Documentation

Table of Contents

Installation

Package Manager

Component Names can be installed using the Package Manager window inside Unity. You can do so by following these steps:

1. Open the Package Manager window using the menu item **Window > Package Manager**.
2. In the dropdown at the top of the list view select **My Assets**.
3. Find **Component Names** in the list view and select it.
4. Click the **Download** button and wait until the download has finished. If the Download button is greyed out you already have the latest version and can skip to the next step.
5. Click the **Import** button and wait until the loading bar disappears and the Import Unity Package dialog opens.
6. Click **Import** and wait until the loading bar disappears. You should not change what items are ticked on the window, unless you know what you’re doing, because that could mess up the installation.

Component Names is now installed and ready to be used!

Keeping Up To Date

You might want to check from time to time if Component Names has new updates, so you don’t miss out on new features.

The process for updating Init(args) is very similar to installing it, and done using the Package Manager window inside of Unity.

You can do so by following these steps:

1. Open the Package Manager window using the menu item **Window > Package Manager**.
2. In the dropdown at the top of the list view select **My Assets**.
3. Find **Component Names** in the list view and select it.
4. If you see a greyed out button labeled **Download**, that means that your installation is up-to-date, and you are done here! If instead you see a button labeled Update, continue to the next step to update Init(args) to the latest version.
5. Click the **Update** button and wait until the download has finished.
6. Click the **Import** button and wait until the loading bar disappears and the Import Unity Package dialog opens.
7. Click **Import**, then wait until the loading bar disappers. You should not change what items are ticked on the window, unless you know what you’re doing, because that could mess up the installation.

Component Names is now updated to the latest version.

What Is Component Names?

Component Names integrates seamlessly with the Inspector and makes it possible to rename components.

Renaming a Component

To start renaming a component select Rename from its context menu in the Inspector.

Alternatively renaming can be started by first selecting the header of the component in the Inspector and then pressing the ***F2*** button.

After this you should be in name editing mode, and can start typing the new name for the component.

Once you are done you can press ***Enter*** on your keyboard or click somewhere outside the name input field to confirm the new.

To cancel the renaming and revert the name of the component back to its previous state press ***Escape*** while still in the name editing mode.

Custom Name (Default Suffix)

By default the default name of the component gets appeneded to the end of the new name you have given in the Inspector.

So if you input to a BoxCollider's name field "Left Wing", you will henceforth see "Left Wing (Box Collider)" displayed in its Inspector header.

Default Name (Custom Suffix)

If you want to add a custom suffix without affecting the name of the component you can type the suffix you want inside parentheses.

So if you input to a BoxCollider's name field "(Left Wing)", you will henceforth see "Box Collider (Left Wing)" displayed in its Inspector header.

Custom Name (Custom Suffix)

If you want to give a component both a custom name and a custom suffix you can type the name for the component followed by the suffix you want inside parentheses.

So if you input to a BoxCollider's name field "Left Wing (Collider)", you will henceforth see "Left Wing (Collider)" displayed in its Inspector header.

Custom Name

If you want to give a component a custom name without any suffix you can type the name of the component followed by parentheses with nothing inside of them.

So if you input to a BoxCollider's name field "Left Wing ()", you will henceforth see "Left Wing" displayed in its Inspector header.

Assembly Definition

To be able to use the extension methods for getting and setting component names in your own code, you need to create assembly definition assets in the roots of your script folders and add references to the ComponentNames assembly.

Namespace

To reference code in the ComponentNames assembly you also need to add the following using directive to your classes:

```
using Sisus.ComponentNames;
```

Get Name

To get the name of a component use the extension method Component.GetName().

```
using Sisus.ComponentNames;

public class NamePrinter : MonoBehaviour
{
    public Component component;
    private void PrintName() => component.GetName(name);
}
```

In the editor GetName() returns the custom name of the component if it has been given one, and otherwise its default name.

All custom name data is stripped from builds entirely. In builds GetName returns the name of the component's class instead.

Set Name

To set the name of a component use the extension method Component.SetName(string name).

```
using UnityEngine;
using Sisus.ComponentNames;

public class ProgressBar : MonoBehaviour
{
    [Range(0, 100)]
    public int progress;

    #if UNITY_EDITOR
    private void OnValidate() => this.SetName($"Progress: {progress}%");
    #endif
}
```

In the above example SetName() is called during every OnValidate method, which means it gets called when the script is first loaded and every time the values of its members are modified.

This means that the component will always have the name "Progress: X%" where X reflects the the current value of the progress member.

This OnValidate SetName pattern is a convenient way to create dynamic headers that change based on the state of the component.

BaseBehaviour

Component Names also comes with a base class called BaseBehaviour. When you make your component derive from this base class instead of MonoBehaviour, they get a couple of enhancements to their handling of custom component names.

With all xml documentation stripped the BaseBehaviour class looks basically like this:

```
public abstract class BaseBehaviour : MonoBehaviour
{
    public virtual new string name
    {
           get => this.GetName();
           set => this.SetName(value);
    }

    public override string ToString() => this.GetName();
}
```

As you can see, in the BaseBehaviour the name property no longer returns the name of the GameObject to which the component is attached to, but the name of the component itself.

Secondly the ToString() method will also return the name of the component, to make debugging easier.

In cases where you can't or don't want to inherit from a custom base class, you can also implement these two overrides very easily in your own base classes.