ForceX 3D Radar EX

Version 1.2.1b

INDEX

Project / Scene Setup

Asset Import Scale

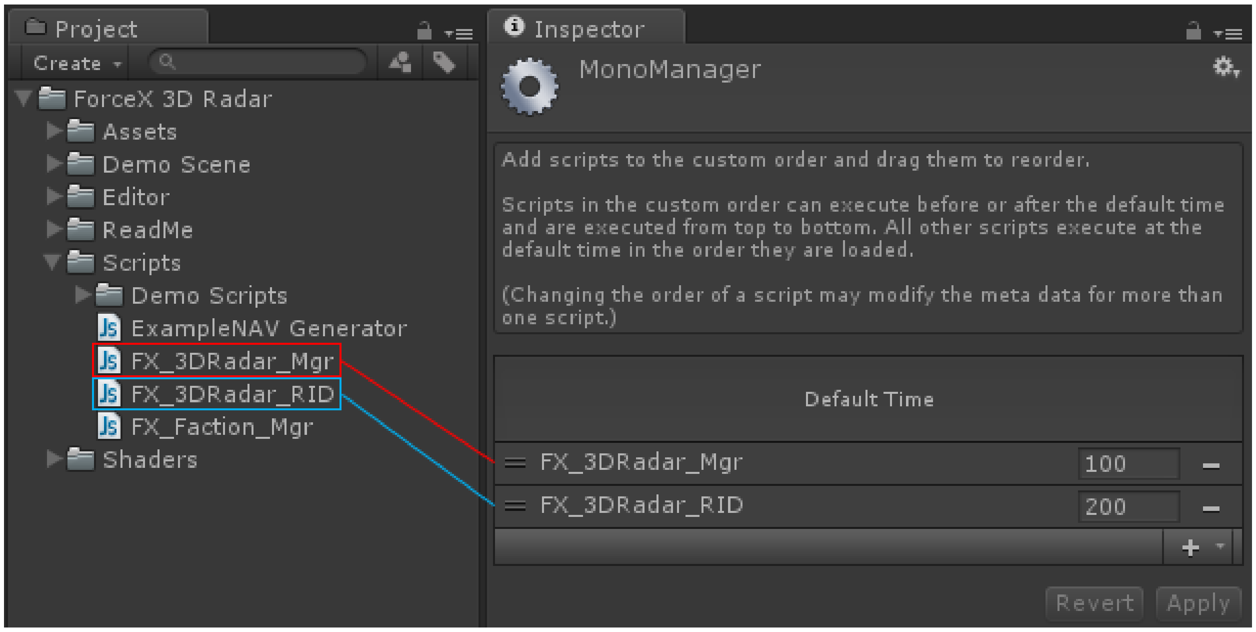
The first thing we are going to want to do is insure that the “**Scale Factor**” for our import meshes is setup correctly.

1. Navigate to ForceX 3D Radar → Assets → Meshes
2. Make sure that the Radar2011 & 2013 have a Scale Factor = 1 in the Import Settings section of the inspector.

Setting The Script Execution Order

Sense both script execute during Late Update we need to setup an execution order. We do this because the FX_3DRadar_RID script is dependent on information from the FX_3DRadar_Mgr script, so we simply want to insure that the Mgr script is executed first before being accessed by the RID script.

- In the Project window go to the Scripts folder and select the “**FX_3DRadar_Mgr”** script.
- In the Inspector window click on the button “**Execution Order**”.
- Drag the “**FX_3dRadar_Mgr”** script into the script execution order window and place it below Default Time.
- Drag the “**FX_3DRadar_RID”** script into the script execution order window and place it below the “**FX_3DRadar_Mgr”** Script.



Add scripts to the custom order and drag them to reorder.

Scripts in the custom order can execute before or after the default time and are executed from top to bottom. All other scripts execute at the default time in the order they are loaded.

(Changing the order of a script may modify the meta data for more than one script.)

