PROCEDURAL PLANETS

MANUAL

Procedural Planets is an asset for Unity that mathematically creates high-resolution planets.

Table of Contents

Getting Started

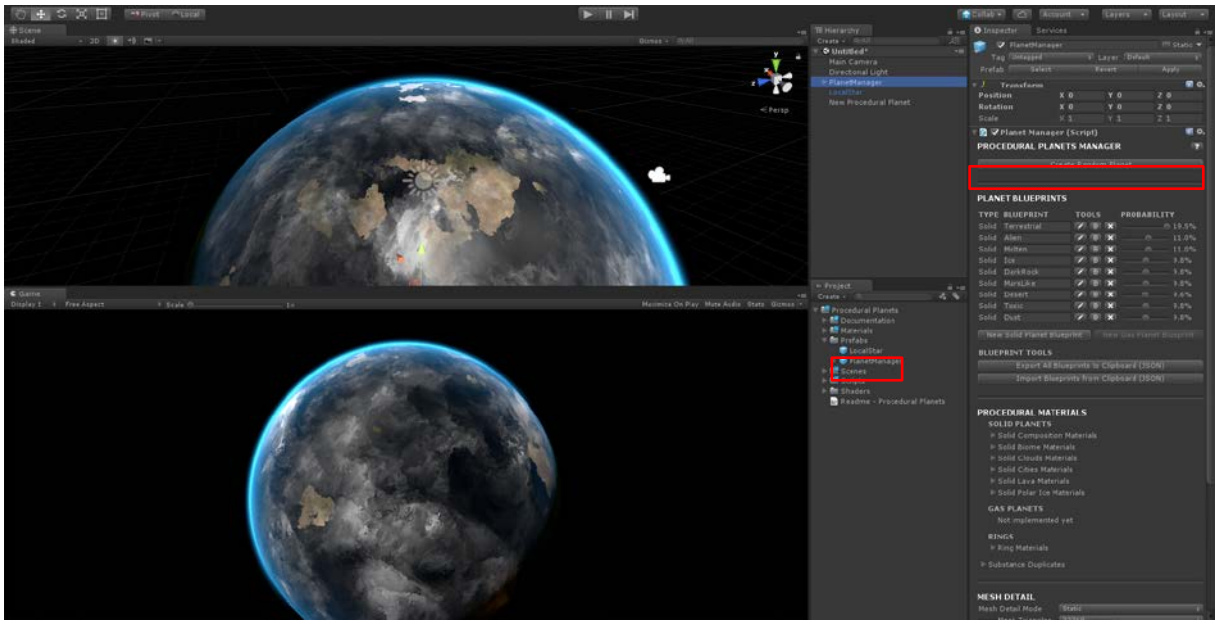
Create your first planet

Preparations

1. Create a new Unity Project
2. (Recommended) Go to **File | Build Settings | Player Settings**...
    a. Under Other Settings change Color Space to **Linear**
3. **Import the Procedural Planets asset** into Unity3D from the Asset Store
4. (Recommended) Change the **Main Camera** to **Solid Color** and set background color to **Black**

Crete PlanetManager, LocalStar and a Random Planet

5. Drag the **Procedural Planets\Prefabs\PlanetManager** prefab from Project window into Hierarchy (or scene)
6. Drag the **Procedural Planets\Prefabs\LocalStar** prefab from Project window into the Hierarchy (or scene)
7. Click the **Create Random Planet** button in the Inspector for the PlanetManager



What just happened?

PlanetManager

The PlanetManager prefab that you added in step 5 is a required component for Procedural Planets to work. The PlanetManager is a Singleton[1] class and can only exist as one instance. It also uses Unity's DontDestroyOnLoad[2] so once it is created it will persist in all scenes and survive scene switching until you manually destroy the PlanetManager.

The PlanetManager have the following main purposes:

- Keeps track of Planet Blueprints (a blueprint controls ranges of random values for planet types)
- Keeps references to Procedural Materials that are used to generate planet textures
- Keeps track of probability of blueprints being used when creating random planets
- Keeps track of Level of Detail (LOD) for planet mesh and texture details
- Has one (or more) procedurally generated spherical meshes that planets share
- Enables creation of planets from scripts using public static methods

> ***Important:*** *You must always have the PlanetManager in your scene.*

[1] https://en.wikipedia.org/wiki/Singleton_pattern
[2] https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

For more details about the PlanetManager, see: PlanetManager, page 15.

LocalStar

The LocalStar prefabs that you added in step 6 is also a required component for Procedural Planets to work. The planets are not using traditional Unity light sources for lighting. Instead, the position of a gameobject with the LocalStar component added to it acts as the source of light that lights the planet and any optional rings that the planet has.

The reason why normal light sources are not used is because the planets are generally rendered by a separate background camera and should only be lit by a star and not by things such as ship lights etc.

You can set the color, intensity and ambient intensity of a LocalStar which affects how the planet and planet rings are lit.

Important:

1. *You must always have a gameobject with the LocalStar component in your scene.*
2. *The star does not appear in the game view, it is only a logic representation of where the light should be coming from.*

Creating the random planet

When you clicked the "Create Random Planet" button in step 7, the PlanetManager creates a random planet and adds it to the scene. The planet is given a "random seed" which is a number that controls what the planet will look like.

The PlanetManager uses one of the Planet Blueprints that it has in its configuration to create the planet. The blueprint is selected randomly but using a probability weighting that you can configure in the PlanetManager.

When the PlanetManager picks a blueprint, it will use that and override the blueprint parameter of the planet that was created to ensure that the blueprint based on the probability is used. For example, if you have configured a blueprint that creates terrestrial (earth-like) planets to have 80% probability whereas desert planets may only have 20% probability, the PlanetManager will most likely create an earth-like planet.

The blueprint is a concept that's important because if planets were created truly random most planets would, simply put, look terrible and very, very few would ever look earth-like. The blueprint specifies ranges for properties when they are randomly generated so earth-like planets, for example, always has a blueish atmosphere, bluish water, white clouds and some water coverage.

To read more about Blueprints, see Blueprints page 19.

Random Planets and the “Butterfly Effect”

If you followed the steps to create a planet - select the **New Procedural Planet** gameobject in the Hierarchy.

In this case, random seed 1461277949 was used and it was forced to be a Terrestrial planet.

This random seed should always produce the same identical planet but, *and this is a very important but*, it is totally dependent on that the PlanetManager configuration is kept *totally* unchanged. This means that any change to probability sliders, adding/removal of blueprints, adding/removal of materials, etc. will result in changing the appearance of a planet totally.

