EXPOSED

RULE-BASED REFERENCES

GETING STARTED

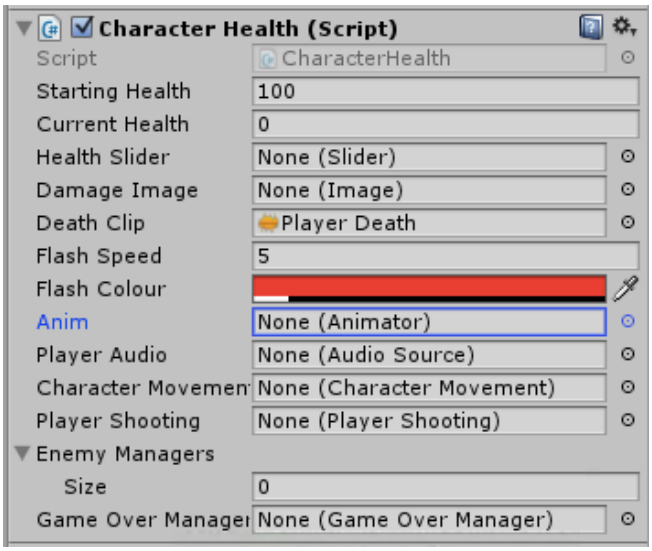
SUMMARY

EXPOSED helps you to easily and automatically map object references for your components. You can set rules for creating these references which are completely reusable also for other components. Everything is configurable in Inspector and you can see the results of the rules settings and references immediately. It will help you to dramatically reduce repeatable reference drag&dropping or reference selecting. It will also help you to visualize reference settings if you prefer writing it directly in code and to get rid of initialization performance overhead.

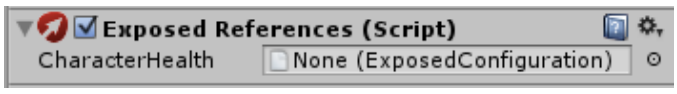
There's no need of coding when setting the rules, you can always use basic predefined rules. But if you need to define extra conditions for obtaining references for scene objects, you can write your own rules and connect them through Exposed API. Or you can use query, which is most flexible and realtime (query feature coming soon, more on this in Query section).

Plugin is entirely written in C#, all source code files are included.

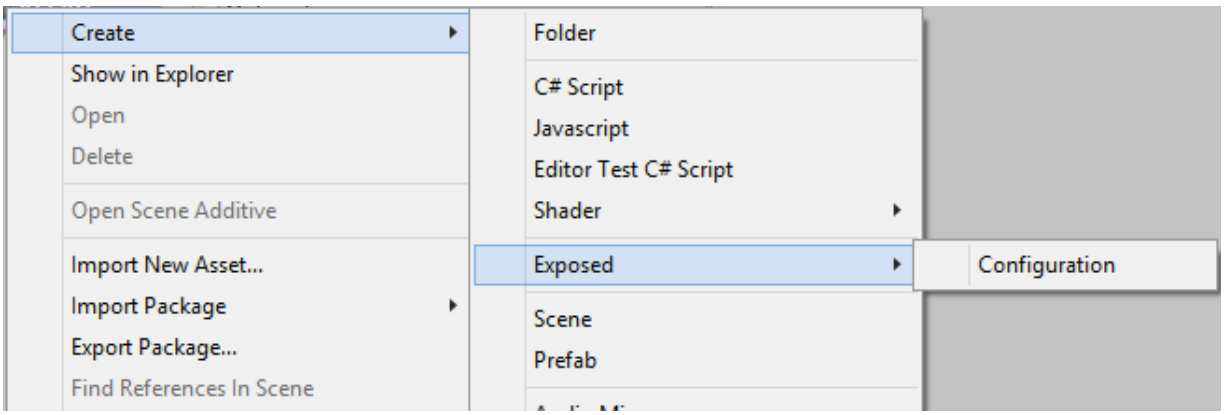
BASICS



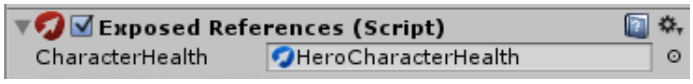
Let's assume that you have GameObject in the scene with component you need to setup. This component has multiple references to it's sub-objects or even references outside of it's scope to somewhere else in the scene. Standard procedure would be to drag&drop or select these references or set them dynamically by implementing them, but here we will use special EXPOSED component and benefit from all of it's great features.



