USER GUIDE FOR TLFORGE 2024

ALEJANDRO SAINZ

Contents

Introduction

Thank you for purchasing!

The Tag and Layer Forge is an intuitive tool designed to enhance the workflow within Unity by speeding up the time dedicated to configure your project customizing Tags, Layers and Physics Collisions directly from the inspector of a GameObject.

Feel free to contact for any enquiry.

Email: alejasainzmartinez@gmail.com

Features

Direct Management from the Inspector: Perform all **Tag** and **Layer** operations without leaving the context of your **GameObjects**. Manage your project settings all from one place.

Efficiency: Save valuable development time with streamlined operations.

Clarity: Gain a better understanding of your project's collision matrices with an intuitive visual interface.

Setup

Requisites

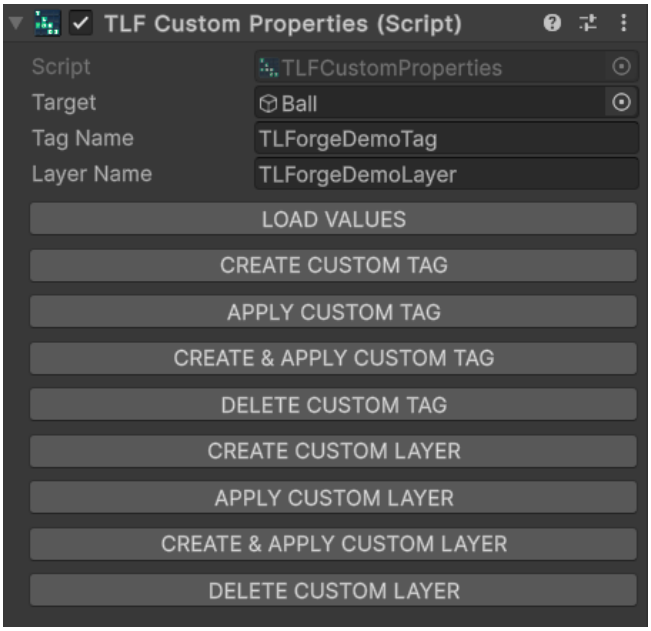
The Unity Tag and Layer Forge has been developed and tested using **Unity 2023.2.7f1**.

Configuration

TLForge will be fully available in your project just by importing the package, without any extra configuration.

Tool Overview

TLForge presents three main components the user will use more often



1-TLFCustomProperties allow the user to define a **Tag Name** and a **Layer Name** to apply to that specific **Target** GameObject.

The component shows a series of buttons that handles from Tag/Layer creation to deletion. This means the corresponding name will be stored or removed from Unity's settings.

Once a name gets created and applied the Target will automatically have that defined value.

This is the main core of the tool.

2-TLFLayerCollisionMatrix2D allow the user to define the state of the project's Physics 2D collision matrix by showing how the existing layers interact with each other.

Every layer interaction is shown as a checkbox the user can mark to set a new collision detection.

Layers can be associated by the indices shown vertically and horizontally.

This component makes possible to define presets of Matrices by exporting the custom matrix to a path defined in the field Exportation Path.

The user can define an **Asset Name** for the Matrix preset and an **Exportation Path** and an **Importation Path**.

The user can either defined both paths by writing or by using **SELECT EXPORT PATH** and **SELECT IMPORT PATH**.

Once a configuration satisfies the user needs is possible to create a Matrix preset by using **EXPORT MATRIX**.

Every time the user selects the GameObject with **TLFLayerCollisionMatrix2D** component the preset matrix defined will be imported.

If the user makes changes to the matrix as long as those changes are not saved by exporting the matrix again, the **IMPORT MATRIX** feature will load the matrix without changes.

Alternatively, the **IMPORT EXISTING PRESET** feature can load any of the created presets without having to define an Importation Path or an Exportation Path.

3-TLFLayerCollisionMatrix3D is a Physics based version of the TLFLayerCollisionMatrix2D component.

Demo Scenes

A Demo folder is inside TLForge directory showing the dimensions with which the tool work.

2D Demo

Exploring the scene

-Open the scene called **"TLForge_2D_Demo"**. This scene presents a 2D based collision test on a canvas with UI buttons that access some of the TLForge features.