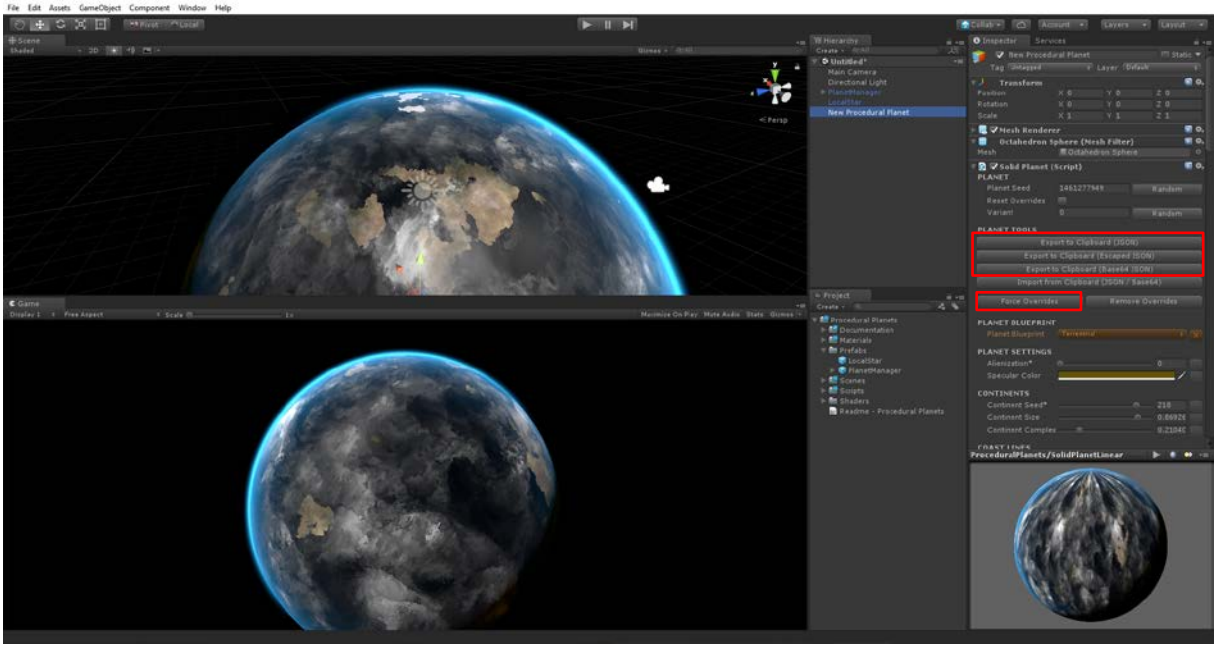
Think of it like the *Butterfly Effect*[3] . The slightest change to a parameter will affect the random engine that pushes it to select a different blueprint and totally different settings for all properties of the planet so instead of a terrestrial planet you could end up with a purple alien planet.

Realistically, and especially during early stages of development, it is very likely that your PlanetManager configuration will change. Also, when the ProceduralPlanets asset is updated it will affect the PlanetManager configuration and planets that rely solely on the seed for their appearance will change.

To ensure that planets always look the same you can either:

1. Lock the planet configuration using the “**Force Overrides**” button in the inspector
    a. Locking a planet configuration changes all properties to an “overridden” state so they no longer rely on the random seed.
2. **Export the planet configuration to a JSON** string and generate (“spawn”) the planets from a script using that JSON string.
    a. Exporting a planet to a JSON string includes all individual properties and when the string is used to create a planet it will check if the random value matches the value in the JSON string and if it does not it will override with the value from the string.

Important: *Saving a planet as a prefab without using “Force Override” will not protect the planet from changing its appearance if the PlanetManager configuration changes.*



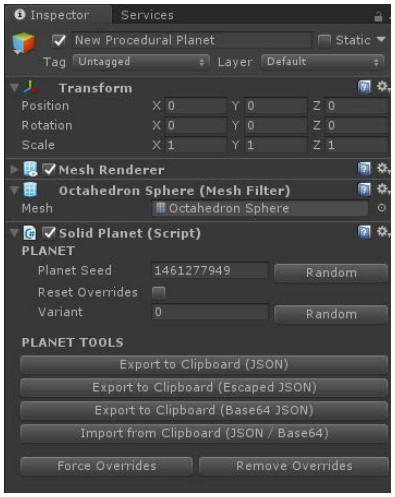
[3] https://en.wikipedia.org/wiki/Butterfly_effect

Customizing the Planet

It's likely that you will want to modify and customize a planet. Maybe you want less water, different biomes for example. When the planet is selected in the Hierarchy, you the custom editor for the planet appears in the Inspector.

Random Seed

At the top you can change the seed, also create variations of the planet. If you change the Variation value it will modify the planet somewhat but keep the general characteristics.



Planet Tools

The JSON[4] format is used which is a human readable open standard format when exporting and importing planet settings. Once exported you can use the string to create planets from scripts or you can import them manually in the Inspector for a planet.

Export Planet Settings

Use the **Export to Clipboard** buttons to export the planet settings to the clipboard. You need to do something with the information in the clipboard and you can, for example, paste it to a text file, store it in a database, or even paste it as a string in a script to recreate the planet.

- **Export to Clipboard (JSON)**
  - Easy to read for humans with indentation and line breaks. Useful when you want to see and edit the content clearly. Not suitable for pasting into scripts since the string contains quotes.
- **Export to Clipboard (Escaped JSON)**
  - Somewhat easy to read and edit for humans but has the benefit of characters like quotes being "escaped" which means the string can be used in a script.
- **Export to Clipboard (Base64 JSON)**
  - Base64[5] is not readable or editable for humans but has the benefit of being a consistent looking block of characters that is safe to use in scripts and databases since it doesn't contain any problematic characters.

Import Planet Settings

You can use the **Import from Clipboard (JSON / Base64)** button to import (which overwrites) the configuration. For this button to work you will need to have copied a valid JSON string into the clipboard and it can be in any of the formats including Base64 encoded.

You can also use the PlanetManager CreatePlanet() method from a script to create a planet with he JSON string supplied as an argument.

Locking a plant configuration

The **Force Overrides** button "locks" all the settings for a planet except the Blueprint by setting them as overridden even if they have the value provided by the random. The benefit of forcing overrides is that the settings of the PlanetManager can be changed without resulting in the planet looking differently.

The **Remove Overrides** button removes all overrides except for the Blueprint. This is a convenient way to remove any overrides and revert the planet back to its randomly generated state.