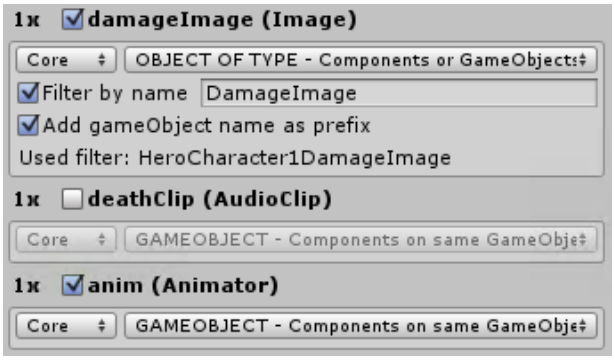
The process begins simply by adding **ExposedReferences** script component to gameObject with your component. At this moment, you can see just the list of components on this gameObject on it. For being able to set the rules for automatic reference settings, you have to create Exposed configuration asset first.



It can be done through Unity main menu (**Assets/Create/Exposed/Configuration**), or with help of context menu directly in Project View (**Create/Exposed/Configuration**). Then you are able to set reference to this configuration file in ExposedReferences component settings in Inspector. After having set this configuration asset, you are able to define rules for setting references of your component. You can use this configuration file anywhere you want, most probably you will use it for same component types, but you can even use it for different components if the variable names are the same.



SETTING RULES



For each reference variable, you can set whether the rules should be applied to it or not. When it's enabled, setting of references is completely managed by rules and you won't be able to change them by hand (through drag&drop or selection). You can disable the rules anytime and take back the control of creating them.

You can set rules for variables which refers to single object (**Component** or **GameObject**), or arrays (**Component[]** or **GameObject[]**) or even lists (**List<Component>** or **List<GameObject>**). These variables have to be serializable (basically all variables you see in Inspector, more on this topic in Unity documentation).

For enabled rules, you will set the type of rule:

- Core - predefined rules *(more to come in future updates)*
- Custom - your own implemented rules *(more on this in Advanced part of this manual)*
- Query - …coming soon… *(more on this in Query part of this section)*

Then you are able to choose what rule will be exactly applied. From predefined rules, you can choose from rules searching in children gameObjects, parent gameObjects, on the same

gameObject, with specific tag or specific type, etc. *(More about Core Rules in it's own section.)*Now you're done and you can immediately see the results of application of these rules in your component's references fields. But how about performance, aren't these kind of searches quite expensive in terms of performance?

INITIALIZATION

Application of rules in editor happens instantly. In running application, it could be big performance problem, so these references are initialized for each gameObject with them **just before the game in editor is switched to playmode**. It works **automatically**. The same principle applies also for

platform builds (installations for Windows, Mac, iOS, Android, etc.). References are set automatically during **scene post processing**.

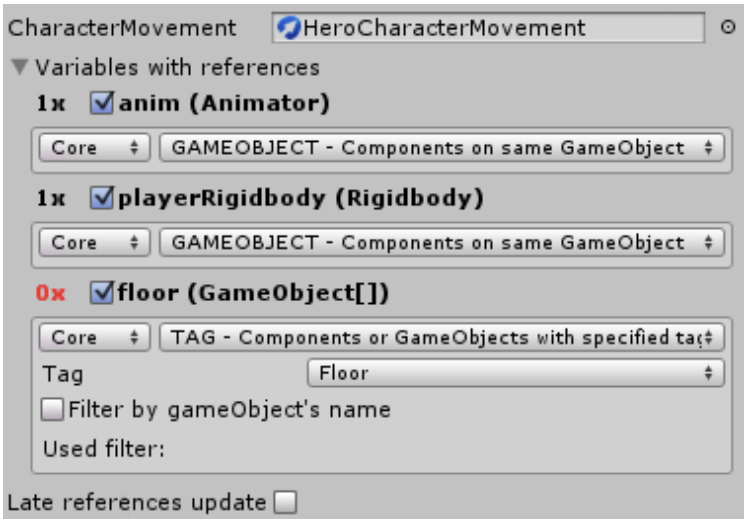
From performance point of view, in final build, the initialization of scenes is faster, because you don't have to execute this kind of search in Awake() method (like for using GetComponentsInChildren function). And you are always sure, that you are caching these kind of references and not using them for example in Update function every frame.

But sometimes you might need to turn this mechanism off and initialize references in Awake() method as you would do when implementing code by hand. For this purpose, you can disable **PreInitialization** on **ExposedConfigurationManager** asset. Sometimes it might be useful for example in Editor for faster playmode loading, because there's some extra overhead of searching for ExposedReferences component when PreInitialization is enabled. For platform builds, preferably leave PreInitialization enabled for achieving better performance.