-Select any **GameObject** on the scene and deploy the existing tags and layers to see a **TLForgeDemoTag** and a **TLForgeDemoLayer** already created for demonstration purposes.

-Check the collision matrix existent in the project by selecting the TLForge tool and looking for the **TLFLayerCollisionMatrix2D** component. The matrix shows all interactions with **TLForgeDemoLayer** deactivated.

Running the scene

-Hit Play mode and the demo will show a 2D bouncing circle falling and hitting the floor. This collision occurs because the layers on the ball and the floor (default and UI respectively) are actually interacting with each other.

-Hit the **New Layer** button on the scene. The circle will be on **TLForgeDemoLayer** and then will fall below the floor.

-Hit the **Reset** button to return the scene to its initial values.

-Click on the New Preset button. This action switches to a preset that has been predefined in your project. Specifically, this second preset is designed to register collisions between the layer TLForgeDemoLayer and layers marked as UI and Default.

- After activating the new preset, proceed by clicking on the New Layer button. You will notice a significant change: the circle will no longer overlap the floor. This behavior is a direct result of the collision settings defined in the newly activated preset.

-You can now change between layers using **New Layer** / **Original Layer** and the circle will stay still.

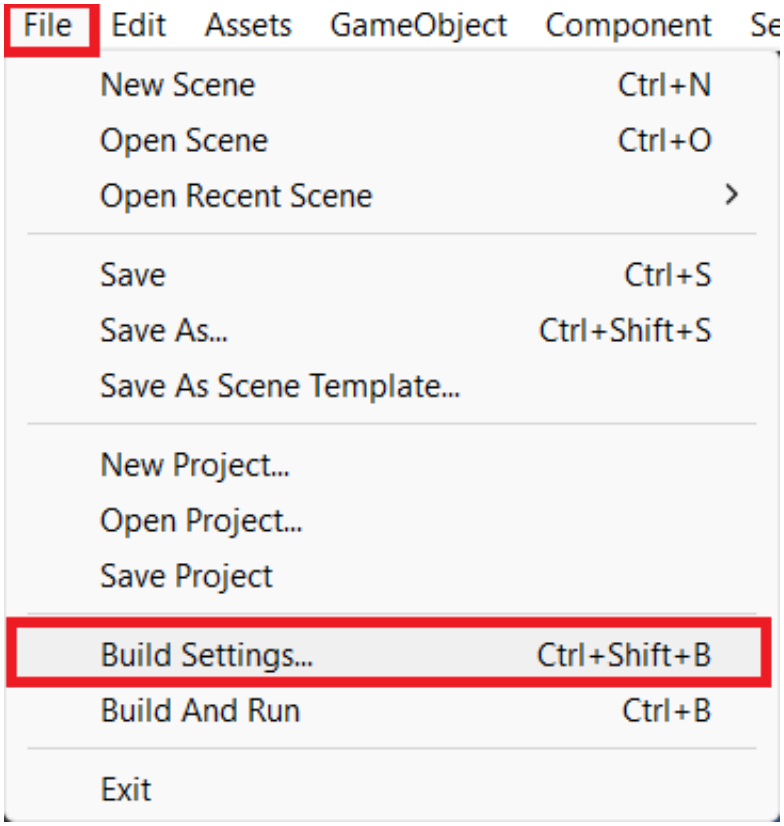
-Finally hit **Original Preset** and the circle will behave as it was when you ran the demo.