---

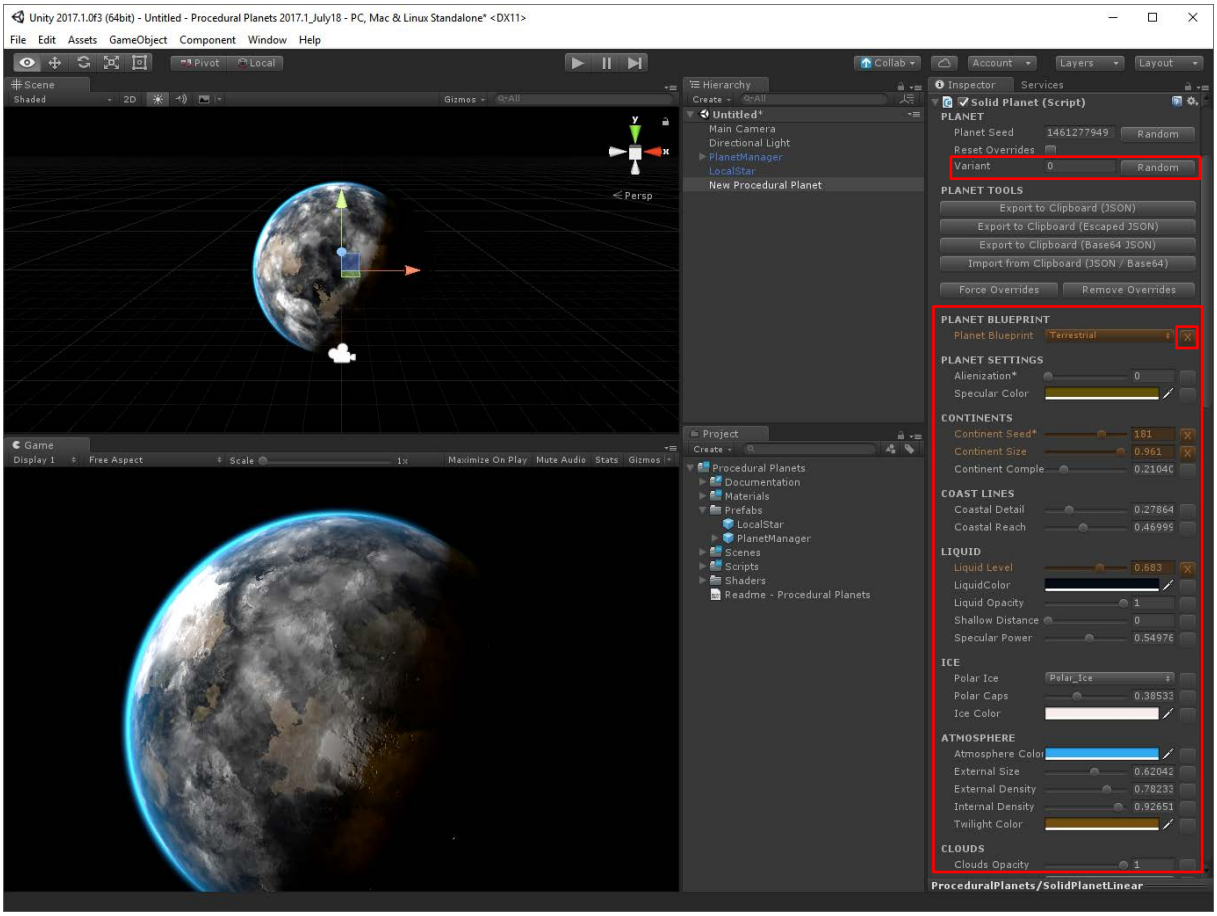
[4] https://en.wikipedia.org/wiki/JSON

[5] https://en.wikipedia.org/wiki/Base64

Planet Properties

You can modify and override all properties of a planet:

1. Select the planet in the hierarchy
2. Scroll down in the inspector for the planet and locate planet settings
3. Use sliders, color fields and dropdown dialogs to override parameters.
    a. As you change a parameter it will switch from being randomly allocated to an overridden value. This is indicated by changing to yellow color in the inspector and a yellow X button appears (which will revert to the original random value if clicked)
    b. See details about all properties and what effect they have on the planet in the next chapter



Planet Blueprint:

By default, a planet will already have the Planet Blueprint set to an overridden value. This is because the manager spawns types of planets based on probability so it needs to force a blueprint type. You can make a planet selecting a random blueprint by clicking the yellow X button next to the planet blueprint.

Variant:

You can change to a new variant of the planet by changing the variant seed. This will change random of some selected parameters to a small degree which should make a very similar planet but with different continents and feature formations.

Solid Planet Properties

Properties marked "(Shader Only)" can be changed without any performance impact and requires no texture rebuild. An asterisk (*) indicates that a lookup texture needs to be regenerated but that is a cheap operation.

Properties marked "(Rebuilds XYZ texture)" requires one or more procedural textures to be rebuilt which is expensive from a performance perspective. Changing just one single procedural texture on a single planet at lower resolutions can be done in near real-time on high end hardware.

Solid Planet Properties:

- **Planet Blueprint (Rebuilds everything)**
  - The blueprint a planet uses to get valid ranges for random properties. A blueprint does not restrict you from overriding properties to any extent.
- **Alienization (Rebuilds Biome 1 & Biome 2 textures)**
  - A hue shifter and the higher it is the more different the colors are. E.g. a forest is green but using the alienization slider you can make it orange and purple.
- **Specular Color (Shader only)**
  - The color tint of specular reflection on liquid.
- **Continent Seed (Rebuilds Composition Texture)**
  - The random seed used for continent formation. Change this if you want to have a planet with different shore line and continent formation.
- **Continent Size (Shader only)**
  - Sets the size of continents where 0 is small and 1 is large.
- **Continent Complexity (Rebuilds Composition Texture)**
  - Warps the composition map to create more complex features and coast lines.
- **Coastal Detail (Shader only)**
  - Sets the smoothness and complexity of shores and coastlines. 0 is smooth and 1 is detailed and complex.
- **Coastal Reach (Shader only)**
  - Affects the archipelago, i.e. number of little islands around coastlines.
- **Liquid Level (Shader only*)**
  - Liquid / water coverage of planet. 0 = no water, 1 = only water.
- **Liquid Color (Shader only)**
  - Color of liquid / water. Usually very dark blues look good.
- **Liquid Opacity (Shader only)**
  - Transparency of liquid / water. 0 = fully transparent, 1 = fully opaque. Tip: Liquid Opacity set to 0 can block out areas of lava.
- **(Liquid) Shallow Distance (Shader only*)**
  - Transparency crossfade region between liquid and land. 0 = sharp coastlines, 1 = long crossfade distance.
- **(Liquid) Specular Power (Shader only)**
  - Specular reflection coverage. 0 = large area, 1 = small area.
- **Polar Ice (Rebuilds Polar Ice Texture)**
  - Material used for polar caps.
- **Polar Caps (Shader only*)**
  - Size of polar caps, how far polar ice reaches from poles towards equator. 0 = no polar caps, 1 = coverage to polar circle.
- **Ice Color (Shader only)**
  - Color to replace where polar caps cover liquid.
- **Atmosphere Color (Shader only)**
  - Color of atmosphere.
- **(Atmosphere) External Size (Shader only)**
  - Size of external atmosphere, how far out it reaches from planet.
- **(Atmosphere) External Density (Shader only)**
  - Density / thickness of external atmosphere.
- **(Atmosphere) Internal Density (Shader only)**
  - Thickness of atmosphere that covers the planet, i.e. rim lighting.
- **Twilight Color (Shader only)**
  - Color tint of region between night and day to simulate dusk/dawn twilight zones.