Initialization of these kinds of references can be achieved only for GameObjects in scenes, but how about prefabs? Even for them the references can be set automatically, when they're instantiated, or you can manually run **ExposedReferences.UpdateReferences()** method, which will update references according to the rules at that precise moment you want.

PREFABS

For prefabs, there's no way how to ensure references by rules in advance, before the prefab is instantiated in scene. Their references are therefore updated in Awake() method. But sometimes, it might be useful to update references later in Start() method. Mostly because of changing instantiated prefab's parent in hierarchy, after it is created. Initialization in Awake() would be performed too early before reparenting. It's possible to use **Late References Update** checkbox to eliminate this early update and use Start() method instead.



Sometimes you might need to update references for prefab at the exact time, without relying on late references update described above. You can always update references manually by calling **UpdateReferences()** method **on ExposedReferences component.** This can be also useful for updating references on GameObject you are adding newly instantiated prefab to (as a child). Then by updating references manually, you get recent reference array (or list) of references. Or you can simply **use List** for storing references and then add the newly created instance to this list by **List.Add().**

```
var gunBarrelEndInstance = Instantiate(gunBarrelEnd);
gunBarrelEndInstance.transform.parent = transform;
gunBarrelEndInstance.GetComponent<ExposedReferences>().UpdateReferences();
```

CORE RULES

Exposed plugin comes with a set of predefined rules (more to come with future updates). Every rule can have more settings for filtering output results. Type of searched component is determined

directly from the type of variable. You can also search just for GameObject, not for specific component.

- **GameObject** - searching for components on same GameObject (useful also for native Unity components like Rigidbody, Animator, or even ExposedReferences script component itself, when updating references on demand)

- **Children** - searching for components or GameObjects in children
    - nonactive included - nonactive objects in hierarchy are included in search
    - self included - components on this GameObject are included in search

- **Parents** - searching for components or GameObjects in parents
    - nonactive included - nonactive objects in hierarchy are included in search
    - self included - components on this GameObject are included in search

- **Tag** - searching for components or GameObjects with specified tag in scene
    - Tag - Unity tag
    - Filter by gameObject's name - filter results by the name of current gameObject (current filter is then shown in *Used filter* label below)

- **Object Of Type** - searching for components or GameObjects of specified type in scene
    - Filter by name - name of GameObject
    - Add gameObject name as prefix - ads current object's name to filter (current filter is then shown in *Used filter* label below)

QUERY (coming soon)

Possibility of defining own rules without programming custom rules, just by specifying string search expression. Connecting different search results together and filtering them through wide range of rules.

ADVANCED TOPICS

CUSTOM RULES

Adding new, custom rule, can be done by creating new C# class in scripts assembly, which implements **PropertyProcessor** class. To have this class included in the whole build process and to be shown in appropriate enum, **it's name has to end with CustomPropertyProcessor** (like MyDefinedCustomPropertyProcessor).

The methods of IPropertyProcessor interface are responsible for identifying rules globally (GetUniqueId()), showing appropriate label in enum (*GetActionTypeLabel()*), processing search for components or gameObjects of single or array type(*ProcessSingleGameObjectField()*, *ProcessSingleComponentField()*, *ProcessArrayGameObjectField()*, *ProcessArrayComponentField()*) and rendering some additional GUI elements (*RenderAdditionalGui()*).

By adding additional GUI elements, it's possible to define more filtering rules of search result. These settings are saved and serialized through ExposedPropertyConfiguration class. This whole method has to be surrounded by **#if UNITY_EDITOR … #endif** directives. It's necessary for platform builds, because it uses UnityEditor library, which isn't available for them (you shouldn't also include **using UnityEditor**, anywhere in the file without directives surroundings).

DEFAULT CONFIG SETTING FOR SCRIPT

In some cases it's appropriate to set predefined config file for component of certain type. It can be done by **adding variable of ExposedConfiguration type** to this component's properties. In

default setting for script, you can then specify this configuration asset, which will be automatically used when ExposedReferences script is added to gameObject with this user-defined component.

```
public Animator anim;
public AudioSource playerAudio;
public CharacterMovement characterMovement;
public PlayerShooting playerShooting;

public List<EnemyManager> enemyManagers;
public GameOverManager gameOverManager;

public ExposedConfiguration exposedConfiguration;
```