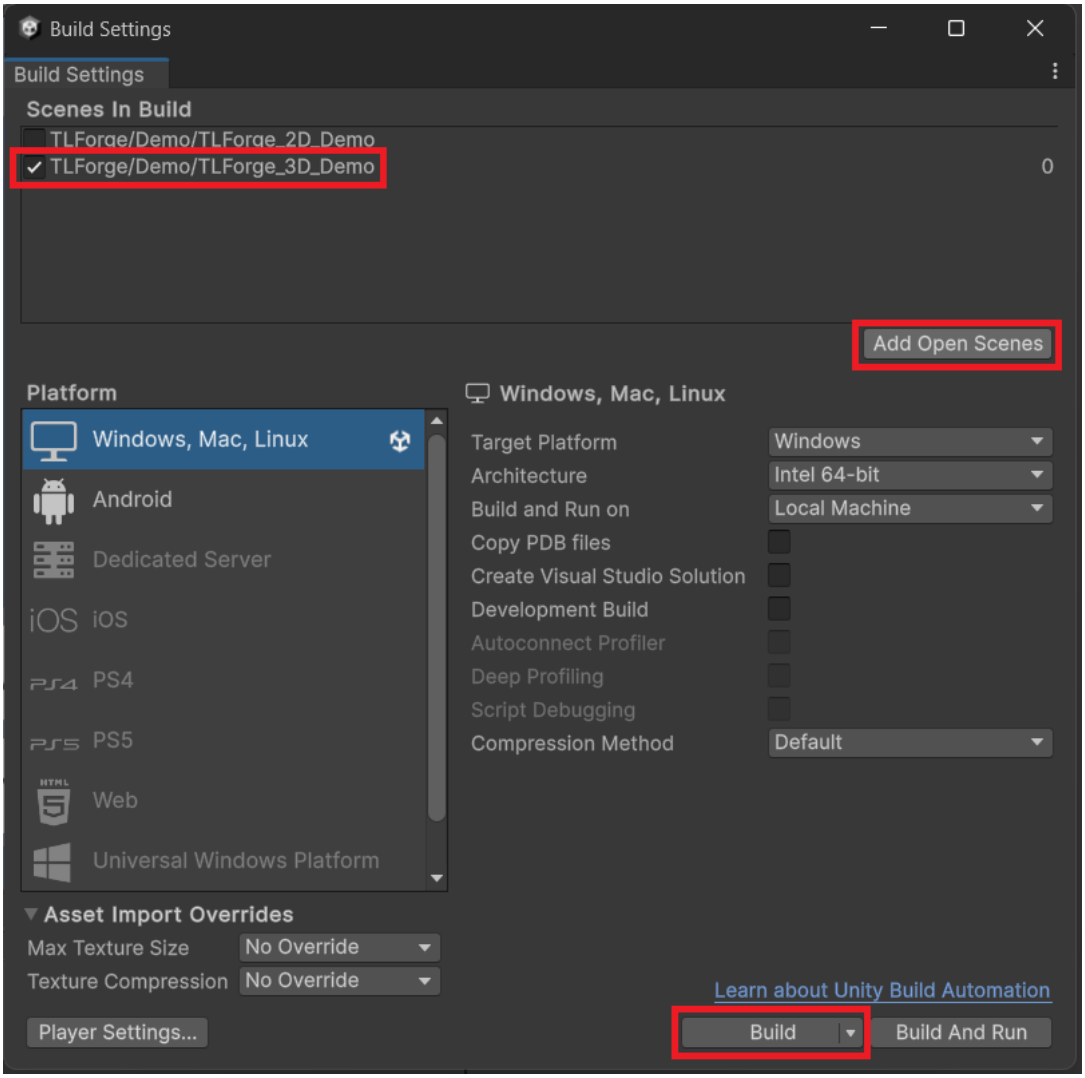
-The demo also allows the user to modify the circle's tag through the UI using **New Tag** and **Original Tag** buttons.

Building the scene

-To test one last thing, after hitting **Exit** button to stop the demo go to:

File → Build Settings → Add Open Scenes → TLForge/Demo/TLForge_3D_Demo → Build.





-After you select a desired directory and the demo build gets created you can run the build and check everything working just as it was on the Unity Editor.

3D Demo

A second scene called **"TLForge_3D_Demo"** presents a 3D based collision test on a canvas with UI buttons that access some of the TLForge features.