- **Clouds Opacity (Shader only)**
    - Transparency of clouds. 0 = invisible (and disabled), 1 = full opacity. Note: Cloud Opacity is applied after Cloud Layers and Cloud Coverage so even with opacity set to 1 clouds can be invisible if coverage or layers are set low.
- **Clouds Seed (Rebuilds Clouds Texture)**
    - Random seed used for cloud noise generators.
- **Clouds Color (Shader only)**
    - Color of clouds.
- **Clouds Roughness (Rebuilds Clouds Texture)**
    - Normal/bump map of clouds.
- **Clouds Coverage (Rebuilds Clouds Texture)**
    - Opacity of Clouds after Clouds Layer 1 + 2 + 3 have been combined.
- **Clouds Layer 1-3 (Rebuilds Clouds Texture)**
    - Individual layers of different styles of clouds. Layer 1-3 are blended together and is then affected by Clouds Coverage and Clouds Opacity.
- **Clouds Sharpness (Rebuilds Clouds Texture)**
    - Sharpness, contrast, detail of clouds.
- **Clouds Tiling (Shader only)**
    - How many times cloud texture is tiled around the planet. Tip: Use 2-6 times for planets fully visible and increase to higher values if viewed from close distance to avoid repetition.
- **Clouds Speed (Shader only)**
    - How fast the clouds rotate around the planet.
- **Clouds Height (Shader only)**
    - How much the cloud shadows should be offset from clouds.
- **Clouds Shadow (Shader only)**
    - Strength of shadow (shadow distance is affected by cloud height).
- **Lava Amount (Shader only)**
    - How much molten lava should exists. For small cracks, use very low numbers, e.g. 0.001 – 0.01. If you want to break up the pattern, consider using a fully transparent Liquid with no Shallow distance to block areas where lava would otherwise be.
- **Lava (Rebuilds Lava Texture)**
    - Texture to be used for lava.
- **Lava Complexity (Rebuilds Composition Texture)**
    - Warping/complexity modifier for lava. The composition texture contains a channel for lava coverage.
- **Lava Frequency (Shader only)**
    - Affects tiling of the lava texture resulting in 0 = large cracks and 1 = small cracks.
- **Lava Detail (Shader only)**
    - Affects straightness of crack edges 0 = straight, 1 = warped.
- **Lava Reach (Shader only)**
    - Affects range of edges with hardened rock islands.
- **(Lava) Color Variation (Rebuilds Lava Texture)**
    - Hue color shifting of lava texture. 0.5 = no change, 0 = shift hue left and 1 = shift hue right.
- **(Lava) Flow Speed (Shader only)**
    - How fast the lava animation flows.
- **(Lava) Glow Amount (Shader only)**
    - Glowing effect around lava. 0 = no glow, 1 = max glow.
- **(Lava) Glow Color (Shader only)**
    - Color of lava glow effect.

- **Surface Roughness (Shader only)**
    - Normal/bump map strength of land areas. Also affects clouds.
- **Surface Tiling (Shader only)**
    - Tiling amount of surface biome textures. This can often be quite high, 10-20, especially if planets also have liquid and cloud coverage.
- **Composition (Rebuilds Composition Texture)**
    - Material to be used for planet composition, i.e. continents and biomes are mixed together.
- **Composition Seed (Rebuilds Composition Texture)**
    - Seed used for composition texture that affects composition map for biome blending.
- **Composition Tiling (Shader only)**
    - Tiling amount of composition texture. For planets viewed from a distance, use a low value, 2-5 and for close distance where you see parts of planets you can have higher tiling values.
- **Composition Chaos (Rebuilds Composition Texture)**
    - Chaos / warp effect of composition texture. Affects how biome blending edges appear.
- **Composition Balance (Rebuilds Composition Texture)**
    - Balance between Biome 1 and Biome 2 surface texture where 0.5 is a 50/50 mix, 0 = only Biome 1 and 2 = only Biome 2.
- **Composition Contrast (Rebuilds Composition Texture)**
    - Contrast / crossfading between Biome 1 and Biome 2 surface materials. 0 = sharp / 1 = heavily crossfaded transition.
- **Biome 1-2 Seed (Rebuilds Biome 1-2 Textures)**
    - Random seed used for texture generation.
- **Biome 1-2 Type (Rebuilds Biome 1-2 Textures)**
    - Material used for biome 1-2.
- **Biome 1-2 Chaos (Rebuilds Biome 1-2 Textures)**
    - Chaos / warp effect of biome texture.
- **Biome 1-2 Balance (Rebuilds Biome 1-2 Textures)**
    - Biome textures have two main textures internally – the balance blends between the two.
- **Biome 1-2 Contrast (Rebuilds Biome 1-2 Textures)**
    - Biome textures have two main textures internally, contrast dictates fading between the two textures where 0 = sharp and 1 = heavily crossfaded.
- **Biome 1-2 Color Variation (Rebuilds Biome 1-2 Textures)**
    - Slight hue shifting of biome texture where 0.5 is original color and <0.5 shifts color hue left and >0.5 shifts color hue right.
- **Biome 1-2 Saturation (Rebuilds Biome 1-2 Textures)**
    - Saturation of biome texture where 0.5 is original saturation value and <0.5 desaturates texture and >0.5 increases saturation.
- **Biome 1-2 Brightness (Rebuilds Biome 1-2 Textures)**
    - Brightness of biome texture where 0.5 is original brightness and <0.5 darker and >0.5 brighter.
- **Biome 1-2 Small/Medium/Large Craters (Rebuilds Biome 1-2 Textures)**
    - Strength of different sized craters on the biome texture.
- **Biome 1-2 Crater Erosion (Rebuilds Biome 1-2 Textures)**
    - Erosion strength of craters where 0 = no erosion and 1 = very eroded.
- **Biome 1-2 Craters Diffuse (Rebuilds Biome 1-2 Textures)**
    - Color variation of craters where 0 = no change in color.
- **Biome 1-2 Canyons Diffuse (Rebuilds Biome 1-2 Textures)**
    - Color variation of canyons where 0 = no change in color.
- **Biome 1-2 Surface Bump (Rebuilds Biome 1-2 Textures)**
    - Normal / bump map strength of biome / surface textures.
- **Biome 1-2 Craters Bump (Rebuilds Biome 1-2 Textures)**
    - Normal / bump map strength of craters.
- **Biome 1-2 Canyons Bump (Rebuilds Biome 1-2 Textures)**
    - Normal / bump map strength of canyons.
- **Cities (Rebuilds City Textures)**
    - Material used for night city lights.
- **(Cities) Random Seed (Rebuilds Cities Textures)**
    - Random seed for city texture generator.
- **(Cities) Population (Rebuilds Cities Texture)**