Default Time

FX_3DRadar_Mgr	100
FX_3DRadar_RID	200

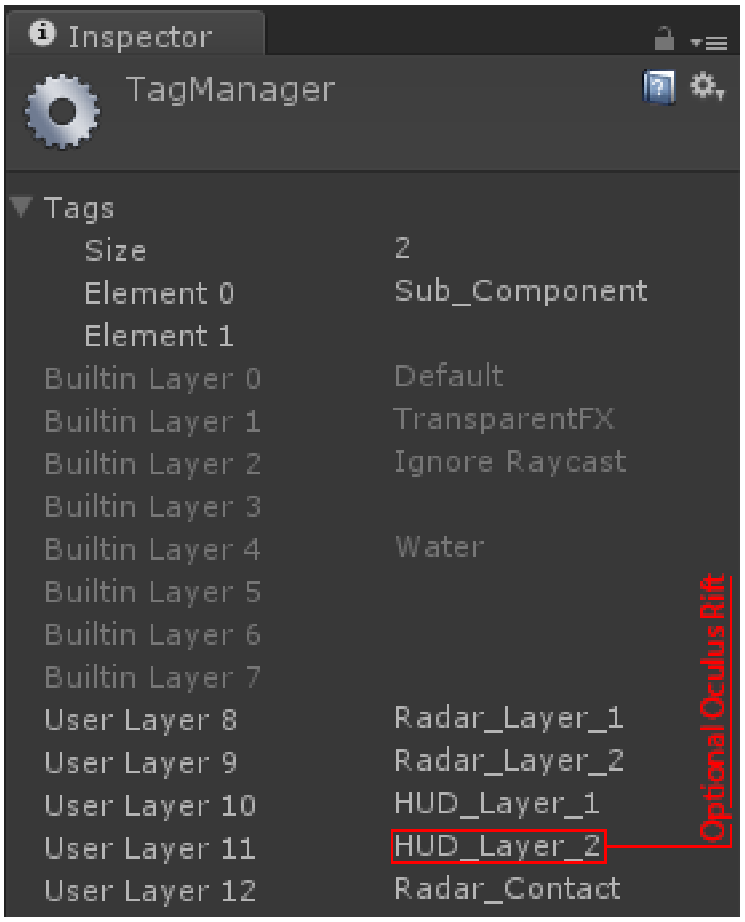
Setting Up The Layers & Tags

1. Create five new layers:

- Radar_Layer_1
- Radar_Layer_2
- HUD_Layer_1
- HUD_Layer_2 (Optional : Oculus Rift)
- Radar_Contact

2. Create one new tag:

- Sub_Component



Tags	
Size	2
Element 0	Sub_Component
Element 1	
Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	Radar_Layer_1
User Layer 9	Radar_Layer_2
User Layer 10	HUD_Layer_1
User Layer 11	HUD_Layer_2
User Layer 12	Radar_Contact

It is important to setup the layers with the names shown above as the Radar Manager will automatically assign the layers based on their names.

Radar_Layer_1 & 2 will be used to separate our radar contacts graphical elements onto their own rendering group with the Radar's camera.

HUD_Layer_1 & 2 will be used to separate our main HUD graphical elements onto their own rendering group with the Players HUD camera. HUD_Layer_2 is used only with the Oculus Rift. This separates the Left and Right graphical elements onto their own cameras.

Radar_Contact layer is used by the radar for the detection of radar contacts when the radar is set to use Blind Radar.

