EXTENDED IMPORT SETTINGS: MODEL
v1.0

Guide

nurzzz27@gmail.com

CONTENTS

DESCRIPTION

Extended Import Settings: Model is a powerful tool for Unity developers that simplifies working with 3D models, allowing modifications directly during import without breaking the connection to the original file. This asset extends the standard import settings and includes ready-made modules for modifying meshes.

Key Features:

- Modify meshes without detaching from the original file — convenient editing without creating duplicates.
- Flexible module system — enable or disable specific modifications for each model.
- Easy to extend — add your own modules with minimal effort.
- Automated mesh processing — configure parameters directly in Unity.

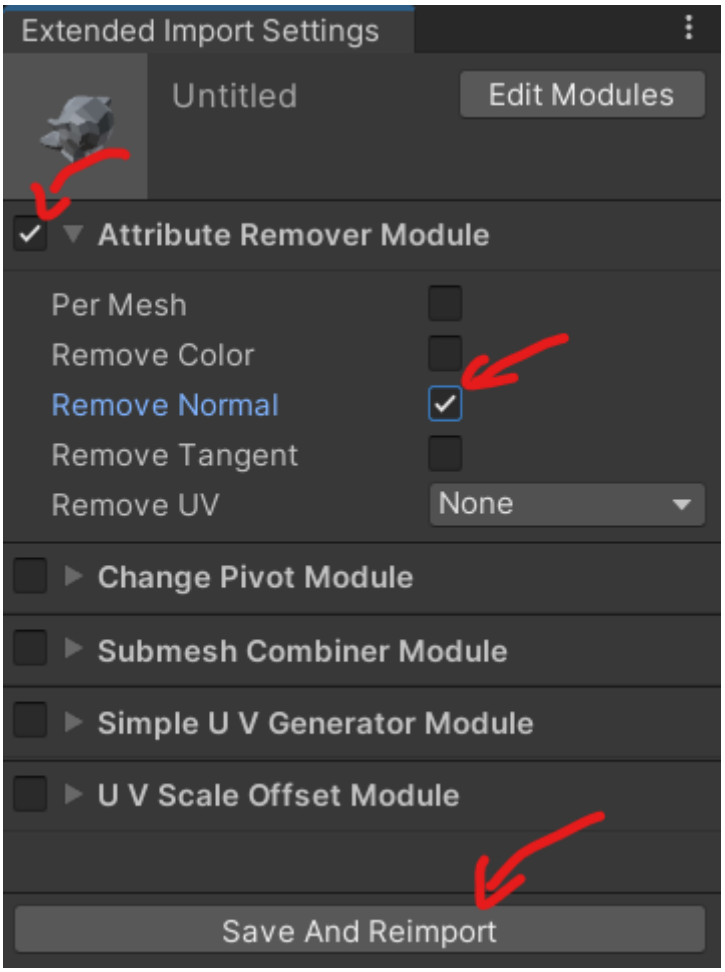
Included Modules:

- **Attribute Remover** — Removes color, normals, tangents, and UV channels.
- **Change Pivot** — Adjusts the mesh's pivot point, including offset and rotation.
- **Simple UV Generator** — Automatically generates UV maps using different projections (flat, spherical, cubic).
- **Submesh Combiner** — Merges all submeshes into one.
- **UV Scale & Offset** — Scales and offsets UV maps.

This asset is perfect for optimizing the workflow with models, minimizing manual adjustments, and increasing import flexibility.

HOW TO USE

1. Add the asset folder to your Unity project.
2. Install the Newtonsoft Json package if it's missing.
3. Open **Tools/Extended Import Settings** in Unity.
4. Select the required settings and click **"Save And Reimport"** at the bottom.



After this, the settings will be automatically applied to your model. The changes are saved in the model's meta file as a JSON string. Then, post-processing uses this string to apply changes to the model itself, ensuring the file's integrity without creating duplicate meshes.
To revert changes, simply disable the module or turn off the specific setting in **Tools/Extended Import Settings**.

MODULE DESCRIPTION

Attribute Remover (Removes attributes)
Allows removal of:

- Colors (RemoveColor)
- Normals (RemoveNormal)
- Tangents (RemoveTangent)
- Specific UV channels (RemoveUV)

Change Pivot (Adjusts the pivot point)
Allows:

- Moving the mesh to the origin (PositionToZero)
- Offsetting the pivot point (PositionOffset)
- Resetting rotation (RotationToZero)
- Adding an additional rotation angle (RotationOffset)

Submesh Combiner (Merges submeshes)
When enabled (Combine), merges all submeshes into one, simplifying rendering.

Simple UV Generator (Generates UV maps)
Automatically creates a UV map based on the chosen projection type:

- Flat (XY, XZ, ZY)
- Spherical
- Cubic

UV Scale & Offset (Scales and offsets UV maps)
Allows modification of UV coordinates:

- Scale (Scale)
- Offset (Offset)
- Selected UV channel (UVChannel)

FUNCTIONALITY EXTENSION

You can create your own modules and add new model processing parameters. A class consists of:

- Fields with parameters.
- A post-processing method responsible for applying changes to the model based on the module's data.
- A method for a custom inspector (optional).

Module

Here's a simple module for rounding coordinates:

1. Create a new file **RoundTransformModule.cs** and declare a class RoundTransformModule inherited from **ExtendedImportModule**.
2. Add the **[Serializable]** attribute before declaring the class.
3. Add **public** variables to control rounding for position, rotation, and scale, along with an accuracy parameter.

```
using ExtendedImportSettings;
using System;

[Serializable]
public class RoundTransformModule : ExtendedImportModule
{
   public bool RoundPosition;
   public bool RoundRotation;
   public bool RoundScale;
   public int Accuracy = 10000;
}
```

The module is almost ready. These are the parameters you will see in the settings window. This class uses Unity's standard serialization. Therefore, there are a few important points:

- All fields that should appear in the settings must be **public**.
- The **[Serializable]** attribute must be present before the class declaration.