    - Number of city lights on the night side of a planet.
- **(Cities) Advancement (Rebuilds Cities Texture)**
    - Strength of “advanced civilizations” large cities.
- **(Cities) Glow (Rebuilds Cities Textures)**
    - Glow strength of city lights.
- **(Cities) Tiling (Shader only)**
    - Amount of tiling of city texture. Usually lower values 2-6.
- **Night Light Color (Shader only)**
    - Color of night city lights.

Gas Planet Properties

Properties marked "(Shader Only)" can be changed without any performance impact and requires no texture rebuild. An asterisk (*) indicates that a lookup texture needs to be regenerated but that is a cheap operation.

Properties marked "(Rebuilds XYZ texture)" requires one or more procedural textures to be rebuilt which is expensive from a performance perspective. Changing just one single procedural texture on a single planet at lower resolutions can be done in near real-time on high end hardware.

Gas Planet Properties:

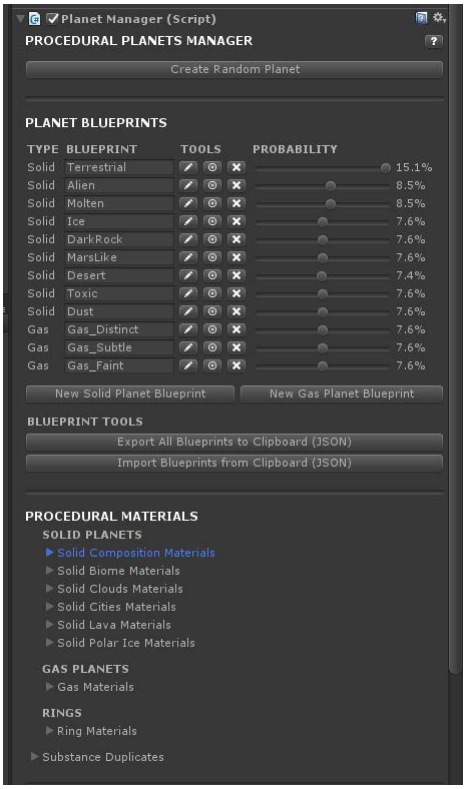
- **Planet Blueprint (Rebuilds everything)**
  - The blueprint a planet uses to get valid ranges for random properties. A blueprint does not restrict you from overriding properties to any extent.
- **Gas (Rebuilds Gas Material)**
  - The procedural material used to create textures.
- **Gas Seed (Rebuilds everything)**
  - Random seed for the gas texture.
- **Horizontal Tiling (Shader only)**
  - Horizontal tiling of the texture on the gas planet.
- **Vertical Tiling (Shader only)**
  - Vertical tiling of the texture on the gas planet.
- **Turbulence Seed (Rebuilds everything except Palette Texture)**
  - Random seed for turbulence pattern.
- **Turbulence (Rebuilds everything except Palette Texture)**
  - Amount of turbulence (0 = none, 1.0 = max)
- **Turbulence Scale (Rebuilds everything except Palette Texture)**
  - Scale of turbulence (0 = largest, 1.0 = smallest)
- **Turbulence Disorder (Rebuilds everything except Palette Texture)**
  - Disorder of turbulence – animates the turbulence with seamless loop if cycling from 0.0 – 1.0 and then restarting at 0.0.
- **Separation (Rebuilds everything except Palette Texture)**
  - Separation of colors/bands in the gas planet.
- **Smoothness (Rebuilds everything except Palette Texture)**
  - Smoothness of lines/bands around planet, applies horizontal blur where 0.0 is no horizontal blur and 1.0 is max horizontal blur.
- **Blurriness (Rebuilds everything except Palette Texture)**
  - Blurs planet textures. Be careful, degrades quality with higher amount of blur.
- **Palette (Rebuilds Palette texture)**
  - Palette index 1-8, different gradient palettes.
- **Detail (Rebuilds Palette texture)**
  - Adds noise to the palette for a more detailed gradient.
- **Detail Offset (Rebuilds Palette texture)**
  - Changes the detail noise.
- **Contrast (Rebuilds Palette texture)**
  - Alters contrast of the palette – be careful, may degrade quality. If you want to have less contrast in the gas planet it is strongly recommended to use the faintness and faintnessColor properties instead!
- **Hue (Rebuilds Palette texture)**
  - Color hue shifting of palette. 0.5 = original hue and 0.0 – 1.0 rotates through the entire hue spectrum.
- **Saturation (Rebuilds Palette texture)**
  - Color saturation strength 0.0 = grayscale, 1.0 oversaturated.
- **Brightness (Rebuilds Palette texture)**
  - Brightness/lightness of palette texture. Be careful, may degrade quality of planet as number of colors are reduced.
- **Banding (Shader only)**
  - Shader-based feature similar to the separation property – ads banding to the planet.
- **Solidness (Shader only)**
  - Affects transparency of the atmospheric edge of the planet – 0.0 very low density and transparent edges, 1.0 = sharp edges with little atmospherical effect.
- **Faintness (Shader only)**