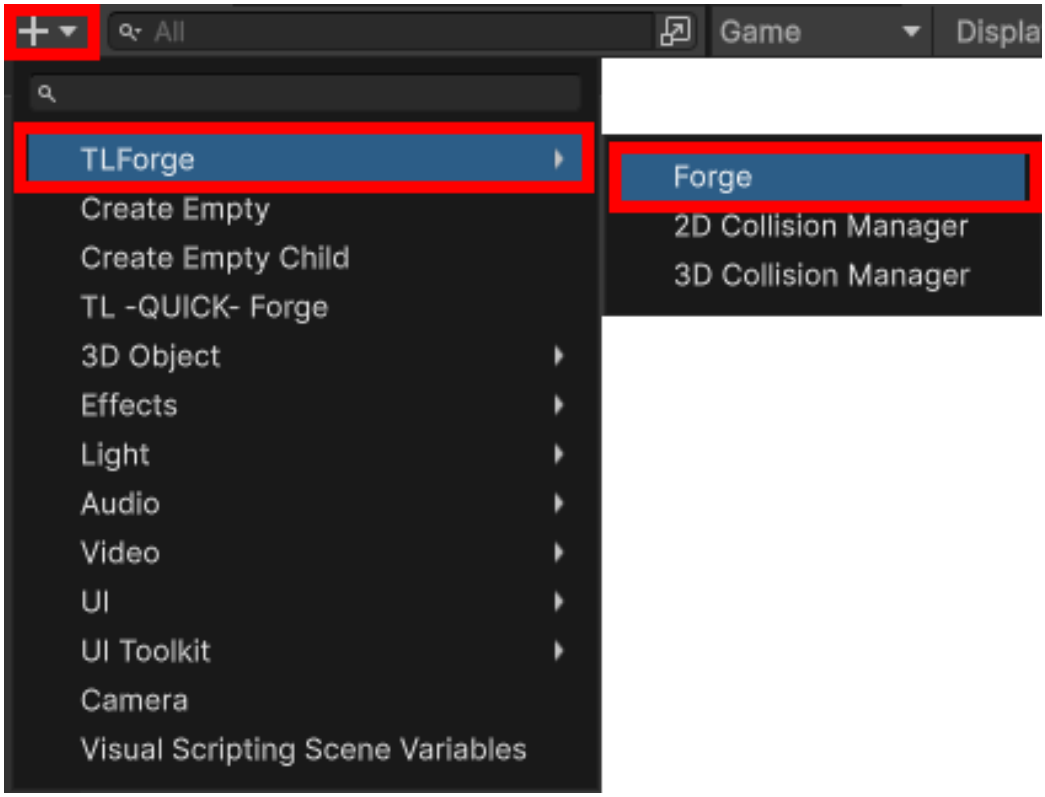
The similarities to the previous scene are significant, therefore if you followed the previous steps, you only need to know that Unity manages collisions across both layers and dimensions. That is why on this scene the main difference with the previous one is that you need to work with **TLFLayerCollisionMatrix3D** component instead of the 2D one because as it was mentioned before, the 3D version is Physics based.

How to use the package in your project

Unity Editor support

To create a Forge

If you are interested in being able to quickly and easily modify your Tags and Layers, while visualizing the impact of said changes in the collision matrix and also being able to create collision presets, add a Forge to the scene using:

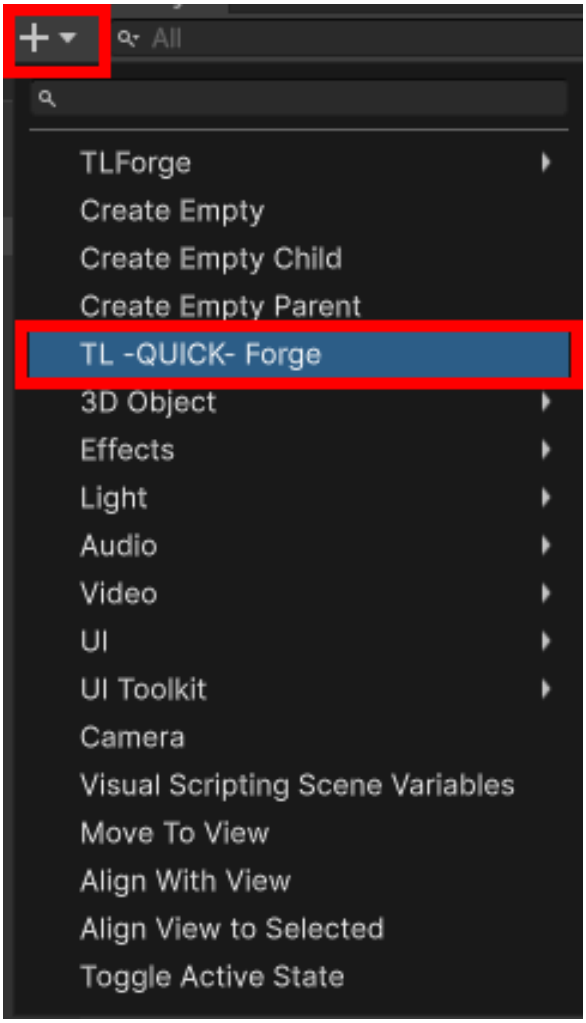


To modify Tags and Layers

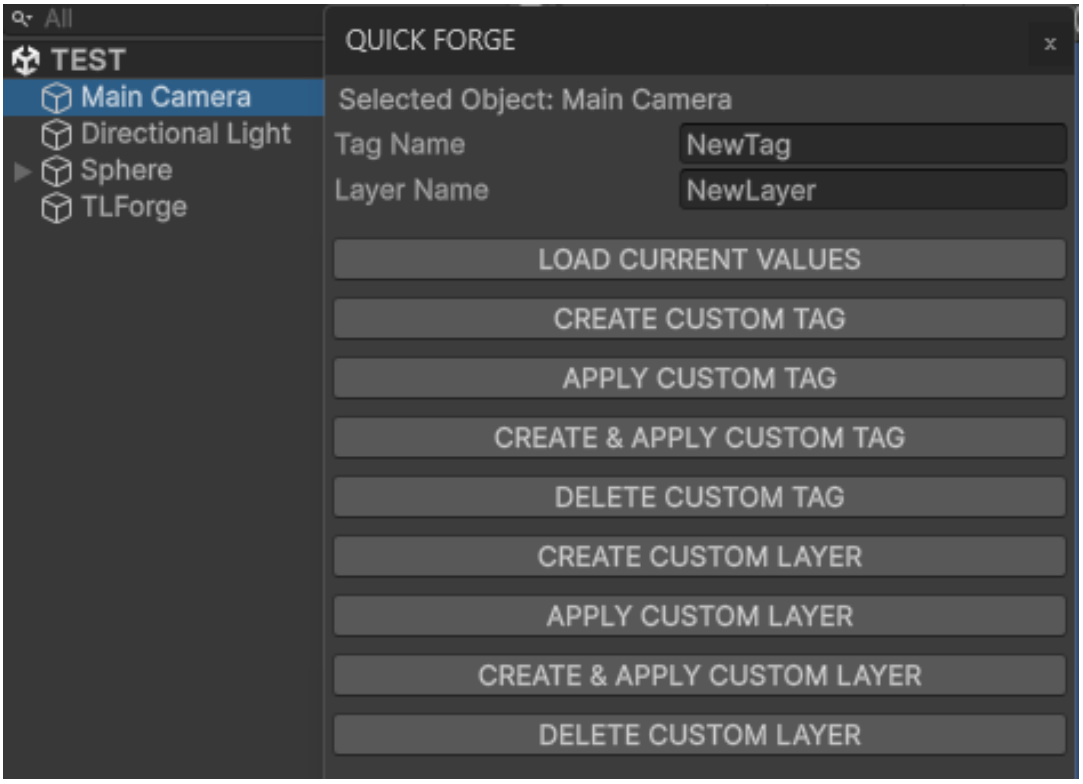
After creating a Forge, select the TLForge GameObject and in the TLFCustomProperties component define a tagName and a layerName. Then with the different buttons it is possible to Create, Apply and Delete Tags/Layers by writing in their corresponding field. You can also speed up the Application process using Create & Apply.

