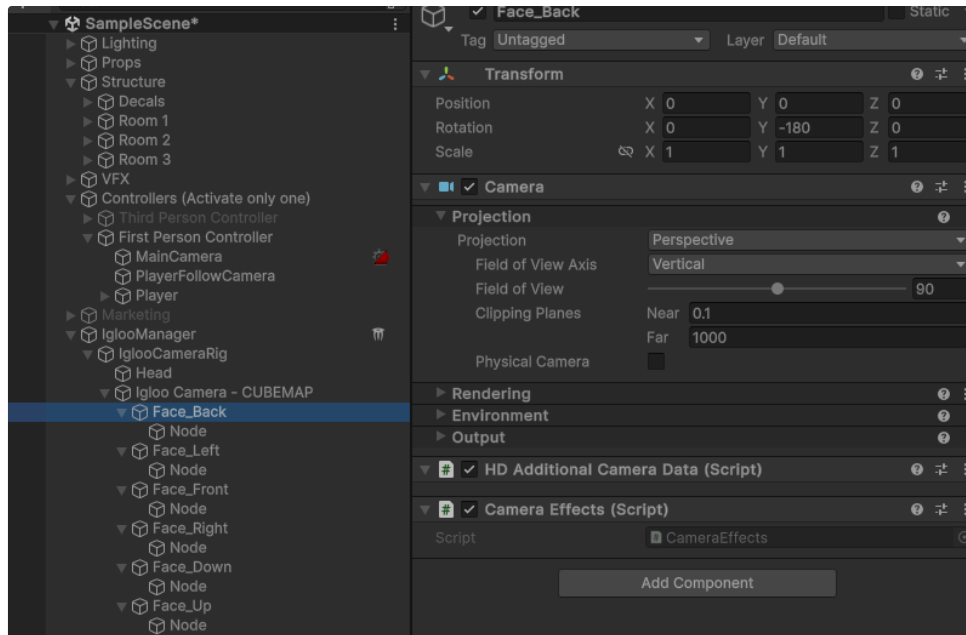


Inspector view of the CameraOverride prefab object

Placing this prefab into the Custom Camera Prefab slot on the Igloo Manager component and then pressing play results in 6 of these cameras being created, along with their subsequent additional components and children.



Camera Override prefab placed into Custom Camera Prefab slot



During Play mode. Additional Node children created and CameraEffects.cs script on Cameras.

Changing the Camera rig output settings

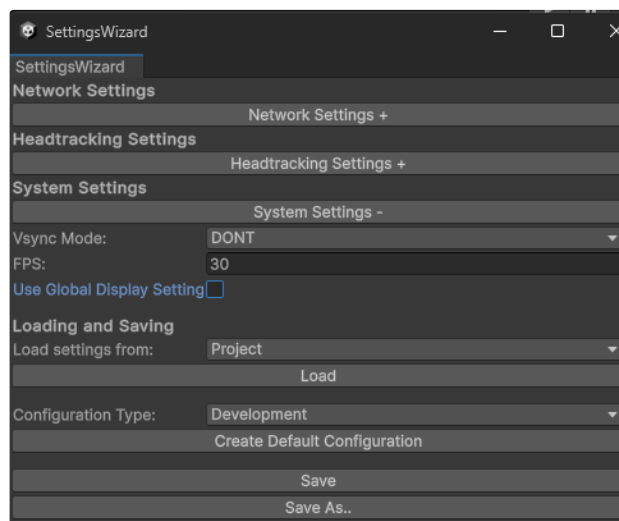
In this version of the camera system, the camera settings are adjusted by a global settings file that is created by the Igloo Setup tool at the point of the ICE installation being created. This is to ensure that Unity is outputting at it's maximum potential on the system that it is operating on. This may mean that it creates it's own windows using the warping files generated by ICE, or it may create the same equirectangular output as in development mode.

The only time these settings would need to be edited would be to use the camera rig in Igloo Spectator mode, whereby it would use the NDI output system rather than Spout.

Changing the output to NDI

Firstly create a new IglooSettings.xml file by launching the Settings Wizard by either clicking the Open Settings Wizard button on the Igloo Manager component, or clicking Igloo → Settings Wizard from the top bar.

1. Click Create Default configuration
2. Expand the System Settings sub menu
3. Uncheck *Use Global Display Settings*
4. click **Save**



Press **Play** and then **Stop** once everything has loaded. This will make Unity create a `IglooCameraSettings.json` file for you in `StreamingAssets`.

