

StructureUnityUBT Plugin

This plugin uses the Structure Sensor's Unbounded Tracker to map physical motion in the real world to player movement in a Unity scene

A unitypackage and examples scenes are provided in order to get started.

Section 0 - Overview

What's included

The StructureUnityUBT Plugin Package For Unity 5.1+ includes a few prefabs that enable positional tracking around a virtual space with minimal effort to get you up and running quickly. When you activate the plugin, you'll be able to walk around in the real world and have all of your movements translate into the virtual environment. For example, if you walk forward 10 feet, the character in your virtual world will also walk forward 10 feet.

Section 0.1 - Requirements

The plugin was built and tested on Xcode **7.3** and iOS **9.2 & 9.3**.

It is known to be compatible with Unity versions **5.1.3f1**, **5.3.2p4**, **5.3.3f1**, **5.3.4f1**, **2017.1.1f1**, & **2017.2.0f3**.

Using other configurations may be possible, though we've not tested outside of the prescribed environment and cannot guarantee compatibility.

NOTE: Some recent versions of Unity (e.g. 5.2.0, 5.3.0 and 5.3.1) are not supported due to issues with `__declspec` that all iOS builds suffer from.

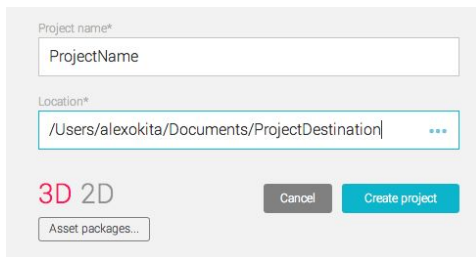
NOTE: We have seen misaligned UI elements on in Unity version 5.3.2f1 and do not recommend using it.

Section 1 - Starting from Scratch

Section 1.1 - Unity3D Project Creation, Package Import, and Build Steps

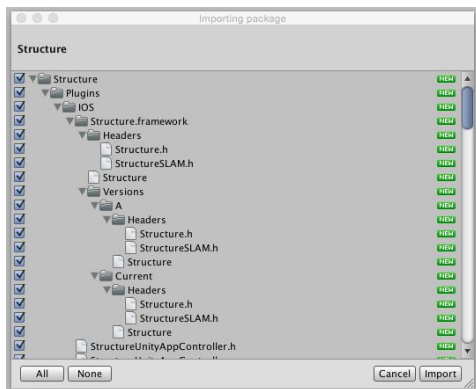
1. Create a new Unity Project:

- Unity Menu: select File → New Project... Save as: Structure Project
- The project name is arbitrary, and can be located anywhere.



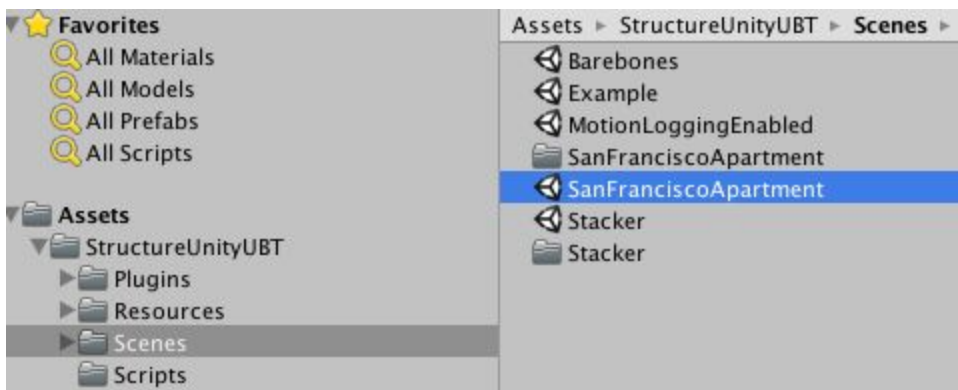
c.

- Unity Menu: select Assets → Import Package → Custom Package...
- Open: **StructureUnityUBT.unpackage**, click [import]



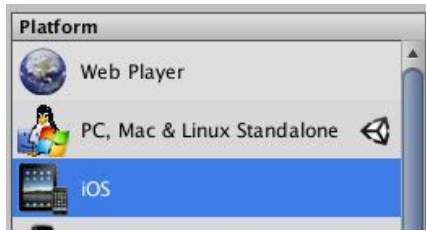
a.

- Project tab: select: **StructureUnityUTB** → **Scenes**
- Open: **Example**, or **SanFranciscoApartment** Either one will do.
 - Example** is a minimal scene
 - SanFranciscoApartment** is a simple architectural example.
 - To build lighting for this scene see [Section 3.5.1.2 Building Global Illumination](#)
 - Select the Scenes directory, the contents of the directories are revealed to the right of the Project window



d.

- Unity Menu: select **File** → **Build Settings**
- Build Settings: select Platform: iOS



a.

8. Build Settings: click [Switch Platform]

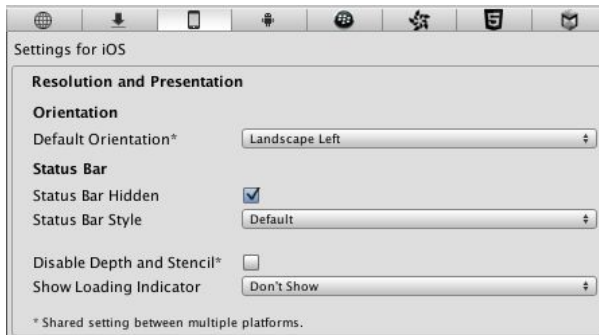
a. Let Unity rebuild / import the assets.

9. Build Settings: click [Player Settings...]



a.

10. Inspector Tab: Resolution and Presentation: Set Default Orientation to: Landscape Left to ensure that the sensor is at the top of the ipad when held in landscape mode.



a.

11. Inspector Tab: Other **Settings** → **Bundle Identifier** change: com.Company.AppName to match your iOS dev certificate.

a. Company and AppName should be something Apple knows about on your developer registration site. Example: com.WidgetCo.StructureWidget

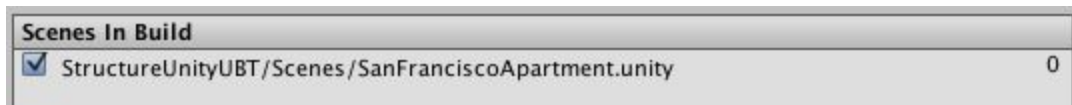


b.

12. The Api Compatibility Level should be changed to .NET 2.0 rather than .NET 2.0 Subset

a.

13. Build Settings: click [Add Current] to add the currently open scene to the built project



a.

14. Build Settings: click [Build] to save the project.



a.

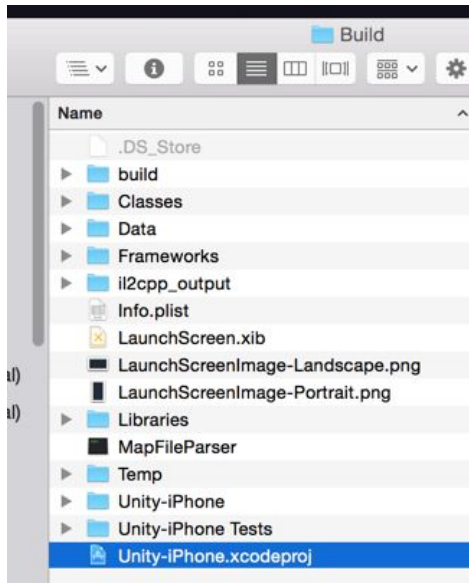
b. Build and run may work, but various scripts may not work on all computer configurations.