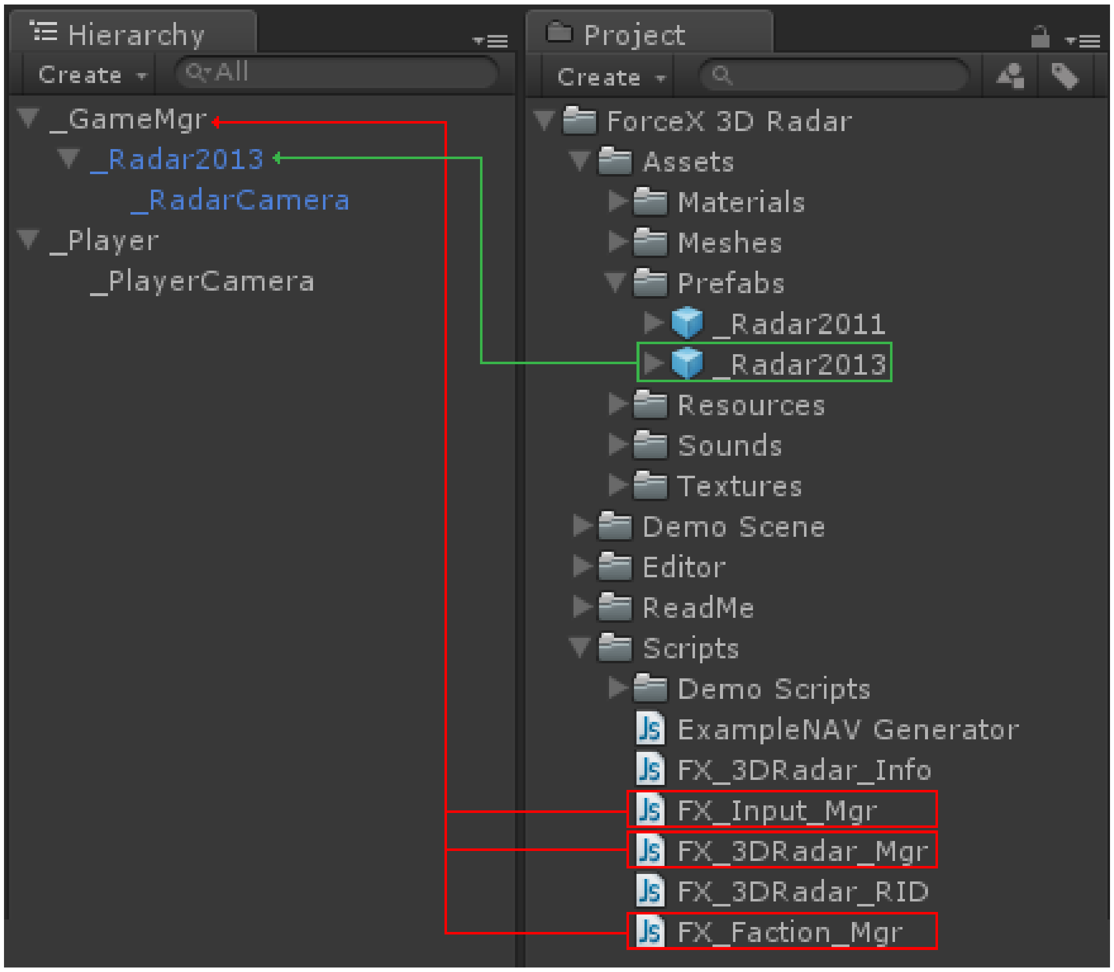
Sub_Component tag is used to define any child gameObjects on a given target that are to be marked as a targetable SubComponent.

Creating The Game Manager & Player

1. Create a new GameObject and name it " **_GameMgr** " and zero out its position and rotation values in the inspector and then assign both the "**FX_3DRadar_Mgr.js**", "**FX_Input_Mgr.js**", & "**FX_Faction_Mgr.js**" script. These scripts can be found in the "**Scripts**" folder.

2. Find the prefab object "**_Radar2013**" located "**Assets → Prefabs → _Radar2013**" and assign it as a child of the "**_GameMgr**".