You can then edit this `IglooCameraSettings.json` file using a text editor like VSCode to change it from Spout to NDI by changing **`isSpoutOutput`** to **`false`**, and **`isNDIOutput`** to **`true`**.

You may also wish to experiment with the `cubemapFaceResolution`, as lowering this will increase frame rate dramatically, but at the cost of quality.

A screenshot of a code editor showing the `IglooCameraSettings.json` file. The file is located at `D:\> 1_Unity_Projects > HDRP_2023_Sample > Assets > StreamingAssets > {`. The JSON content is as follows:

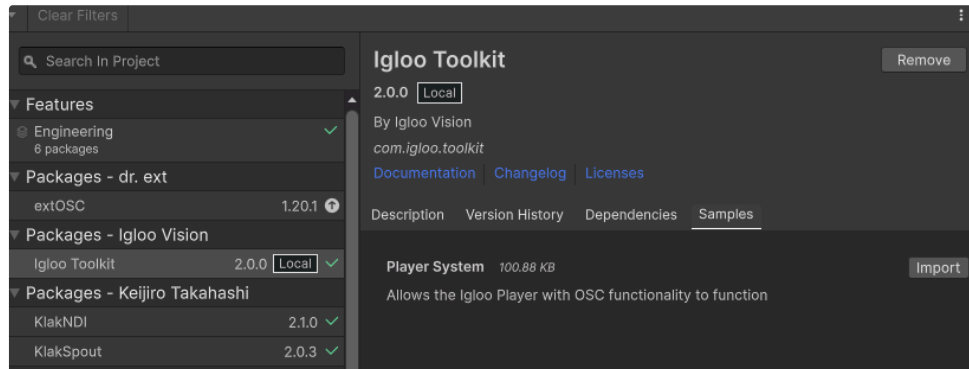
```
1 {
2   "isEnabled": true,
3   "isStereo": false,
4   "isSpoutOutput": true,
5   "isNDIOutput": false,
6   "useWarpAndBlend": false,
7   "nearClipPlane": 0.10000000149011612,
8   "farClipPlane": 1000.0,
9   "stereoEyeDistance": 0.05000000074505806,
10  "cameraRotationOffset": {
11    "x": 0.0,
12    "y": 0.0,
13    "z": 0.0
14  },
15  "cameraType": 0,
16  "CubemapData": {
17    "useTruePerspective": false,
18    "cubemapFaceResolution": 2000,
19    "faceList": []
20  },
21  "OffAxisData": {
22    "canvasWidth": 0,
23    "canvasHeight": 0,
24    "cameras": []
25  }
26 }
```

Adding the Igloo Player

Unlike previous generations of the Igloo Toolkit, the player system is not included automatically and instead must be added if it is required. This is achieved by using the Samples system of the Package Manager which adds the various scripts and features required for the player system to function, to your Project folder.

Importing the Igloo Player

Open your Package Manager and navigate to the Igloo Toolkit in the 'In Projects' section, then click on the Samples page.

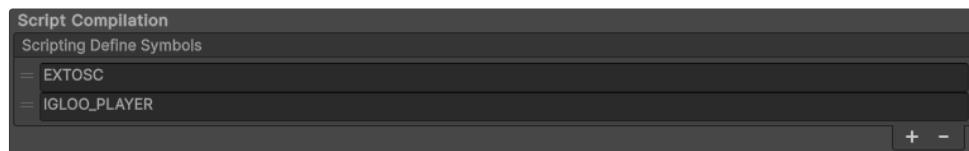


Samples page of the Igloo Toolkit Package

Then click Import on the Player System, and it will add the Samples → Igloo Toolkit → Player System directory to your Project Assets.

It will also add a IGLOO_PLAYER scripting Define Symbol into the Player section of the Project Settings.

⚠ If you happen to not require the Player System and wish to remove it, you also need to manually remove the IGLOO_PLAYER scripting define symbol from Project Settings to avoid compile errors.



Scripting Define Symbols showing IGLOO_PLAYER added

You can now press play and you'll notice that the Igloo player system is created.

New Features of the Igloo Player

With this evolution of the Igloo Player, we've streamlined the process of using the player further with the following features

- You now no longer need to choose what input method you use. It will begin listening for OSC and Mouse events and respond to either.
- The only exception to the above, is if Headtracking is being used, in which case you specify in the settings that it is active, and then it will listen to either VRPN or Optitrack messages based on the setting in the xml.
- The player is now fully generated from code, so to modify it you can add additional features to the code creation process. We found many clients preferred this method over modifying the prefab.
- A lot of old code was removed from the player system, further streamlining its creation and processing cost.