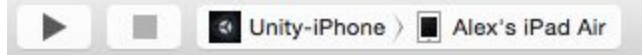
15. Save: For this tutorial, name the saved project as Build and target your desktop folder.

a. The project can be named anything and built to any folder.

Section 1.2 - We're done in Unity... The following is in Xcode.

1. Open the Build directory and open the Unity-iPhone.xcodeproj in Xcode.



- a.
2. Plug in your iOS device
 - a. make sure you have certificates to build to your device.
 - b. <https://developer.apple.com/programs/> for more information.
 3. Xcode: select Product → Run (command+r)
 - a. The Xcode Run button, which is a play button icon, is shown next to the 'Unity-iPhone' target and 'Alex's iPad Air' device.
 - b. or press the above "Play" button.
 - c. This builds your Unity project and deploys to your device.
 - d. Depending on your computer speed, this could take a few minutes.
 4. Unplug device from your computer, plug in your Structure sensor
 - a. The app will launch automatically when deployed by Xcode but will quit once you unplug the device from your computer.
 5. Launch the app.
 - a. Now you have 6 degrees of freedom tracking! Walk around and observe that the player is following your real world movements.

Section 1.3 - Barebones

Minimal example

If you open the Barebones.unity scene you'll be able to take a look at a very minimal approach to using the SDK. There are only three main objects in the scene.



You'll have a Directional Light, Ground Plane, and the [BarebonesPlayer.prefab](#) which contains a PlayerPOV and a Main Camera.

Note: if you like you can even remove the Directional Light and Ground Plane as neither are actually necessary, but both help check that the tracker is moving the player around properly.

Attached to the [BarebonesPlayer.prefab](#) you'll find a Rigidbody, a Capsule Collider, and the `BarebonesTransformController.cs` to which the PlayerPOV and Rigidbody are assigned to Player POV and Player Body respectively.



The code in the `BarebonesTransformController.cs` is contained in the `Start`, `Update` and `HandleTrackerUpdate` methods. The `Start` initializes and assigns the `HandleTrackerUpdate` to the `Structure.Sensor` object. And then `Update` prevents the Rigidbody from sliding around from velocities pushed on the Rigidbody as you move around.

The `HandleTrackerUpdate` method is called by the `Structure.Sensor.TrackerUpdateEvent`. This takes in the `STTrackerUpdate` and moves the Rigidbody.

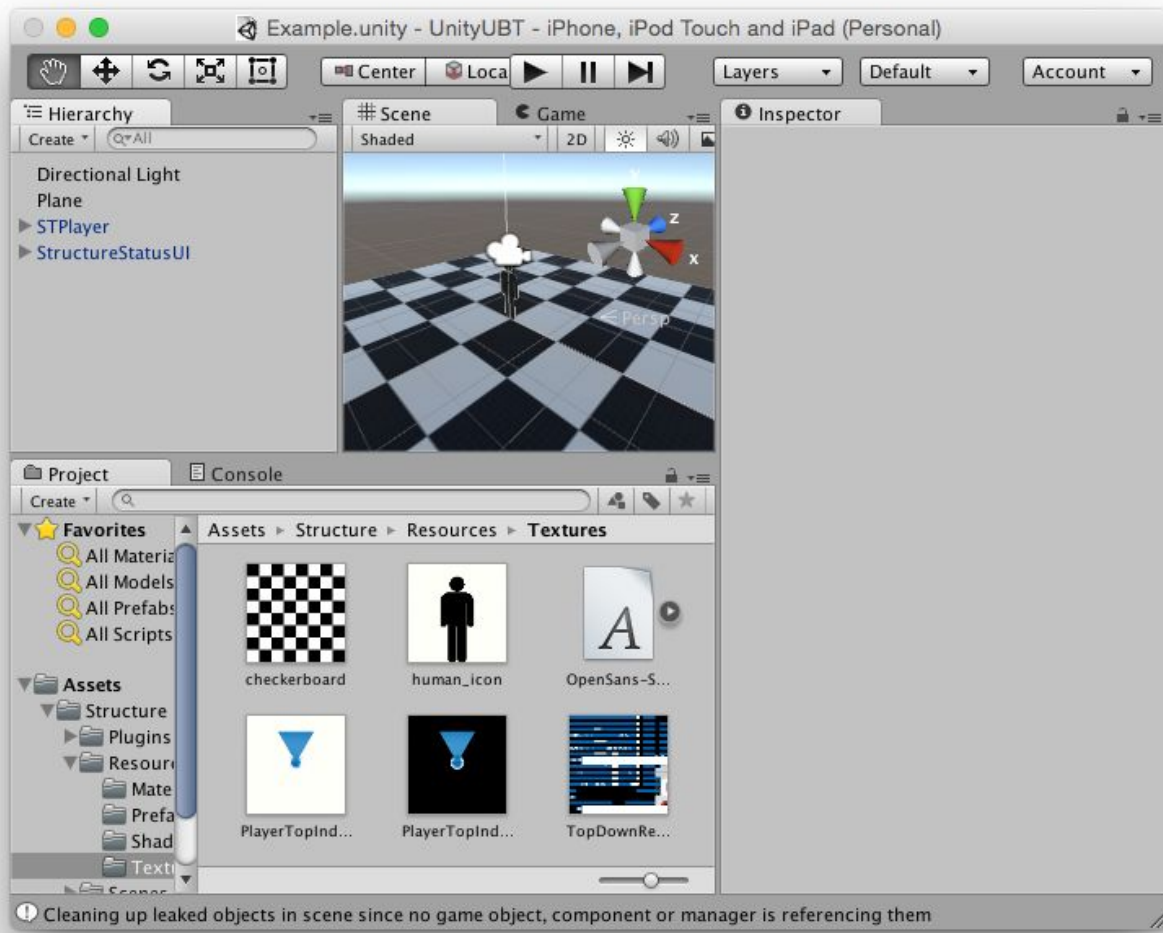
Section 2 - Getting Started (In detail):

Lets take a closer look at what's going on

When you're importing the assets from `StructureUBT.unitypackage` you're allowed to skip a few things in case you're looking to keep your project simple.

- The **SanFranciscoApartment** is a large asset (~19MB) this asset isn't required by the plugin. It's a simple scene to give you a quick asset to work with if you're starting from scratch.
- The **SanFranciscoApartment** directory also contains a scene the prefabs included so you can build the scene and get started right away. If you aren't interested then you can uncheck this when importing the package.

Open the Scenes directory and open the Example scene. This scene contains an [STPlayer.prefab](#), a [StructureStatusUI](#) prefab, and a simple environment that allows you to better observe tracked movement.



Section 2.1 - Running in the Unity Editor for testing:

To start running the game in-editor, press the "Play" arrow at the top of the UI.



It is not possible to preview the full motion characters of Unbounded Tracking on desktop using the editor. The [STPlayer.prefab](#) allows you to spatially navigate the virtual environment like you do for game play testing.