3. Create a new GameObject to represent your player and name it "**_Player**". Then assign your default scene camera as a child of the "**_Player**" GameObject and zero out its position and rotation values in the inspector. Change the cameras name to "**_PlayerCamera**"

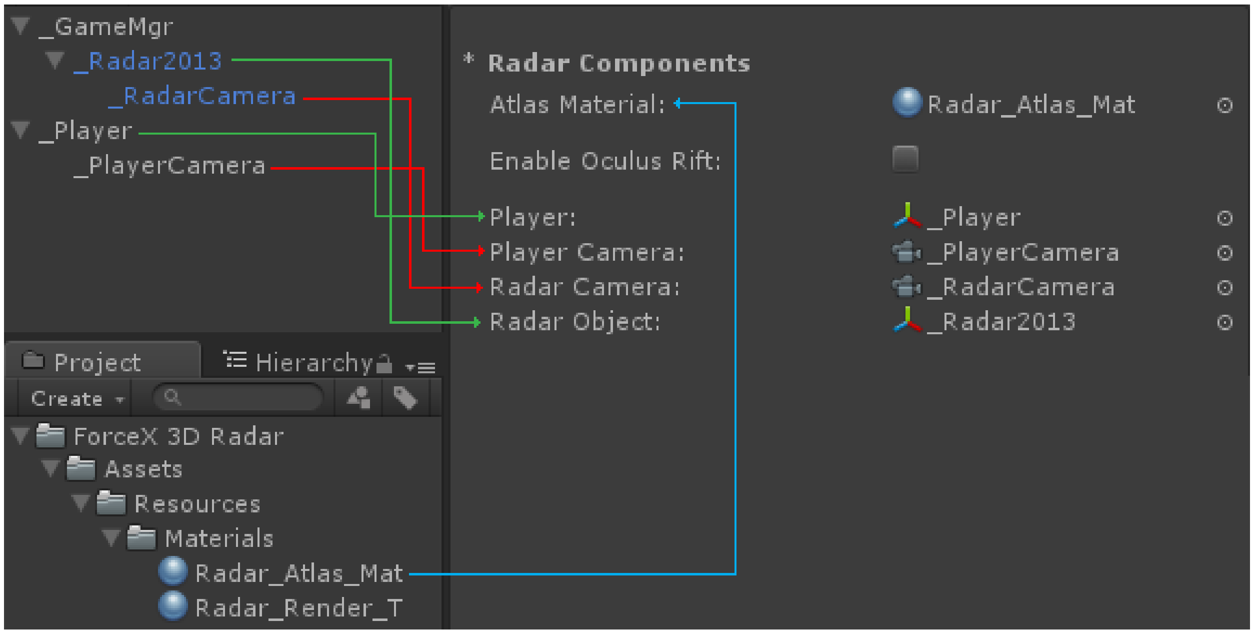


Setting Up The Radar Manager

If you are using the default assets then the **FX_3DRadar_Mgr** will automatically find and assign the components to their respected fields. If you are using your own custom assets then you will need to enable **Assign Custom Assets** in the inspector and assign your components as necessary. Below is an example on how to assign the required components to the inspector using the default assets.

1. Select your "**_GameMgr**" and navigate to the "**FX_3DRadar_Mgr→ Radar Components**" in the inspector and assign the following.

- Atlas Material = "**Radar_Atlas_Mat**" Found : **Assets→ Resources → Materials→ Radar_Atlas_Mat**
- Player = "**_Player**"
- Player Camera = "**_PlayerCamera**" Found : **_Player→ _PlayerCamera**
- Radar Camera = "**_RadarCamera**" Found : **_GameMgr→_Radar2013→_RadarCamera**
- Radar Object = "**_Radar2013**" Found : **_GameMgr→ _Radar2013**



Radar Components
Atlas Material: Radar_Atlas_Mat
Enable Oculus Rift:
Player: _Player
Player Camera: _PlayerCamera
Radar Camera: _RadarCamera
Radar Object: _Radar2013

Setting Up The Faction Manager For The 3D Radar

By now you should have followed the steps to get the radar system setup in your project. If not then go back and read the section on how to do this.