Advance: Adjustment of JSON files

There are two json files that control how the Igloo Camera Toolkit creates and operates the visible output:

- The Camera Settings File
- The Display Settings File

Both of these are stored globally in the Igloo ICE settings files, however if the IglooSettings.xml has been modified to use 'local' settings, then these files can be found in StreamingAssets alongside the venerable IglooSettings.xml file.

Reasons to modify these files

- You are an Igloo installer and you're setting up ICE for the first time.
- An adjustment has been made to the installation, and the files need to be adjusted to suit.
- You are making an NDI or 3D version of an application that differs from the Global version stored on the ICE machine.

Locating the Global Files

By default the global files are located at the path below.

```
1 C:/ProgramData/Igloo Vision/IglooCoreEngine/settings/Unity/
```

You can also modify this in the IglooSettings.xml, by adjusting line 3 as below:

```
1 displaySettingsOverridePath="C:/ProgramData/Igloo Vision/IglooCoreEngine/settings/Unity/displaySettingsOver
```

Breaking Down the Camera Settings file

Below is a table of each adjustable setting in the IglooCameraSettings.json file. This is the default file that would exist on most Igloo ICE installations.

Option	Input Type (<i>Default value</i>)	Effect
isEnabled	boolean (<i>true</i>)	Enables the Igloo Camera system.
isStereo	boolean (<i>false</i>)	Enables 3D output of the Igloo Camera system.
isSpoutOutput	boolean (<i>true</i>)	Creates a Spout sender, which sends the full camera output texture to another application on the same machine.
isNDIOutput	boolean (<i>false</i>)	Creates an NDI sender, which sends the full camera output texture to another application on any machine on the same LAN.
useWarpAndBlend	boolean (<i>false</i>)	Utilises the Igloo ICE warping and blending files in order to bypass the ICE software. Allows unity to create its own windows and display the output without ICE. Useful for large projects or complex systems to gain maximum performance.

outputName	string (<i>"IglooUnity"</i>)	When using Spout or NDI, this value will be the sharing name of the texture stream.
nearClipPlane	float (<i>0.1</i>)	Adjusts the near clip for every generated igloo camera in the camera rig.
farClipPlane	float (<i>1000.0</i>)	Adjusts the far clip for every generated igloo camera in the camera rig.
stereoEyeDistance	float (<i>0.5</i>)	Adjusts the inter pupillary distance between each camera when outputting 3D stereo.
cameraRotationOffset	vector3 (<i>x:0.0 , y:0.0 , z:0.0</i>)	Rotates the camera rig within the 3D space. Useful for when Unity 'faces forwards' on the wrong wall in an Igloo Space.
cameraType	integer (<i>0</i>)	Enum: 0 = Cubemap System 1 = OffAxis System.
CubemapData	Struct of Cubemap Data	<i>explained below.</i>
>useTruePerspective	boolean (<i>false</i>)	If true, an additional shader pass using the Igloo TruePerspective data will be carried out before outputting the texture. This can be a performance advantage. in some cases, over ICE doing it.
>cubemapFaceResolution	integer (<i>2000</i>)	When generating a cubemap, each face of the cubemap will be rendered as a square with the resolution of this value on x and y.
>faceList	Array of Faces to generate	The faces in this array do not have to be in order. As they will be generated in the following order regardless: Left, Front, Right, Back, Bottom, Top. Each face in the array can be disabled if required. A black texture will be sent, but it will not render the scene.
>> face	string (<i>"FaceName"</i>) boolean (<i>true</i>)	e.g: "face":"Back", "enabled": true.
OffAxisData	Struct of Off Axis Data	<i>explained below.</i>
>canvasWidth	float (<i>0</i>)	The total width of the output canvas.
>canvasHeight	float (<i>0</i>)	The total height of the output canvas.
>cameras	array of OffAxisCameras	Informs the offAxisCamera system on how to create each camera in the scene. <i>Explained Below</i>
>>id	string (<i>"cameraName"</i>)	ID for the camera. Name similar to cubemap faces for consistency.
>>roi	rect (<i>x:0.0, y:0.0, z:0.0 w:1.0</i>)	The region of the canvas that this camera is going to output to.
>>lowerLeft	vector3 (<i>x:-1.0 , y:-1.0 , z:-1.0</i>)	Lower Left corner of the camera frustum

		in 3D space relative to the player origin.
>>lowerRight	vector3 (x:-1.0 , y:-1.0 , z:-1.0)	Lower Right corner of the camera frustrum in 3D space relative to the player origin.
>>upperLeft	vector3 (x:-1.0 , y:-1.0 , z:1.0)	Upper Left corner of the camera view frustrum in 3D space relative to the player origin.