Move around by pressing the following keys:

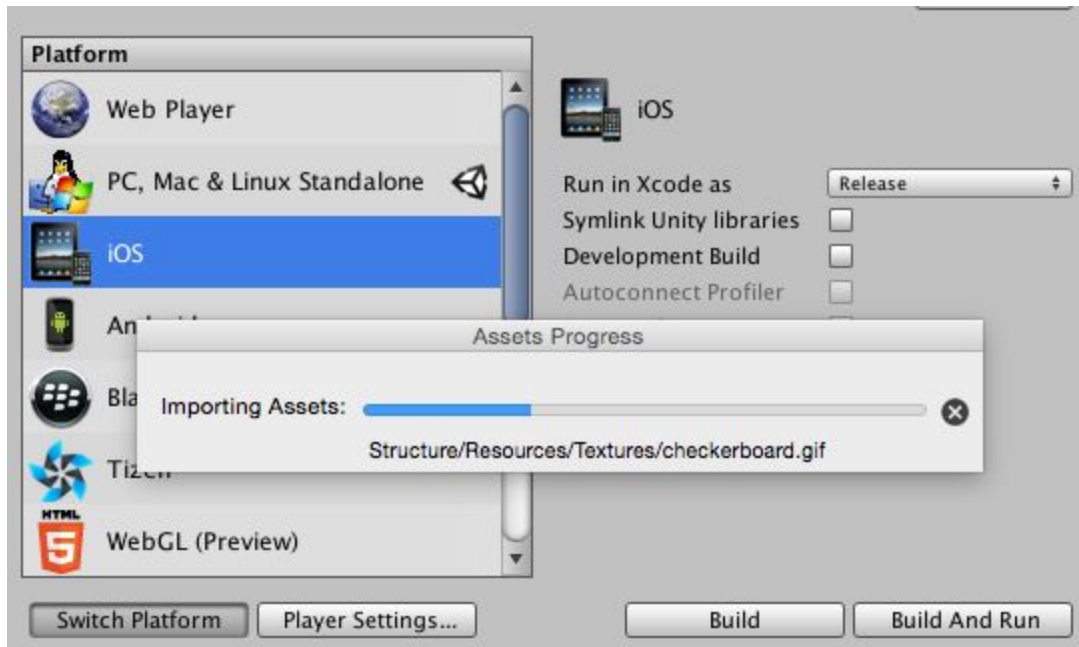
- W - forward
- S - backwards
- A - left
- D - right
- E - look up
- C - look down

With your mouse, you can pan the view or tap to warp to a location, just as you can with a finger when the project is built to an iOS device. For this to work, make sure that the prefab has both the Warp and Pan check boxes enabled.

Section 2.2 - Building to iOS

To deploy this to an iOS device with a structure sensor attached select **File->Build Settings**.

From this dialog select iOS and click on the Switch Platform button. Unity may rebuild the assets once the platform has changed.



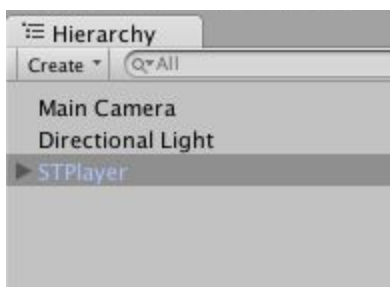
You'll also need to change your iOS App's Bundle Identification. By default it is:
com.Company.ProductName



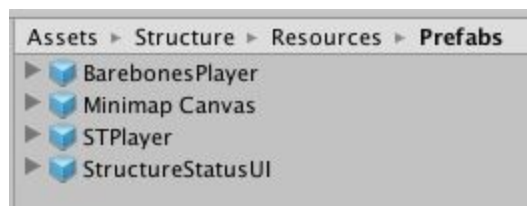
The default will not allow you to build a working Xcode project. You'll have to change this to match an Apple iOS certificate you may have set up. Visit developer.apple.com for more information on iOS development.

Section 2.3 - Starting with a new Scene:

If you want to build a scene from scratch you can include one prefab at a time.



With a new scene you just need to drag the [STPlayer.prefab](#) into the Hierarchy. You'll find the prefab in the Assets/Structure/Resources/Prefabs



For convenience the directory name Resources makes it easier to load assets through script on demand using:

```

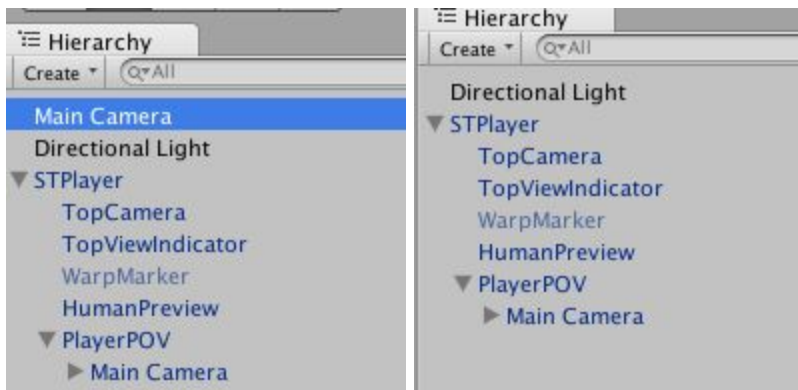
void Start ()
{
    GameObject Player = Instantiate(Resources.Load("Prefabs/STPlayer",
typeof(GameObject))) as GameObject;
}

```

With the above you'll be able to make an instance of the [STPlayer.prefab](#), in this case referenced as Player in your code when you need it.

Next you'll want to delete the default Main Camera that is created by default in a new scene. When the STPlayer is used in the scene it's already built with a Main Camera under the PlayerPOV game object. Leaving the original Main Camera will also result in multiple Audio Listener component objects in the scene.

By adding a new STPlayer.prefab to your scene and deleting the old Main Camera you should be able to Build a new Xcode project and run it on device. This assumes you've changed your player settings from [Section 2.2](#) and entered a usable bundle ID.



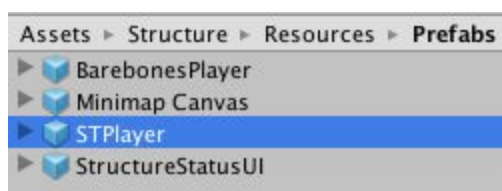
Section 3 - Structure/Resources/Prefabs

There are four relevant prefabs, with scripts attached:

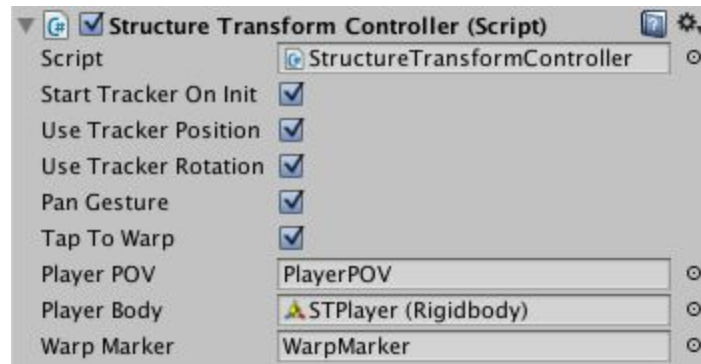
- [STPlayer](#) (StructureTransformController.cs)
- [StructureStatusUI](#) (StructureStatusUI.cs)
- [Minimap Canvas](#) (MiniMap.cs)
- [BarebonesPlayer](#) (BarebonesTransformController.cs)

Each of these prefabs connects to Structure/Plugins/Structure.cs, our wrapper for C#/Unity and the Structure.Framework Any script that intends to manage a relationship with the structure sensor should call Structure.InitStructure(). Refer to [Section 4](#) for additional details.