1. In the Hierarchy view select the **_GameMgr** gameObject.
2. In the inspector click the Start Faction Manager button.
3. Set the number of factions by adjusting the **Factions slider's** position. Up to 32 factions can be created.
4. In the new text fields assign names for each of the faction groups. If not the default names will be used.
5. Click the **Faction Relations** button to switch to the relationship matrix window.
6. Assign a cutoff value for the Hostile and Friendly fields. Any number below the value assigned for Hostile will be considered hostile, while any number above the value assigned for Friendly will be considered to be friendly. Any number between these two values will be neutral.
7. Assign the players Faction by adjusting the slider's position.
8. Assign starting relationship values for the Player and each faction. **Optional**
9. Assign starting relationship values for each of the factions in the Faction Relationship Matrix.

Assigning Factions

The 3D Radar stores all individual faction information inside the FX_3DRadar_RID so any gameObject with this script attached will have access to the Factions list stored in the Faction Manager. Assigning a Faction to any given gameObject is as simple as adjusting the Faction Slider's position.

Depending on the values assigned in the Faction Relation Matrix each of these gameObjects will now appear as Friendly, Neutral, or Hostile towards the player. Don't worry this doesn't just work for the player, any gameObject can compare their relationship with any other gameObject; for this we need to understand how the Faction Manager works.

Accessing The Faction Manager Through Script

How The Faction Manager Works

At its core the faction system is a Dictionary. Since a Dictionary stores information through the use of a Key, this means that in order to retrieve any information from the Dictionary we must have the Key. This leaves us with the question, what is the key? The answer to that is quite simple. The Faction Manager will assign each faction a number known as a FactionID. While this number by its self has no immediate use, but when added together with another faction's ID number, will give us the Key we need in order to look up the relationship values for these two factions in the Dictionary. Once we have the Key and access to the Key's paring value we can either simply use the returned value for our own purpose, such as checking if a faction is Hostile or Friendly, or we can update the value to something different.

Let's take a look at how this is done.

Check The Relationship Value Between Two Factions

1. Get the FactionID's for the two objects you need to compare.
- FactionID_1 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[2];
- FactionID_2 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[2];

2. Add both values from step 1 together. This will give you a unique key that is used to look up the relationship value for these two factions.
- var ThisRelation : int = FX_Faction_Mgr.FactionRelations[(FactionID_1 + FactionID_2)];
- The value returned is the numerical relationship for these two factions.

Change The Relationship Value Between Two Factions

1. Get the FactionID's for the two objects you need to update.
- FactionID_1 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[2];
- FactionID_2 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[2];

2. Add both values from step 1 together. This will give you a unique key that is used to access and change the relationship value between these two factions.
- FX_Faction_Mgr.FactionRelations[(FactionID_1 + FactionID_2)] = Your Value;

Check The Players Relationship Value With A Faction

1. Get the Faction number for the faction you want to compare.
- Faction_N = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[1];

2. Get the value form the PlayerRelations array.
- Value = FX_Game_Mgr. PlayerRelations[Faction_N];

Change The Players Relationship Value With A Faction

3. Get the Faction number for the faction you want to compare.
- Faction_N = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction[1];

4. Get the value form the PlayerRelations array.
- FX_Game_Mgr. PlayerRelations[Faction_N] = Your Value;

Changing The Players Faction

FX_Faction_Mgr.PlayerFaction = Your Value;

Changing The Others Faction

The Radar system stores the faction values for any given object other than the player in one place; this is in the FX_3DRadar_Rid.js script. The current faction value is stored in the ThisFaction[1] : int, and the current factionID value is stored in ThisFaction[2] : int. These values can be updated manually by executing some code in a certain order. There is also a ThisFactionTemp value. This value is used to compare it's self against the currently assigned faction number and automatically update all faction information automatically if any change occurs to the faction value. However there may be instances where you need to update the faction number and the faction ID number manually. Below these methods are described.