To modify Tags and Layers without a Forge

If you want to speed up the whole process and don't want to have a Forge with all the components on the scene but need to change a Tag or a Layer, you can go to:

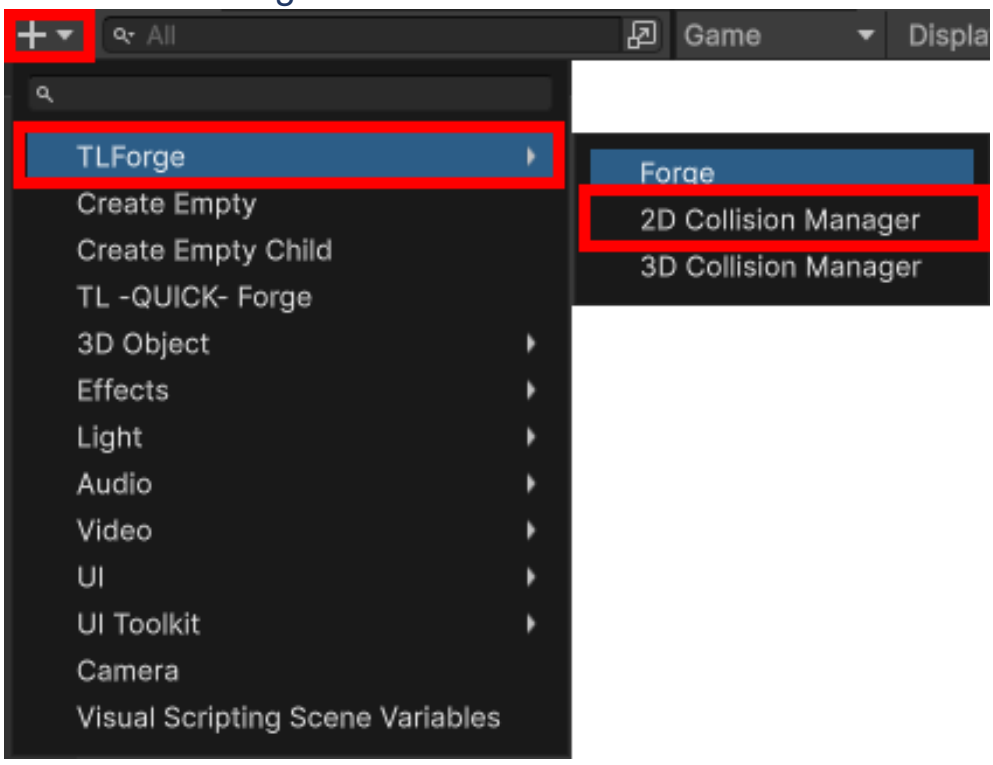


Or you can right click a GameObject and select the option “TL -QUICK- Forge” as well. Either way this window will open:



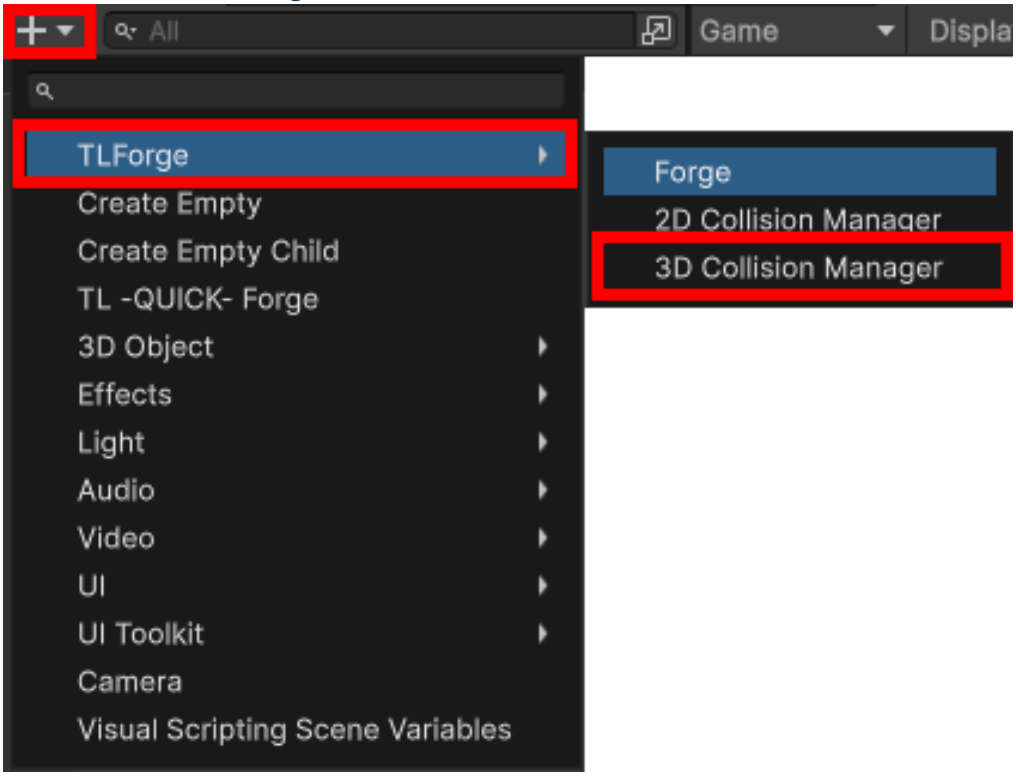
Quick Forge lets you change right away the Tag and the Layer of a selected GameObject without having a Forge on the scene or having to define a Target.