    - Overlay amount of faintnessColor – use this to dim a planet to faintnessColor to reduce contrast and get very subtle planets.
- **Faintness Color (Shader only)**
    - Color to dim to with the faintness property.
- **Roughness (Rebuilds everything except Palette Texture)**
    - Normal mapping strength 0.0 = no normal/bump map, 1.0 strong normal/bump mapping.
- **Twilight Color (Shader only)**
    - Color for dusk/dawn transition between day and night of planet. To reduce the effect, use darker colors. Use black for no twilight effect.
- **Storm Mask Index (Rebuilds storm mask lookup texture)**
    - Index of storm mask – this cycles the position of the storm around the planet. The index allows the storm to be drawn on only one tiled square. E.g. if planet tiles 4 times horizontally and 2 times vertically the storm index will have 8 slots and position the storm somewhere in the 4x2 tiled planet.
- **Storm Squash (Rebuilds everything except Palette Texture)**
    - Amount storm should by squashed vertically – higher values will flatten the storm making it more oval.
- **Storm Color (Shader only)**
    - Color that storm region should interpolate towards.
- **Storm Tint (Shader only)**
    - Amount of tinting towards stormColor (0.0 = no tinting, 1.0 = full tinting to stormColor)
- **Storm Scale (Rebuilds everything except Palette Texture)**
    - Size of the storm (0.0 = smallest, 1.0 = largest) – be aware, values between 0.5 and 1.0, especially with large stormNoise values may make the storm masking seam visible somewhat.
- **Storm Noise (Rebuilds everything except Palette Texture)**
    - Noise of storm – 0.0 = round, 1.0 = warped and unevenly distributed. Be aware of values between 0.5 and 1.0 as seam from masking could become somewhat visible.
- **Atmosphere Color (Shader only)**
    - Color of internal planet rim lighting. Use darker colors for less prominent atmosphere.
- **Atmosphere Falloff (Shader only)**
    - Falloff amount of rim lighting atmosphere. 0.0 = little falloff (more atmosphere) 1.0 = strong falloff (less atmosphere)

PlanetManager

Manager is an essential key component that must persistently exist in all scenes for the planets to work.

What does the PlanetManager component do?



The PlanetManager maintains a repository of Planet Blueprints that allow you to create random planets of specific types, e.g. terrestrial (earthlike) planets, ice planets, dusty planets, etc. The PlanetManager also keeps track of the probability of a planet type is to be randomly created so that you can make derelict rocky planets more common than populated water worlds.

- Keeps track of Planet Blueprints (a blueprint controls ranges of random values for planet types)
- Keeps references to Procedural Materials that are used to generate planet textures
- Keeps track of probability of blueprints being used when creating random planets
- Keeps track of Level of Detail (LOD) for planet mesh and texture details
- Has one (or more) procedurally generated spherical meshes that planets share
- Enables creation of planets from scripts using public static method: CreatePlanet()

The PlanetManager maintains a repository of Procedural Material arrays. These arrays can be filtered by blueprints so dusty planets only allow dusty surface materials. This also allows for more procedural materials for planet composition, biomes, polar ice, clouds, and lava to be added at a later stage.

Procedural Planets Manager Section

- The **Create Random Planet** button creates a new random planet using a random blueprint.

Planet Blueprints Section

- The **Pen Icon** next to a blueprint edits the blueprint – basically it is just a shortcut to select the blueprint in the hierarchy
- The **(+) Icon** next to a blueprint creates a planet using that specific blueprint
- The **X Icon** next to a blueprint deletes that blueprint
- The **Slider** next to a blueprint controls the probability of the blueprint being chosen when creating a random planet
- The **New Solid Planet Blueprint** and **New Gas Planet Blueprint** buttons creates new blueprints and adds them to the list
- The **Export All Blueprints to Clipboard (JSON)** button can be used to back up or copy all the blueprint configurations to clipboard so you can paste it into a text file or import it from the clipboard again to recreate all the blueprints. This is useful if you need to create a new PlanetManager component or copy settings from one project to another.
- The **Import Blueprints from Clipboard (JSON)** button allows you to import blueprints but be aware that this overwrites any existing blueprints. You must have a valid JSON string of blueprints copied to the clipboard first which can be obtained by using the export button above.

Procedural Materials Section

There are three sections for arrays of procedural materials for solid planets, gas planets and rings. The arrays contain procedural materials that are used when planets are created. The arrays must contain unique materials and the PlanetManager prefab has already been configured with the most appropriate allocation of materials.

The “Substance Duplicates” array is needed to reference duplicate materials that are used if multiple planets are created in a scene since all planets must use unique instances of Procedural Materials so they don’t overwrite eachother’s textures.

> ***Important:*** *It is recommended to use the PlanetManager prefab which comes configured with the material arrays already populated.*

Level of Detail Section

See the Level Of Detail (LOD) section on page 17 of this documentation for Level of Detail (LOD).

Scripting Reference

Please refer to the online documentation for full details of scripting, including example code:
https://www.imphenzia.com/documentation/procedural-planets

CreatePlanet() is a public method that you can access through scripts to create planets during runtime.

You can call it like this to create a random planet at a Vector3 position in the current scene:

```
PlanetManager.CreatePlanet(<Vector3 position>);
```

You can also specify a specific planet seed, like this:

```
PlanetManager.CreatePlanet(<Vector3 position>, <integer seed>);
```

If you want to override and force a specific blueprint, e.g. a terrestrial planet, you can do this:

```
PlanetManger.CreatePlanet(<Vector3 position>, <integer seed>, <string blueprint name>);
```

Finally, you can also create a very specific planet using a JSON-string, like this:

```
PlanetManager.CreatePlanet(<Vector3 position>, <string JSON settings>);
```

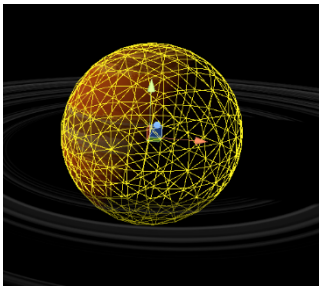
Level Of Detail (LOD)

The Level of Detail section of the PlanetManager allows you to choose between static and dynamic levels of details for meshes and texture resolutions.

Mesh Detail

Choose Mesh Detail Mode to be **Static** or **LOD.**

- **Static** mode uses one single procedurally generated mesh of a set number of triangles that you can configure. The single mesh is used as a shared mesh by all planets and atmospheres so it only exists once in memory regardless of how many planets you use. You can choose 7 different levels ranging from 8 to 32768 triangles. It is recommended to use at 2048 triangles or more if the planets are occupying a larger part of the screen.



- **LOD** mode uses 2-5 levels of detail. You can select how many levels using the Levels slider. All the meshes are created by the PlanetManager and exist as one instance at all times. Planets select the most appropriate mesh based on how large the planet is on screen as seen by the LOD camera (defaults to Main Camera but can be changed in the Inspector just above the Mesh Detail section.

  You can use the sliders or drag the perimeter lines between the blue sections to set the size of a planet used to transition between different levels of detail.

  You can click the **Show LOD Editor** toggle to show the mesh in the scene view.

Texture Detail

Choose Texture Detail Mode to be **Static, Static_Separate**, **Progressive, Progressive_Separate**, **LOD** or **LOD_Separate**.