Section 3.1 - STPlayer



The STPlayer.prefab has a few options on the StructureTransformController.cs shown in the Inspector below:



Section 3.1.1 - Start Tracker On Init

Optionally start the camera tracking when needed. If this is off then the plugin will not send tracker updates until they are requested. To start getting tracker updates you need to make a call to `Structure.Sensor.StartUnboundedTracker()` to start the tracker. Then you can use `Structure.Sensor.StopUnboundedTracker()` to stop the tracker. We'll cover the specific method calls in [Section 4](#) where we go over specifics of each class.

Section 3.1.2 - Use Tracker Position

Applies a force to the `STPlayer.prefab` to push it in the same direction as the tracker is moving. A force is applied rather than a direct movement to allow physics collisions in the game to prevent the player from moving through game objects with collision.

Section 3.1.3 - Use Tracker Rotation

Allows you to pick which parts of the incoming data you'll be using on the `gameObject` that the script is attached to and also gives you the option to turn on and off the different functions even after the updates have started coming from the plugin.

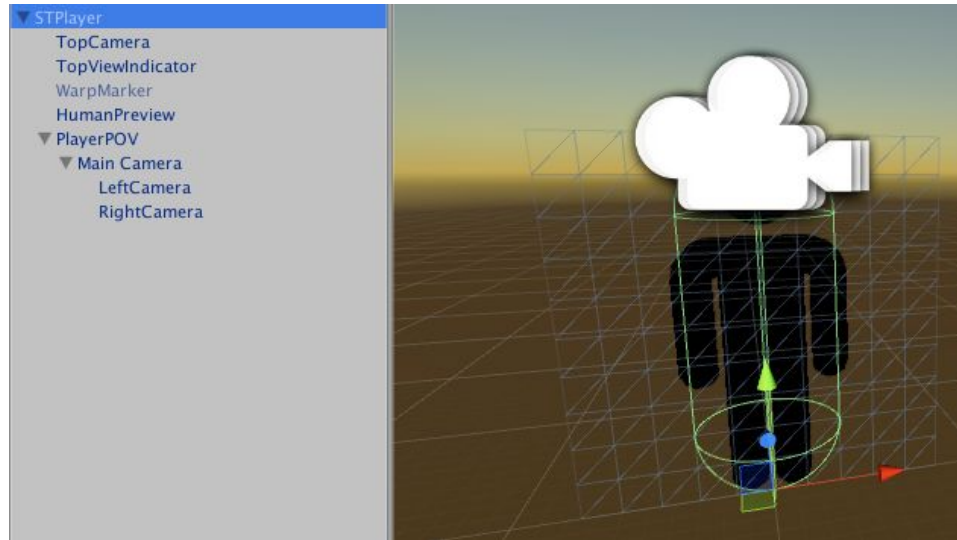
Section 3.1.4 - Pan Gesture

Enables a useful movement utility which allows you to touch drag on the iOS device's screen to rotate the virtual camera. Real world rotation may point you at an actual reality obstacle which doesn't exist in the virtual reality space. In actual reality, turn to face away from obstacles, then use the pan gesture on device to spin the virtual camera toward your intended direction to continue navigation.

Section 3.1.5 - Tap To Warp

Enables the player to tap in the scene to push the `STPlayer gameObject` to the tapped location. The warp object starts off parented to the prefab, but is unparented when the scene is started. This behavior is intended to allow for easier updates in the game scene.

Section 3.1.6 - Prefab Construction



The components assigned to the script are set so that the transform script can function properly and should not be changed.

TopCamera and TopViewIndicator are used by the Minimap to show you a top down map view of your player as represented in the world.

Warp Marker is a simple sphere to indicate the warp target position when you tap in the scene to start a [Tap To Warp](#) movement.

Section 3.2 - StructureStatusUI

The StructureStatusUI is a simple prefab which displays various status update events that originate from the Structure plugin and Framework.

By default, StructureStatusUI includes:

StructureBanner, which shows information from the Structure SDK,
DebugText, a development utility for printing to the screen, and
BatterySlider, which shows the battery level of the Structure Sensor.
It also includes several extra features which are off by default.



Section 3.2.1 - Controlling Sensor Input Manually

In [STPlayer.prefab](#), turn off the Start Tracker On Init check box. Then in StructureStatusUI you may see a hidden set of buttons.

You'll find Start and Stop tracker buttons as well as Start and Stop Color Camera buttons. These buttons and the UI are mostly for demonstration purposes. The code here shows off various features of the plugin in relation to how to use the StructureSDK through the Structure.Framework.

Now you'll be able to start and stop the tracker with function calls made to the Plugin.

These calls into the StructureStatusUI.cs script result in method invocations in Structure.Sensor. See [Section 4](#) for more details on the various methods we use to communicate with the structure sensor.

Section 3.2.2 - BatterySlider

The BatterySlider object is a UISlider added to the Battery Slider in the StructureStatusUI component on the parent StructureStatusUI GameObject.



This gets updates from the plugin while the Structure Sensor is plugged in and after the plugin has been initialized.

StructureBanner and StructureMessage

These work together to display various status messages which arrive from the plugin as various events occur.

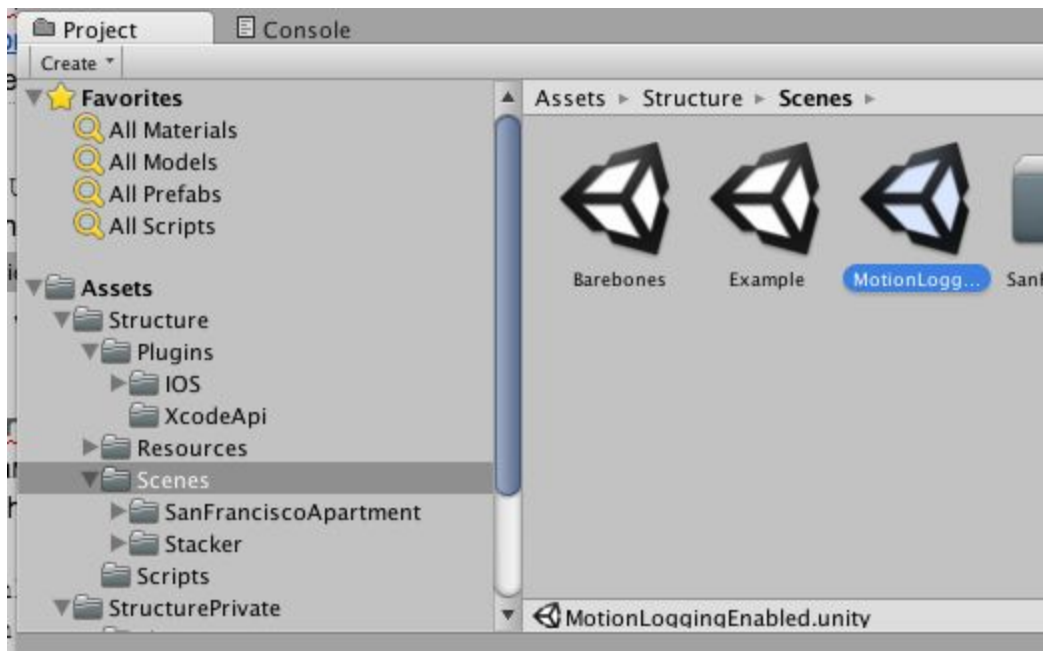
Messages include the following:

- STTrackerStatusGood
- STTrackerStatusDodgyForUnknownReason
- STTrackerStatusFastMotion
- STTrackerStatusTooClose
- STTrackerStatusTooFar
- STTrackerStatusRecovering
- STTrackerStatusModelLost

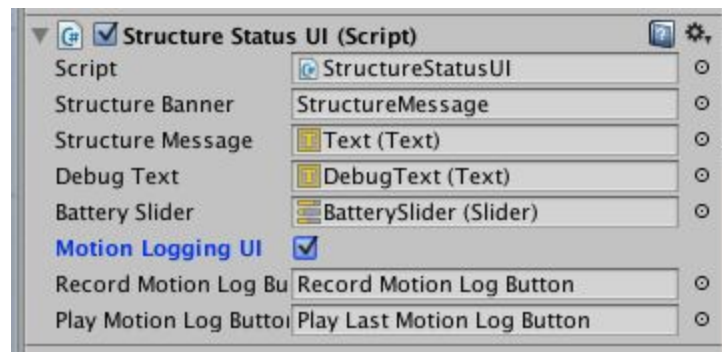
The status messages will notify on each frame as they are received. For the most part you can expect STTrackerStatusGood which means tracking is working and pose updates are estimated properly.

Section 3.2.3 - Recording Motion

A sample scene has been provided called `MotionLoggingEnabled` located in the Scenes directory. Deploy this with the Record and Playback buttons enabled.

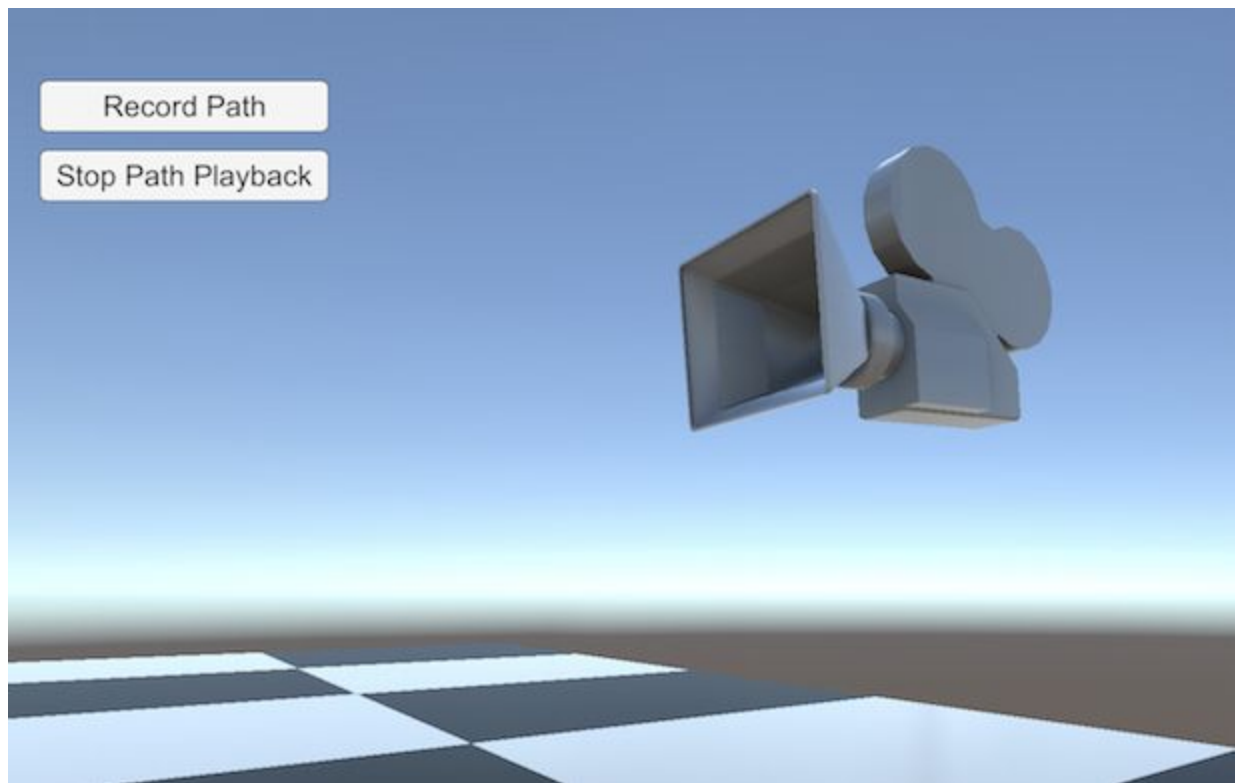


To turn on the Motion Logging UI and its features, check the “Motion Logging UI” box on the Structure Status UI Script:

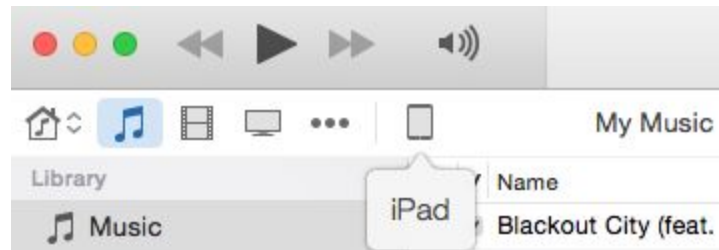


This will enable two buttons for recording and playing back motion logs.

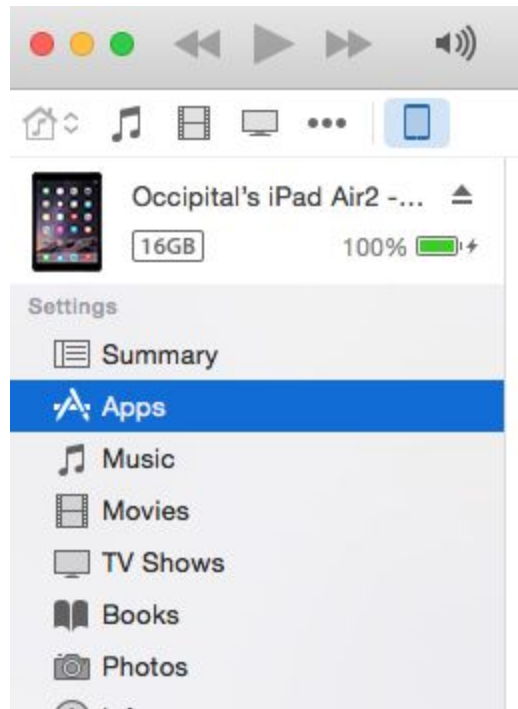
To start recording a path, tap the “Record Path” button. Have some fun with it. Go up and down, too. When you’re satisfied, press the “Stop Recording” button. The plugin writes the path to the iTunes File System space, and the “Play Last Path” button should be visible. Move to the side (so you aren’t in the middle of playback when it starts), and press the button to play back to the path.