Auto Update

- FX_3DRadar_RID.ThisFaction = Your Value;

Full Manual Update

- FX_3DRadar_RID.ThisFaction = Your Value;
- FX_3DRadar_RID.ThisFactionID = FX_Faction_Mgr.FactionID[ThisFaction[1]];

Things To Avoid

One thing to note is that you cannot compare the values of two objects that are the same faction. If you are creating a function that checks the relationship values for a bunch of objects that may share the same factions as the object that is doing the checking then you will want to omit objects of the same faction.

If(FactionID_1 != FactionID_2) { Do Something }

Doing this simple check will avoid generating a Key value that does not exist in the Dictionary.

Creating A Target

Now that the project and initial scene have been setup we will create and configure a new GameObject for the radar to target.

1. Create a new GameObject and name it “**Target**” and assign it the “**FX_3DRadar_RID.js**” script. This script can be found in the “**Scripts**” folder.

That’s it you have created a target. You can setup your target any way you require.

Destroying A Target

In order to properly destroy a radar target you must access the targets FX_3DRadar_RID script and call the DestroyThis() function.

JS

YourTarget.GetComponent(FX_3DRadar_RID).DestroyThis();

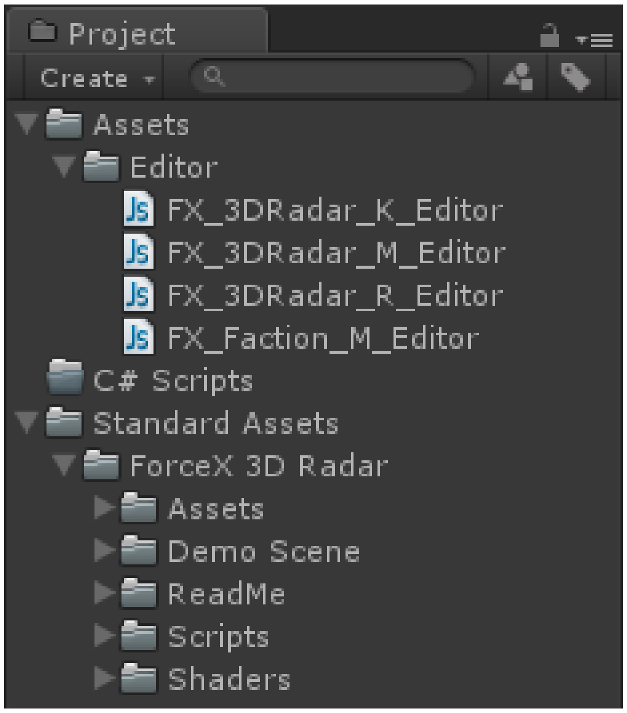
C# (See Using The Radar With C# To Setup Directory Structure)

YourTarget.GetComponent<FX_3DRadar_RID>().DestroyThis();

Using The Radar With C#

The 3D Radar is written in JS so there is some prep work that must be done in order to be able and access the 3D Radars scripts from a C# script.

- Create a new folder named Assets and make sure that it is a root folder.
- Create a new folder named Standard Assets and make sure that is a root folder.
- Move the Editor folder from the ForceX 3D Radar directory and place it inside the Assets folder you just created.
- Move the ForceX 3D Radar folder and place it inside the Standard Assets folder you just created.



```
Assets
    Editor
        FX_3DRadar_K_Editor
        FX_3DRadar_M_Editor
        FX_3DRadar_R_Editor
        FX_Faction_M_Editor
    C# Scripts
Standard Assets
    ForceX 3D Radar
        Assets
        Demo Scene
        ReadMe
        Scripts
        Shaders
```

Keyboard Assignments

E : Closest Hostile
R : Next Hostile
F : Previous Hostile
T : Next target
Y : Previous target
U : Closest target
C : Clear target

M : Next SubComponent Target
N: Previous SubComponent Target
B: Clear SubComponent Target

K : Display NAV Scrolling List
L : Display Target Scrolling List
H: Display Only Hostile Contacts

Z : Switch 3D <-> 2D Radar Modes

X: Toggle Target Indicators