- **Static** mode forces all textures used to create planets to a fixed resolution.
- **Static_Separate** mode allows you to set fixed resolutions for different texture types. For example, the cloud and composition textures are of greater importance to have high resolution compared to the biome textures of a solid planet because the biome textures are more tiled and look good even at lower resolutions.
- **Progressive** mode allows you to use the same resolution for all textures but the planets will progressively improve the quality of the textures based 2-5 times depending on how you configure the Steps slider. The planet will first be created with Step 0 texture, usually this is kept very low to have the planet appear fast, and then the planet will gradually improve the quality of the texture until it has reached the final step.
- **Progressive_Separate** mode is similar to Progressive but it allows you to set different resolutions for different types of textures.
- **LOD** mode uses 2-5 levels of detail. You can select how many levels using the Levels slider. Planets select the most appropriate texture detail based on how large the planet is on screen as seen by the LOD camera (defaults to Main Camera but can be changed in the Inspector just above the Mesh Detail section. You can use the sliders or drag the perimeter lines between the blue sections to set the size of a planet used to transition between different levels of detail.
- **LOD_Separate** mode is similar to LOD mode but it allows you to set different resolutions for different types of textures. You can use the sliders or drag the perimeter lines between the blue sections to set the size of a planet used to transition between different levels of detail.

Which Texture Detail mode should I choose?

Static mode is recommended if you want to keep it simple and if you are happy with the way the planets look, how much memory they occupy and how long they take to create.

Static_Separate mode is recommended if you want to keep it simple but you need to preserve memory or make the planets generate faster. The textures you can consider reducing first in quality are Biomes, Lava and PolarIce since those are tiled much more than the other textures.

Progressive mode is recommended if you need planets to appear very fast on screen and you are happy with the planets gradually improving their appearance once they have been generated. It is not recommended to use progressive mode if you plan to animate planet properties.

Progressive_Separate mode is recommended if you want to progressively improve the planets like normal progressive mode but need to preserve memory or have the planets generating faster. It is not recommended to use progressive mode if you plan to animate planet properties. The textures you can consider reducing first in quality are Biomes, Lava and PolarIce since those are tiled much more than the other textures.

LOD mode is recommended if you want planets to have lower resolution when they occupy a small portion of the screen and gradually improve their appearance as they occupy the screen more.

LOD_separate mode is recommended if you want planets to have lower resolution when they occupy a small portion of the screen and gradually improve their appearance as they occupy the screen more and if you need to preserve memory or have planets generating faster. The textures you can consider reducing first in quality are Biomes, Lava and PolarIce since those are tiled much more than the other textures.

Blueprints

Planet Blueprints are templates with specific random ranges for parameters.

Why are planet blueprints needed?

Totally random planets tend to not look very good unless you are very lucky. Also, you may want to be able to specify a particular type of planet to be created, e.g. a terrestrial (earth-like) planet, ice planet, molten planet, or dusty planet.

How do you create blueprints?

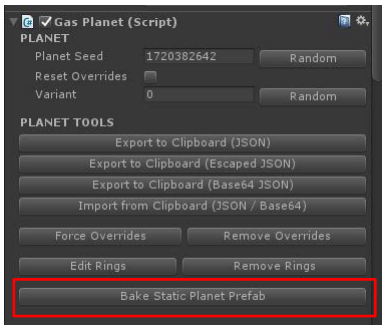
1. Select the PlanetManager component in the inspector and click, for example, “Create Solid Planet Blueprint”.
2. Set the Probability value in the Manager inspector – this dictates how likely this blueprint is to be used when a random planet is created.
3. Highlight the newly created child object in the hierarchy (or click the pen icon in the PlanetManager inspector)
4. Rename the planet blueprint to a new unique descriptive name, e.g. Desert
5. Configure sliders in the inspector to filter use of only certain materials, minimum/maximum ranges of properties (e.g. if you want no molten lava on the planet you set the both Min and Max values of the slider to 0 and if always want at least half the planet to be covered by water you set Liquid Level Min to 0.5 and Max value to 1.0)
6. You can optionally also create a child ring blueprint and set the probability of a planet created sing this blueprint having rings.

Important: *Creating blueprints will require you to have a fair understanding of all the properties so you may want to play around with a planet in the scene and pull all sliders to see what effect they have.*

Baking Planets

If you need to create a planet that generates instantly with a high level of detail, or if you simply prefer to have a planet as a static prefab object, you can “bake” a planet.

Select a procedural planet in the hierarchy and click the **“Bake Static Planet Prefab”** button in the inspector for the planet.

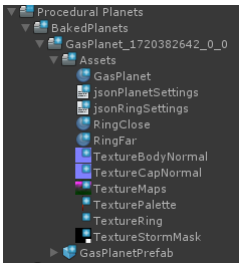


> ***Important:*** *The current procedurally generated textures are used to create the static textures so it is important that you have allowed the planet to fully generate its textures before baking a planet. If you want to ensure the highest quality possible, set the Texture Level of Detail in the PlanetManager to “Static” mode and the highest resolution possible.*

Unity will take a few seconds to render the textures and create the assets. The assets will be placed in a new folder in the Project tab:
Procedural Planets\Baked Planets\<PlanetType>_<Seed>_<Variation>

If you want to use the baked planet, drag the SolidPlanetPrefab/GasPlanetPrefab from the folder into the scene. All assets required by the planets are stored in the Assets subfolder and it includes materials and textures. Text JSON files are also included if you want to recreate the procedural version of the planet in the future or from script.

> ***Important:***
>
> 1. *Baked planets can no longer be changed or animated except for changing material properties that are exposed via the shader.*
> 2. *Baked planets are big! Textures will occupy a lot of disk space and significantly increase the size of a game distributable. Use texture resolutions that balance size and acceptable visual appearance.*



Animating Properties

You can animate properties during run-time from scripting.

The Planet method AnimateFloat animates a float after a delay from a source value to a destination value over the time of duration.

> ***Important:***
>
> 1. *Animating shader features is fast - animating procedural materials is slow!*
> 2. *It is not recommended to have PlanetManager Texture Detail set to any of the Progressive modes if you plan to animate non-shader based properties.*

Refer to the **ProceduralPlanets_Properties.pdf** documentation for a full reference of property names. The reference also states if a property is a shader feature (fast to animate) or which procedural materials need to be rebuilt (slow).

Please refer to the online documentation for full scripting documentation:
https://www.imphenzia.com/documentation/procedural-planets

Color Space

Linear or gamma workflow?

- **Designed for Linear** color space
- **Works in Gamma** color space (but looks different)
- More Information:
    - https://docs.unity3d.com/Manual/LinearRendering-LinearOrGammaWorkflow.html

Scripting Reference

- **Programming Language**
    - All scripts are written in c#
- **Namespace**
    - To avoid name conflicts all scripts use the namespace “ProceduralPlanets”
    - Use “using ProceduralPlanets;” directive in scripts to access the namespace.
- **Persistent PlanetManager Component**
    - A persistent Game Object with the “PlanetManager” component must exist in each Unity scene that has procedural planets.