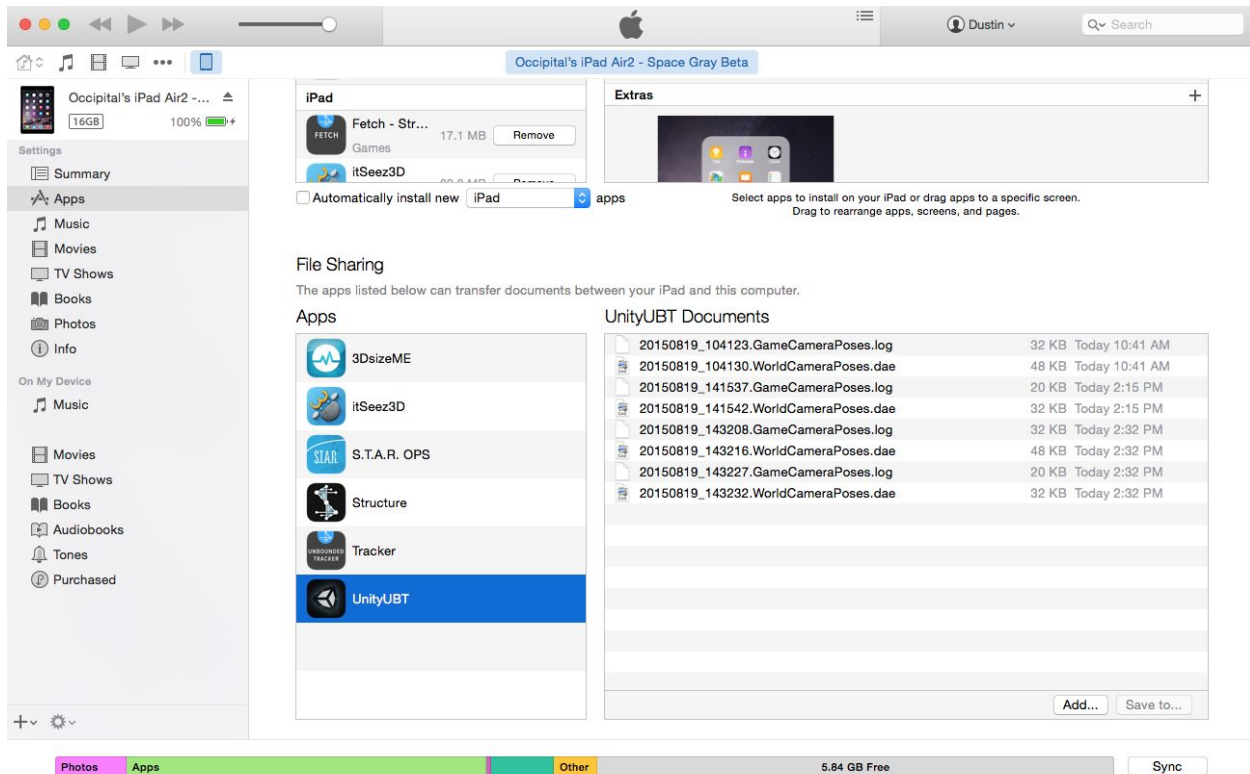
You can continually record and play new paths. We save the motion path files to the iTunes File System within the built Unity app. To access this, plugin your iOS device and open iTunes. Then, select your device.



Select “Apps” in the left side panel



Scroll the right panel down to the “File Sharing” section, and select the your app.



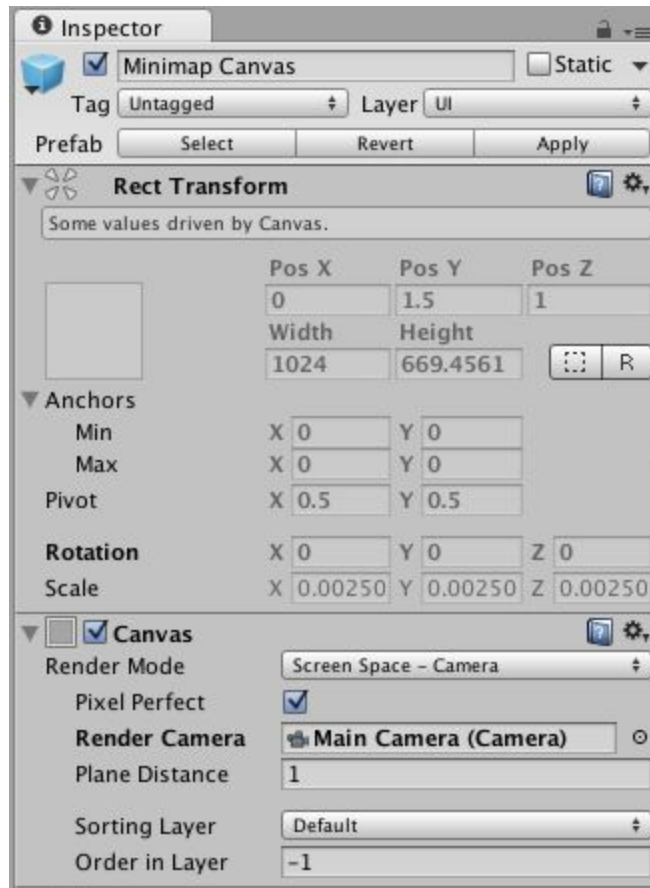
You will see a list of files that come in pairs based on the same date and time stamp. The GameCameraPoses.log files are a simple comma-separated-value (.csv) format representing camera motion in the game environment, while the WorldCameraPoses.dae files are the COLLADA format files representing movement with respect to the real world. Game motion and world motion can differ based on in-game actions, such as panning or warping, so this is why we treat them differently.

Select any files you want to export to your computer, and select “Save to...” in the lower right to export them.

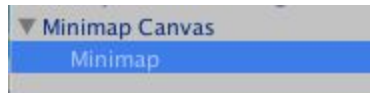
Special note: COLLADA does not support a native 4x4 matrix transformation format, so we write to the file as a translation or rotation. If you use this to make smoothed animations, this can lead to gimbal lock issues, which appear as sudden 180 flips in playback. You may need to manually edit rotations.

Section 3.3 - MiniMapCanvas

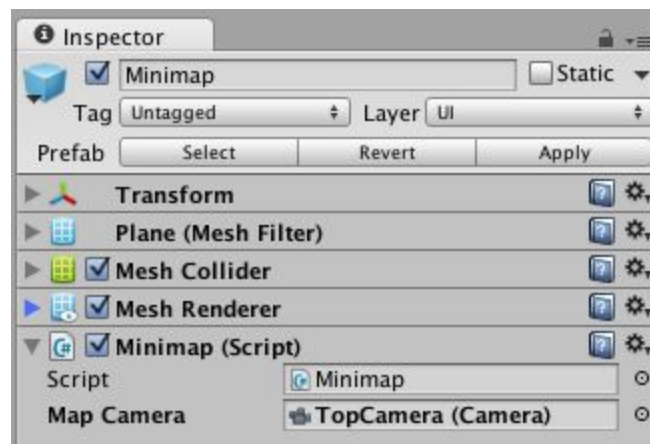
This is a basic top down camera that works with STPlayer to allow you to tap on the map and set the position of the player to the point that was tapped on the map. When you drop the MiniMapCanvas.prefab into a scene you will need to hook up two components. The first requires the Main Camera added to the Canvas Render Camera. This will allow the map to draw to the player's view.



To allow the player to click on the minimap to warp to the point selected in the mini map you'll need to select the child of the Minimap Canvas in the Hierarchy.