Post-processing

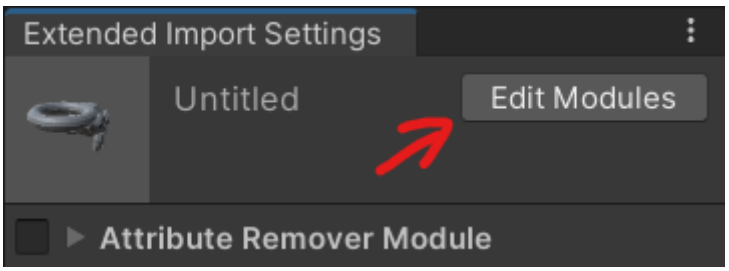
Now, you need to write the post-processing method that will process the data according to your settings:

1. In the same class, **RoundTransformModule**, override the OnPostprocessModel method.
2. Next, all that remains is to implement the logic for rounding coordinates using the module parameters.

```
public override void OnPostprocessModel(GameObject gameObject)
{
   var transforms =
gameObject.GetComponentsInChildren<Transform>();
   foreach (var transform in transforms)
   {
       if (RoundPosition) transform.localPosition =
RoundVector3(transform.localPosition, Accuracy);
       if (RoundRotation) transform.localEulerAngles =
RoundVector3(transform.localEulerAngles, Accuracy);
       if (RoundScale) transform.localScale =
RoundVector3(transform.localScale, Accuracy);
   }
}

private Vector3 RoundVector3(Vector3 vector, int accuracy)
{
   return new Vector3(
       Mathf.Round(vector.x * accuracy) / accuracy,
       Mathf.Round(vector.y * accuracy) / accuracy,
       Mathf.Round(vector.z * accuracy) / accuracy
   );
}
```

Great! Now, to add the created module to the settings window, open the **ExtendedImportModuleList** file. You can do this manually or through the interface: go to **Tools/Extended Import Settings**, select any model, and click the **"Edit Modules"** button in the upper right corner of the window.



In the opened file, add the created **RoundTransformModule** class. The order of modules in this list affects the sequence of post-processor execution, so in this case, it is best to place it at the end.

```
public class ExtendedImportModuleList : ScriptableObject
{
   public AttributeRemoverModule AttributeRemoverModule;
   public ChangePivotModule ChangePivotModule;
   public SubmeshCombinerModule SubmeshCombinerModule;
   public SimpleUVGeneratorModule SimpleUVGeneratorModule;
   public UVScaleOffsetModule UVScaleOffsetModule;
   public RoundTransformModule RoundTransformModule;
}
```

After making all the changes, open the settings window — the newly created module should now appear. Select any model to verify that everything is working correctly and that your module has been successfully added. Now it's ready to use!

By the way, if you need a more customized appearance, you can implement a custom module rendering by overriding the **OnInspectorGUI** method inside the **RoundTransformModule** class.

```
public override void OnInspectorGUI(SerializedProperty
moduleSerializedProperty, object assetImporter)
{
   //YOUR GUI
}
```

Full code for the RoundTransformModule module

```
using System;
using ExtendedImportSettings;
using UnityEngine;

[Serializable]
public class RoundTransformModule : ExtendedImportModule
{
   public bool RoundPosition;
   public bool RoundRotation;
   public bool RoundScale;
   public int Accuracy = 10000;

   public override void OnPostprocessModel(GameObject gameObject)
   {
       var transforms = gameObject.GetComponentsInChildren<Transform>();
       foreach (var transform in transforms)
       {
           if (RoundPosition) transform.localPosition =
RoundVector3(transform.localPosition, Accuracy);
           if (RoundRotation) transform.localEulerAngles =
RoundVector3(transform.localEulerAngles, Accuracy);
           if (RoundScale) transform.localScale =
RoundVector3(transform.localScale, Accuracy);
       }
   }

   private Vector3 RoundVector3(Vector3 vector, int accuracy)
   {
       return new Vector3(
           Mathf.Round(vector.x * accuracy) / accuracy,
           Mathf.Round(vector.y * accuracy) / accuracy,
           Mathf.Round(vector.z * accuracy) / accuracy
       );
   }
}
```

Template for Per-Mesh Settings

If you need a module with individual settings for each mesh, you can use the ready-made template. It allows you to easily configure the module for your specific tasks.

Additionally, you can open any existing module and see how it is implemented — all of them use this template.

```
using UnityEngine;

namespace ExtendedImportSettings
{
   public class PerMeshTemplateData : PerMeshDataBase
   {
       public string Parameter;
   }

   [System.Serializable]
   public class PerMeshTemplateModule :
PerMeshModuleBase<PerMeshTemplateData>
   {
       //All parameters are moved to the PerMeshTemplateData class.
       //Here you can place general parameters for the entire model.

       protected override void OnPostprocessModelPerMesh(Mesh mesh,
PerMeshTemplateData data,
           GameObject meshGameObject, GameObject rootGameObject)
       {
           //Here, all necessary operations with the mesh are performed.
       }
   }
}
```

Congratulations! 🎉 You have successfully created and configured your own module and implemented a post-processing for it. Now your tool is ready to use and will help automate model processing in Unity.

Thank you for your purchase! I appreciate your choice and hope this asset makes your work more convenient and efficient.

Wishing you productive work and awesome projects! May your module be useful and simplify routine tasks. Good luck with your development!

If you have any questions or suggestions, feel free to reach out via email: nurzzz27@gmail.com. I'm always happy to help!