For complete scripting documentation visit:

- https://www.imphenzia.com/documentation/procedural-planets

Known Issues

Very important

Changing blueprint order, blueprint configuration, adding / removing blueprints, and changing procedural material arrays will change appearance of planets even if the same seed is used. This is the nature of randomness. Planets exported to JSON-strings, however, include all values and can force property override so planets look the same to some extent.

Good to know

- **MSAA anti-aliasing may create artifacts** around the outer atmosphere – if this becomes a problem it is recommended to consider using Post-Processing Anti-Aliasing instead with the Unity Post Processing Stack to avoid the artifacts.
- **Can't use Unity Reset button** (found under the cog wheel icon) in inspector to reset a planet.
- **Editing prefabs of planets** that are not instantiated is not supported.
- **Not all Unity supported platforms are supported**, e.g. WebGL due to limited substance support in Unity.
- **Linear color space looks different from Gamma color space** (unavoidable – choose a color space to your liking)
- **Planetary Rings cannot be rotated around any axis yet** – this is because they need to auto align with the camera for transparency sorting purposes.
- **Unity 2018.x not supported yet** – This is due to the pending production release of Allegorithmic's Substance asset which will replace the Unity built in Procedural Materials. The Substance asset is still in beta and does not have the necessary API functionality. As soon as the Substance asset is released with the necessary feature support the Procedural Planet asset will be updated and released with a 2018 version.

Warnings in 2017.3+:

Some warnings will be displayed in Unity 2017.3 and newer because "Procedural Materials" will be deprecated. The procedural materials are based on technology by Allegorithmic and for Unity 2018.x Allegorithmic will release their own plugin instead to allow more rapid implementation of features compared to when it is integrated in Unity as it is today. Although this will create a dependency on another component it may come with the benefit of allowing higher resolution textures, GPU processing for textures, like in other game engines that use Allegorithmic's own plugin. I will actively be working to get Unity 2018 to work with this asset as soon as possible.

Support and Bug Reporting

Contact **support@imphenzia.com** for support.

Unity3D Forum Post: **https://forum.unity.com/threads/released-procedural-planets.503854/**

Bugs Reports - you can register and follow progress on bugs here: **https://www.imphenzia.com/mantisbt**

1. Please set debug level to "Detailed" in the bottom of the inspector of the PlanetManager manager component.
2. Reproduce the problem (include description in the bug report if possible)
3. Click on dropdown of the console and select Open Editor Log
4. Include the content of the editor log in the bug report

Supported Platforms

Unity Versions

- **Unity 5.6.x**
  - Fully supported
- **Unity 2017.x**
  - Fully supported
    - 2017.3 and later will produce warnings as Unity is replacing Procedural Materials with a new plugin.
- **Unity 2018.x**
  - Not yet supported
    - Unity has deprecated the support for native Procedural Materials but they are replacing it with a new independent plugin developed by Allegorithmic (the company behind the Substance technology used for procedural materials). Unfortunately the plugin released by Allegorithmic is still in beta and lacks some API features that are essential for Procedural Planets to work. The status of the plugin is being closely monitored and as soon as it is available for production and features the necessary API functions Procedural Planets asset will be updated and released for a 2018 version.

Build Platforms

Supported

- **Windows**
- **Android**
  - Linear Color Space: Requires Android 4.3+ with OpenGL ES3+ or Vulkan
  - Gamma Color Space: No Special Requirements
- **Mac**
  - Works with limitations:
    - Requires Metal API (not OpenGLCore)

Not tested (but should work)

- **Linux**
- **Xbox One**
- **PS4**
- **iOS**
  - Linear Color Space: Requires Metal graphics API with Open GL ES3+
  - Gamma Color Space: No Special Requirements

Not supported

- WebGL ,consoles, WSA, Tizen - dynamic procedural materials for these platforms are not supported by Unity
- Consider using the “Bake Planet” feature to create static planets for these platforms.

Frequently Asked Questions

How do I create a random planet?

- **In the inspector**
  - Highlight the Manager in the hierarchy, and click the “Create Random Planet” button
- **In scripts (using ProceduralPlanets namespace)**

```
ProceduralPlanets.PlanetManager.CreatePlanet(<Vector3 position>);
```

How do I create a planet of a specific type/blueprint

- **In the inspector**
  - Highlight the Manager in the hierarchy, and click the (+) button for a specific planet blueprint.
- Or
  - Highlight a Planet Blueprint under the Manager in the Hierarchy and click button “**Create Planet**”
- **In scripts (using ProceduralPlanets namespace):**

```
ProceduralPlanets.PlanetManager.CreatePlanet(<Vector3 position>, <integer seed>, <blueprint name>);
```

  - Tip: You can use -1 as seed for a random seed.

How do I create a specific (exported) planet in a script?

```
ProceduralPlanets.PlanetManager.CreatePlanet(<Vector3 position>, <JSON string>);
```

  - If you customize a planet in the inspector and override a number of parameters you can click on the “Export to JSON” buttons to export that exact planet configuration.

  - The exported string can be used in place of <JSON string> in the line of code example above. It’s best to use either the “Base64” or “Escaped” exported JSON string formats if you plan to instantiate them via script (or via text files or web URLs) because those formats to not contain characters such as backslash and quotation marks that would break the string handling in scripts.

Example (Base64 Encoded JSON):

```
_planet = ProceduralPlanets.PlanetManager.CreatePlanet(Vector3.zero,
"eyJjYXRlZ29yeSI6InBsYW5ldCIsInR5cGUiOiJTb2xpZFBsYW5ldCIsInZlcnNpb24iOiIwLjE…”);
```

***Important**: The string is in reality 4000+ characters long since it contains all property values including those that are not overridden if random state changes in the configuration.*

How do I override a property from a script?

- Get the component of the Planet and use the OverridePropertyFloat() method to override a property with a new float. The properties use “camelCase” with a small initial letter and capitalized letters for following words without spaces.
- Example:

```
// You need to have a reference to the Planet component:
ProcedrualPlanets.Planet _planet;
// You can get the component when a planet is created (or use GetComponent at a later stage)
_planet = ProceduralPlanets.PlanetManager.CreatePlanet(Vector3.zero, -1, “Terrestrial”);
_planet.OverridePropertyFloat(”liquidLevel”, 0.5f);
_planet.OverridePropertyFloat(”cloudOpacity”, 0f);
// You can also get a valueof a property
_planet.GetPropertyFloat(“liquidLevel”);
_planet.OverridePropertyFloat(”liquidLevel”, _planet.GetPropertyFloat(“liquidLevel”) + 0.1f);
```

See ProceduralPlanets_Properties.pdf for all parameter camelCase names and details.