Below is an example IglooCameraSettings.json file

▼ IglooCameraSettings.json - example code

```

1  {
2      "isEnabled": true,
3      "isStereo": false,
4      "isSpoutOutput": true,
5      "isNDIOutput": false,
6      "useWarpAndBlend": false,
7      "outputName": "IglooUnity",
8      "nearClipPlane": 0.1,
9      "farClipPlane": 1000.0,
10     "stereoEyeDistance": 0.050000000074505806,
11     "cameraRotationOffset": {
12         "x": 0.0,
13         "y": 0.0,
14         "z": 0.0
15     },
16     "cameraType": 0,
17     "CubemapData": {
18         "useTruePerspective": false,
19         "cubemapFaceResolution": 2000,
20         "faceList": [
21             {"face": "Back", "enabled": true},
22             {"face": "Left", "enabled": true},
23             {"face": "Front", "enabled": true},
24             {"face": "Right", "enabled": true},
25             {"face": "Down", "enabled": true},
26             {"face": "Up", "enabled": true}
27         ]
28     },
29     "OffAxisData": {
30         "canvasWidth": 0,
31         "canvasHeight": 0,
32         "cameras": [
33             {
34                 "id": "Down",
35                 "roi": {"x": 0.0, "y": 0.0, "z": 0.333333, "w": 1.0},
36                 "lowerLeft": {"x": -1.0, "y": -1.0, "z": -1.0},
37                 "lowerRight": {"x": 1.0, "y": -1.0, "z": -1.0},
38                 "upperLeft": {"x": -1.0, "y": -1.0, "z": 1.0}
39             },
40             {
41                 "id": "Front",
42                 "roi": {"x": 0.0, "y": 0.33333, "z": 0.333333, "w": 1.0},
43                 "lowerLeft": {"x": -1.0, "y": -1.0, "z": 1.0},
44                 "lowerRight": {"x": 1.0, "y": -1.0, "z": 1.0},
45                 "upperLeft": {"x": -1.0, "y": 1.0, "z": 1.0}
46             },
47             {
48                 "id": "Back",
49                 "roi": {"x": 0.0, "y": 0.66666, "z": 0.333333, "w": 1.0},

```

```

50         "lowerLeft": { "x": 1.0, "y": -1.0, "z": -1.0},
51         "lowerRight": { "x": -1.0, "y": -1.0, "z": -1.0},
52         "upperLeft": { "x": 1.0, "y": 1.0, "z": -1.0}
53     }
54 }
55 }
56 }

```

Breaking Down the Display Settings file

Below is a table containing an example of what a DisplaySettings.json file could look like. **This is not a default example**, as in most cases this would be left blank.

Option	Input Type (Default Value)	Effect
Displays	array of displays	<i>options explained below</i>
>targetDisplay	int (1)	Each display corresponds to a screen plugged into the PC. Starting at 0 with the main desktop.
>SourceX	float (0.0)	Start X position of output canvas (measured from top right of window)
>SourceY	float (0.0)	Start X position of output canvas (measured from top right of window)
>SourceWidth	float (0.5)	How much width of the output canvas to display on this target display
>SourceHeight	float (1.0)	How much height of the output canvas to display on this target display

In the above example, which is part of the example code below, Half of the output canvas would be displayed on Screen 1, with the other half of the canvas being displayed on Screen 2.

▼ IglooDisplaySettings.json - example code

```

1  {
2    "Displays": [
3      {
4        "targetDisplay": 1,
5        "SourceX": 0.0,
6        "SourceY": 0.0,
7        "SourceWidth": 0.5,
8        "SourceHeight": 1.0
9      },
10     {
11       "targetDisplay": 2,
12       "SourceX": 0.5,
13       "SourceY": 0.0,
14       "SourceWidth": 0.5,
15       "SourceHeight": 1.0
16     }
17   ]
18 }

```