Inspector Gadgets

v6.5 Created by Kybernetik

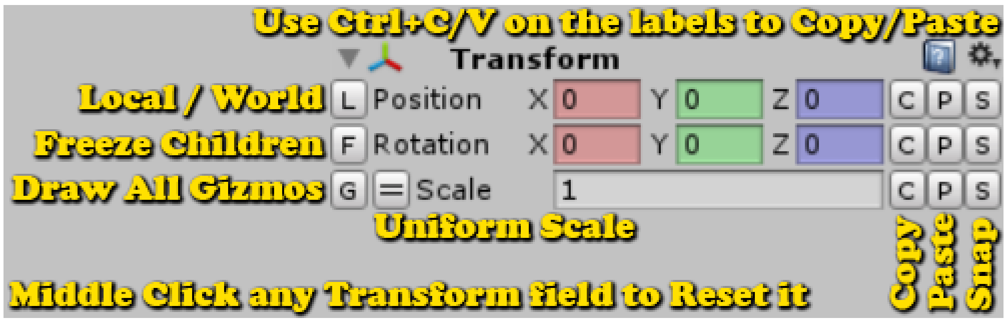
This is a quick overview of Inspector Gadgets. The Online Documentation is much more detailed.

1. Preferences

Inspector Gadgets Pro allows you to customise its settings in the *Edit/Preferences* window.

2. Transform Inspector

Inspector Gadgets automatically replaces the default Transform Inspector to add various controls:



3. Drag & Drop Sub-Assets [Pro-Only]

Unity allows assets to be saved inside other assets to group them in the Project window, however it does not have an inbuilt way to add or remove those sub-assets manually so Inspector Gadgets gives you the ability to do so while holding `Alt` (or you can set a different modifier key in the *Edit/Preferences* window).

You can extract a sub-asset back into a regular asset by dropping it onto a folder. Unfortunately, it is not possible to just drop it into the blank area in the current folder.

4. Auto Hide UI

Many users find it annoying having a Screen Space UI Canvas take up a massive amount of space in the scene. To work around this, the first time you select a UI object after importing Inspector Gadgets, it will ask if you want to automatically show and hide the UI layer.

- On UI selected: show UI layer, enter 2D orthographic mode, and focus the camera on the selected object.
- On UI deselected: hide UI layer and return camera to the previous state.
- The UI layer will be automatically shown when you close the Unity Editor just in case the next project you open doesn't have Inspector Gadgets.

5. Context Menu Functions

Inspector Gadgets automatically adds various useful functions to the context menu (right click menu) of each field in the Inspector based on its type:

- These functions only support multi-object selection in Inspector Gadgets Pro.
- Inspector Gadgets Pro also has several additional functions specific to Object Fields.
- Most types have `Copy` and `Paste` functions which allow you to copy values between fields in Unity as well as to and from other programs.
- The fields in the `Transform` Inspector have functions to snap them to the grid, raycast down and snap to the ground, and rotate to look at another object.
- The fields in the `RectTransform` Inspector have functions to square them (set the height equal to the width or vice versa) and to snap them to the edges of its siblings in a particular direction (Right/Up/Left/Down).
- Randomize within common ranges:
  - 0-1, 0-100, 0-360, 0-CurrentValue for `float`
  - Random value for `enum`
  - Random `Vector2` in a unit circle
  - Random `Vector3` *on* or *in* a unit sphere
  - Random `Quaternion`
  - Random euler angles
  - Random hue for `Color`
- Convert between degrees and radians for `float`.
- Set common vectors: zero, right, up, forward, one.
- `Normalize` vectors.
- `String` to lower or upper case.
- `Log` the current value of the field.
- `Help` functions to open the Inspector Gadgets Documentation or do a Google search for the name of the target script.

6. Object Reference Fields [Pro-Only]

Inspector Gadgets applies various improvements to `Object` reference fields (any field that inherits from `UnityEngine.Object` such as any reference to a `GameObject`, `Component`, or any asset).

- A `Get` button is shown if a field is empty to easily find an appropriate reference.
- While a reference is assigned, it will be shown with a foldout arrow to show the referenced object's editor nested below the field.
- If you drag and drop a `GameObject` into a `Component` field, Unity would normally just assign the first `Component` of the correct type, but if there is more than one then Inspector Gadgets will instead show a context menu for you to choose which one you want.
- In addition to the general Context Menu Functions added by Inspector Gadgets, there are several more which are specific to `Object` fields:

Function	Effect
`Null`	Sets the field to `null`.
`Destroy`	Destroys the referenced object.
`Open Inspector`	Opens a new Inspector window to display the selected object's details.
`Find Object of Type`	Calls `Resources.FindObjectsOfTypeAll` and selects the object with a name closest to the field's display name.
`Find Asset of Type`	Calls `IGEditorUtils.FindAssetOfType` and selects the asset with a name closest to the field's display name.
`Pick from List`	Gathers all scene objects that could be assigned to the field and display a list to let you choose which one you want. The `Pick from Prefabs` function will do the same for assets in the project (note that it may take several seconds or more to execute in large projects). Non-`Component` fields go straight to the assets list.
`Find Component (Progressive Search)`	`Component` fields only. Calls `IGUtils.ProgressiveSearch` and selects the `Component` with a name closest to the field's display name.
`Add Component`	`Component` fields only. Adds a `Component` of the appropriate type to the current `GameObject` and assigns it to the field. If there are multiple types inheriting from it, this function is displayed with a sub-menu for each of them.
`Create New Instance`	`ScriptableObject` fields only. Creates a new instance of the appropriate type and assigns it to the field. If there are multiple types inheriting from it, this function is displayed with a sub-menu for each of them.
`Save as Asset`	Opens a window asking where you want to save the referenced object. This is particularly helpful after using `Create New Instance` to make a `ScriptableObject` because they are not automatically saved unless referenced by a scene object.

7. Missing Scripts

Inspector Gadgets improves the inspector for missing scripts by adding a button to easily remove the component in question as well as one to open a window that will search through all assets and scenes to find any more missing scripts. This window can also be opened from the *Edit/Preferences* window.

The Missing Script Hunter window tries to suggest other possible alternatives for each missing script based on its name and the names and types of each of its serialized fields. Unfortunately, Unity tends to lose this data after scripts go missing so it isn't always possible to offer any suggestions.

8. Auto Prefs

The `InspectorGadgets.AutoPrefs` class contains a group of nested classes which simplify the way you can store and retrieve values in `PlayerPrefs` and `EditorPrefs`.

```
// First you declare your pref with the key and default value (optional):
public static readonly AutoPrefs.Bool MyPref = new AutoPrefs.Bool("MyPref", true);

// If you don't want to specify a default value, you can use an implicit cast:
public static readonly AutoPrefs.Bool MyOtherPref = "MyOtherPref";

// Then you can get and set the value without using the key everywhere:
if (MyPref)// Or MyPref.Value.
{
    MyPref.Value = false;
}
```

- `AutoPrefs.Bool` stores its value in `PlayerPrefs` while `AutoPrefs.EditorBool` stores its value in `EditorPrefs`.
- Other pref types are also available: `float`, `int`, `string`, `Vector2`, `Vector3`, `Vector4`, `Quaternion`.
- You can create your own `AutoPref` types by inheriting from `AutoPref<T>`.