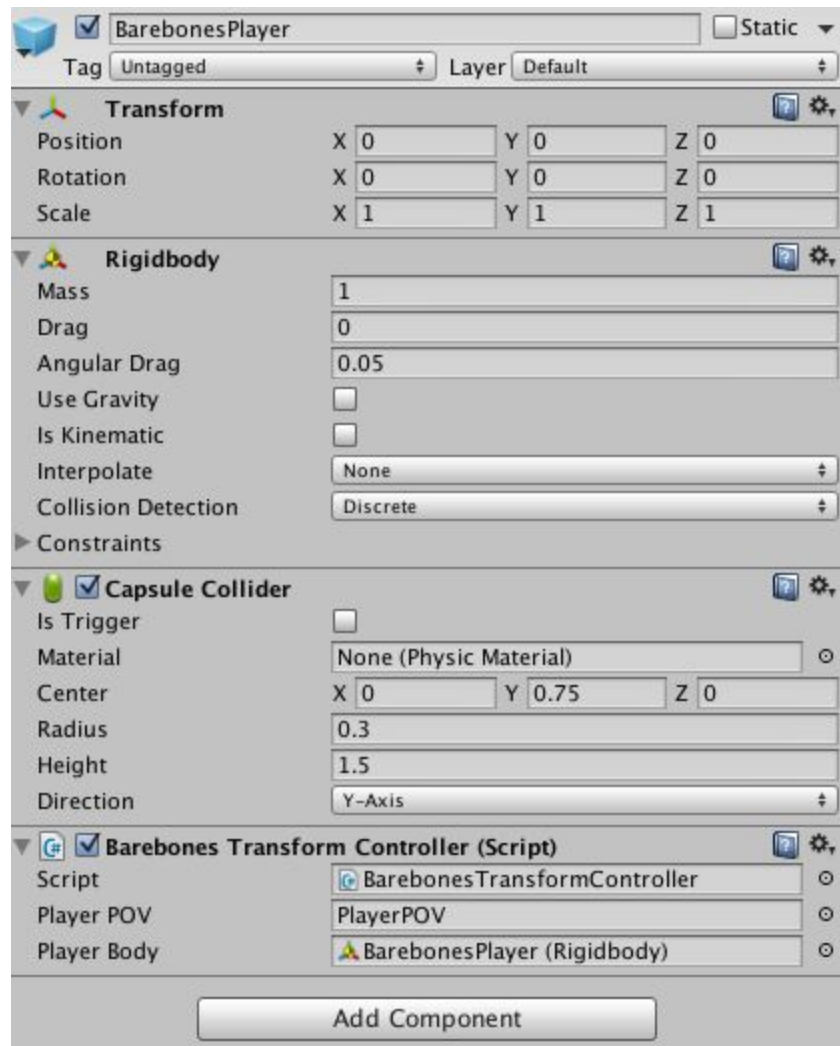
Then add the STPlayer's TopCamera to the Minimap Map Camera in the Inspector panel.



This should be setup ahead of time in the [Example.unity](#) scene file.

Section 3.4 - BarebonesPlayer

This class is the bare minimum needed to use the tracker updates.



There are no options in the inspector, and when the scene begins the Start() function will call the required initializations to initialize and start the motion tracking. If you want to modify how tracker movements are consumed by the player you'll want to do so inside of the HandleTrackerUpdates method which is called by the plugin.

Section 3.5 - Example Scenes

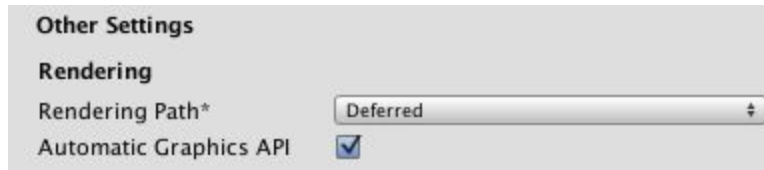
The Example scenes are intended to provide some basic inspiration for your own projects as well as a template to show you how we've intended the plugin to be used. This is by no means a restriction on how you're able to use the plugin. If you've got some clever method of using the tracked position and rotation you're encouraged to use the input data however you feel.

Section 3.5.1 - SanFranciscoApartment.unity

This is a simple example showing off what a home visualization might look like. The geometry is simple and has a simple collision set such that the [STPlayer](#) cannot walk through the walls. You'll want to spend some

time bake the lighting to get the best view of this apartment. More on Unity's lighting tools can be found [here](#) where you'll find an overview of Unity's lighting system.

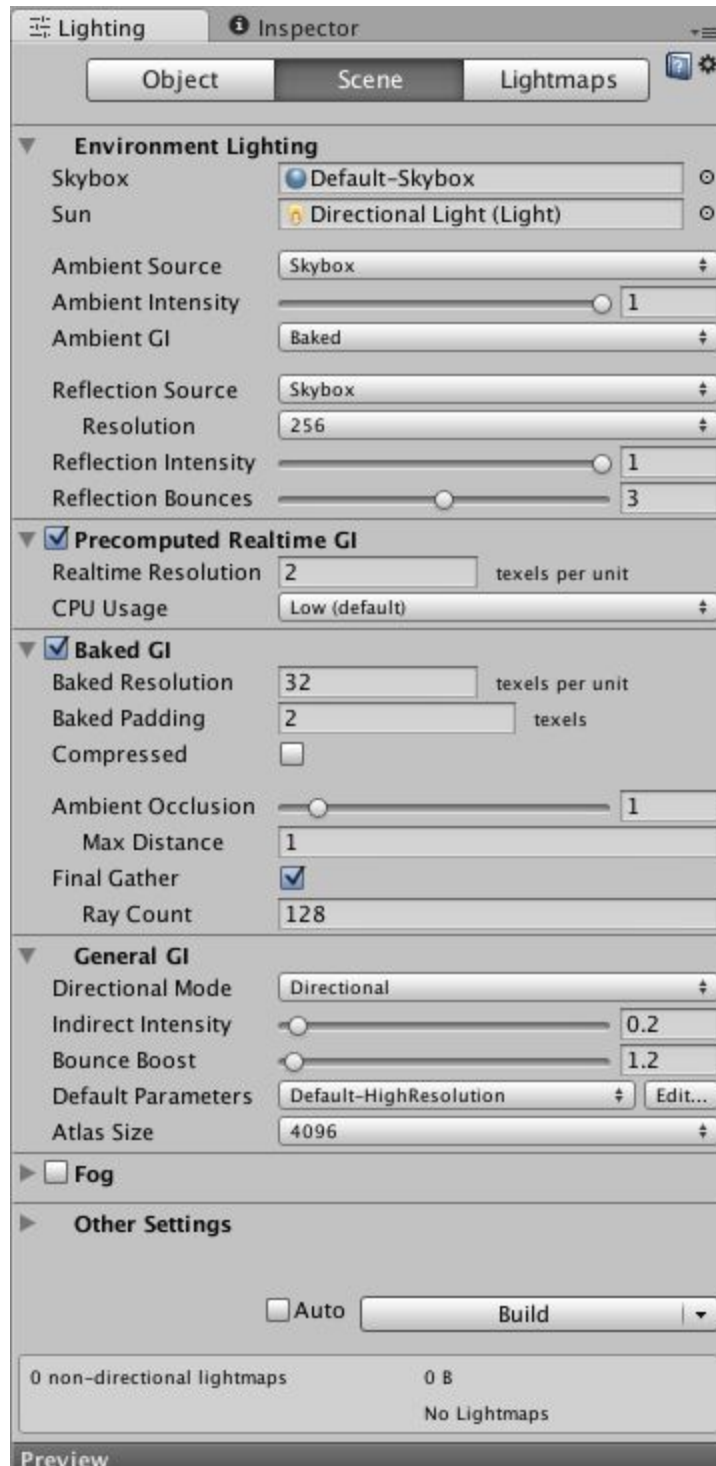
For the best rendering results you'll want to set the graphics api to automatic.