To create a 2D Collision Manager



This will create a GameObject with a TLFCollisionMatrix2D component, giving you access to create collision presets to modify dynamically the state of the Unity Physics 2D collision matrix.

To create a 3D Collision Manager



This will create a GameObject with a TLFCollisionMatrix3D component, giving you access to create collision presets to modify dynamically the state of the Unity Physics collision matrix.

Scripting support

Although the tool has methods for removing or adding tags and layers, these features are not allowed in project builds. Therefore, only the methods that users are likely to use through code, and that will run both in the Editor and in a Build, will be covered.

Needed fields

```
// Required fields to edit Tags and Layers

[SerializeField]

private TLFCustomProperties cProperties;

[SerializeField]

private GameObject newTarget;

// Required fields to edit Layer Collision Matrices

[SerializeField]

private TLFCollisionMatrix2D matrix2D;

[SerializeField]

private TLFCollisionMatrix3D matrix3D;
```

To modify the current Tag of a GameObject

```
// Creates (Editor only) and applies the defined Tag to the ‘newTarget’ value.

cProperties.CustomizeThenApplyTag(newTarget, "TLForgeDemoTag");

// Find (Editor and Build) and applies the existing Tag to ‘newTarget’

cProperties.ApplyCustomizedTag(newTarget, "TLForgeDemoTag");
```

To modify the current Layer of a GameObject

```
// Creates (Editor only) and applies the defined Layer to the ‘newTarget’ value.

cProperties.CustomizeThenApplyLayer(newTarget, "TLForgeDemoLayer");

// Find (Editor and Build) and applies the existing Layer to ‘newTarget’

cProperties.ApplyCustomizedLayer(newTarget, "TLForgeDemoLayer");
```

To modify the Collision Layer Matrix preset

```
// Import the ‘TLForgeDemoMatrix2D’ json stored in ‘EditorData/CollisionData/2D/’

matrix2D.Import("EditorData/CollisionData/2D/", "TLForgeDemoMatrix2D");

// Import the ‘TLForgeDemoMatrix3D’ json stored in ‘EditorData/CollisionData/3D/’

matrix3D.Import("EditorData/CollisionData/3D/", "TLForgeDemoMatrix3D");
```

FAQs

Why an error shows up every time the TLForge GameObject is selected?

If an error with a message “There is no CollisionMatrix to import at Assets/EditorData/CollisionData/LayerCollisionMatrix.json” shows up every time the TLForge is selected is because the tool is loading some inexistent preset Matrix in your project. The error won't show up anymore if a correct preset is set.

Each time a new TLForge is created, the LayerCollisionMatrix is set as a default preset for both dimensions. However, it's very likely that this default preset does not exist within the project. Consequently, it becomes necessary to create a new, valid preset.

Why the Target of the TLFCustomProperties can never be null?

A target is always necessary for the **TLFCustomProperties** to work properly and in order to avoid null data, if no target is referenced, the GameObject owner of the component will be the target.

Do I need to type the entire name of a Tag or a Layer that I want to delete?

On the current version of the project if you want to delete a Tag or a Layer that no GameObject is using, you need to write the whole name, but if the Tag or the Layer is in some GameObject, you can set that GameObject as Target in **TLFCustomProperties** and select **LOAD VALUES** button to automatically get its data into the desired fields.