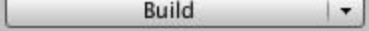


Depending on your device this will usually set the api to Metal or OpenGL ES 3. Note: The generated light map and direction maps that are created by Unity can be quite large. Settings for smaller file sizes are included to avoid storing the larger higher quality files.

Section 3.5.1.2 - Building Global Illumination

To build the lighting for the scene select: Window -> Lighting this will open the following popup.



On the lower right, click on Build  to generate the necessary lighting data. The settings provided have been lowered to decrease build times, but this results in a lower fidelity generated lighting texture. To increase the quality select a higher Resolution setting.

Resolution 

Increasing the resolution of the generated light map and various quality settings can dramatically increase the time it takes to generate a high quality light map.



For further information on light maps and quality settings refer to the Unity Manual section on [Global Illumination](#).

Section 3.5.2 - Barebones.unity

This is the almost absolute minimum scene required to make use of the motion tracking provided by the plugin. The only extra items include the directional light and the ground plane. The [BarebonesPlayer.prefab](#) is a super minimal prefab that makes use of plugins tracking capabilities.

Section 3.5.3 - Example.unity

This is a basic scene which features the [STPlayer](#), the [StructureStatusUI](#) and the [MiniMap](#). The directional light and plane can be replaced by your own assets for visualizing a more interesting scene.

Section 3.5.4 - Stacker.unity

Located in Assets/Scenes/Stacker/Stacker.unity is a simple physics based puzzle game where you try to pull out blocks and stack them on the top of the column of blocks. The [STPlayer](#) has a few settings disabled to allow the touch interaction for grabbing and moving the blocks in the scene.

Section 4 - StructureInterop

The Wrapper class contains all of the methods that are required to interface with the plugin.

The `StructureInterop` class located in `Structure.cs` is an internal class as it's only used by `Structure`. This class contains all of the direct calls into the `UnityInterop.mm` module located in **Structure/Plugins/IOS**.

To initialize the plugin and get it ready to send events and data to Unity `_initStructureSensor()` should be called first. This creates the `_sensorController` object used to communicate with the sensor. The `_initUnboundedTracker()` call initializes the tracking system.

Call `_startUnboundedTracker()` to begin sending transforms to Unity. `_stopUnboundedTracker()` will deactivate the tracking, and Unity will stop receiving transforms from the plugin.

`_registerTrackerEventCallback()` will send a function pointer from Unity to the plugin. This allows the plugin to call the function directly.

When `_getBatteryChargePercentage()` is called it returns an int value between 0 and 100 indicating the battery charge percentage. In the [StructureSensorUI.prefab](#) you'll see a system in place which makes regular calls to this method in the plugin then sets the prefab's battery slider component to the value returned by the method.

`statusUpdateInterop(SensorEvent evt, int val)` various events in the plugin will call this function sending a `SensorEvent` enumeration and a value. This is how the battery is updated as well as the banner when the sensor is plugged in and unplugged. Various tracking events also call this delegate.

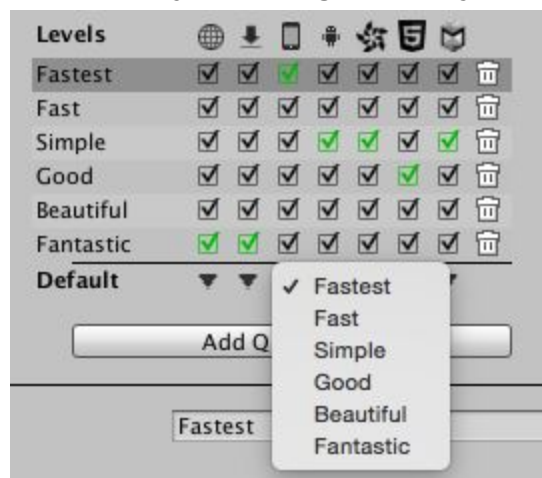
`_getBatteryChargePercentage()` will tell the plugin to provide a `statusUpdateInterop()` event with `SensorEvent.BatteryLevel` with a value between 0 and 100. `_getSensorStatusUpdate()` will ask the plugin to send a status for connected and a status for streaming.

Section 5 - Options

Section 5.1 - Improving performance

Depending on your device you have the option to change the quality settings within Unity to improve the rendering speed. This will have a direct effect on the frame rate and the perceived quality of the unbounded tracking.

To change the quality settings select **Edit->Project Settings->Quality**



In the inspector Simple is used as the quality used for the iOS device. Select either Fast or Fastest for better performance.

Section 5.2 - Modifying the FOV of the POV

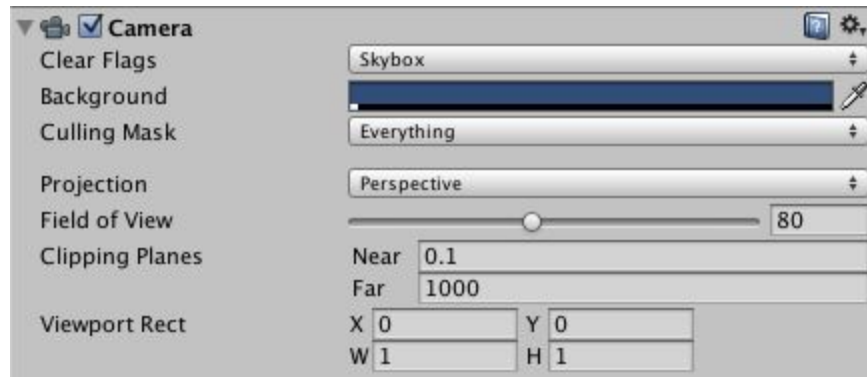
To modify the FOV of the camera select the Main Camera component of the STPlayer



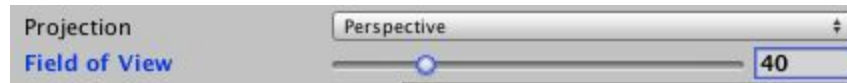
Or in BarebonesPlayer select the Main Camera under the same PlayerPOV node.



To change the fov select find the Field of View in the Inspector Panel.



You're able to change the Field of View here. By default the FOV is set to 80, but some may find this to be too wide.



A setting of 40 degrees will approximate a 50mm "film" lens.

Section 6 - Troubleshooting

Some of the more common problems you might encounter with Unity development.

There are a lot of basic problems when working with a plugin and Unity. A lot of these are easily fixed and are usually a result of version updates made by Unity.

Section 6.1 - Warnings

With the current release of Unity there may be various warnings related to the Xcode api. These are known and Unity is working on updates to their Xcode project generation API.

Section 6.2 - Errors

We have encountered various errors when using iOS 9 and Xcode 7, we recommend against using these until we have fully claimed support for these beta versions. We're also noticing some problems when using .NET 2.0 rather than .NET 2.0 Subset

The Api Compatibility Level is easy to miss, but having this set to Subset may lead to difficult to track down bugs after the Xcode project has been generated.