UI Toolkit

Glow, Outlines & Shadows

Table of contents

Requirements

Unity 2021.3 or higher is required since that is when Unity added the UI Toolkit Module for runtime use. If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

Keep in mind, UI Toolkit as a whole is still a work in progress and not quite ready for prime time. Unity itself still recommends using uGUI instead of UI Toolkit for runtime applications (source).

Intro (read this first)

The glow effect can be achieved in two ways. Either you simply wrap your element in a Glow element (found in the UI Builder Library) – OR – you use the non-destructive GlowDoument workflow.

The Glow element works just like any other custom control. You drag it in from the library, change the attributes and you are done (don't forget to keep overflow set to visible).

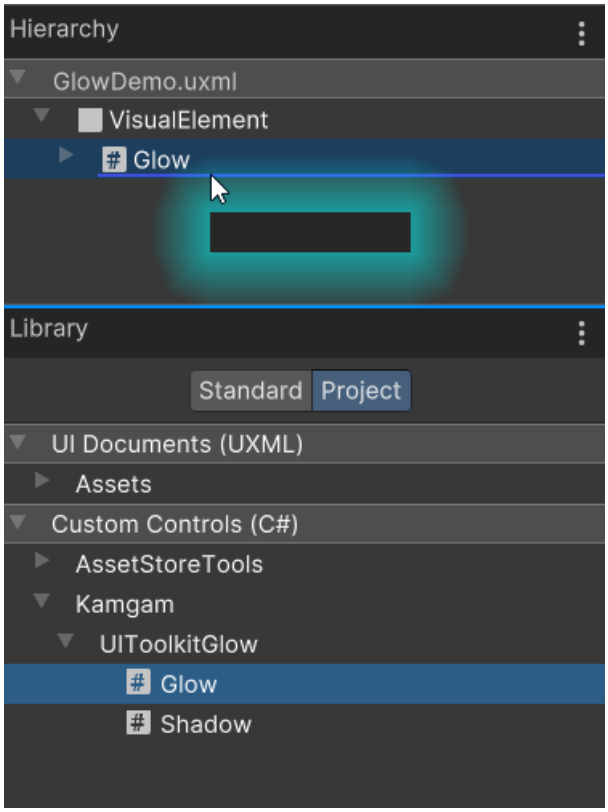
The other workflow requires you to add a GlowDocument component to your UI Document in the scene. Then you can add a special USS class to any element that should have a glow effect. The advantage of this that you do not have to modify your existing hierarchy. You can simply add and remove glow effects by adding USS classes (also works via code at runtime).

On a technical level this works by adding a Manipulator to each element.

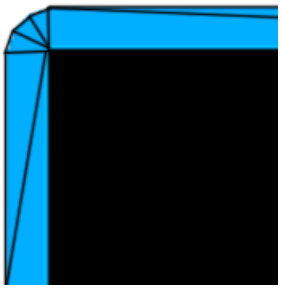
Confused?
Don't worry, below you will find some step by step guides with lots of screenshots and more explanations.

GLOW: Using the Glow element.

The easiest way to use the glow effect is to wrap your elements in a Glow control. You can find it in the **UI Builder Library** under **Kamgam > UIToolkitGlow > Glow**.

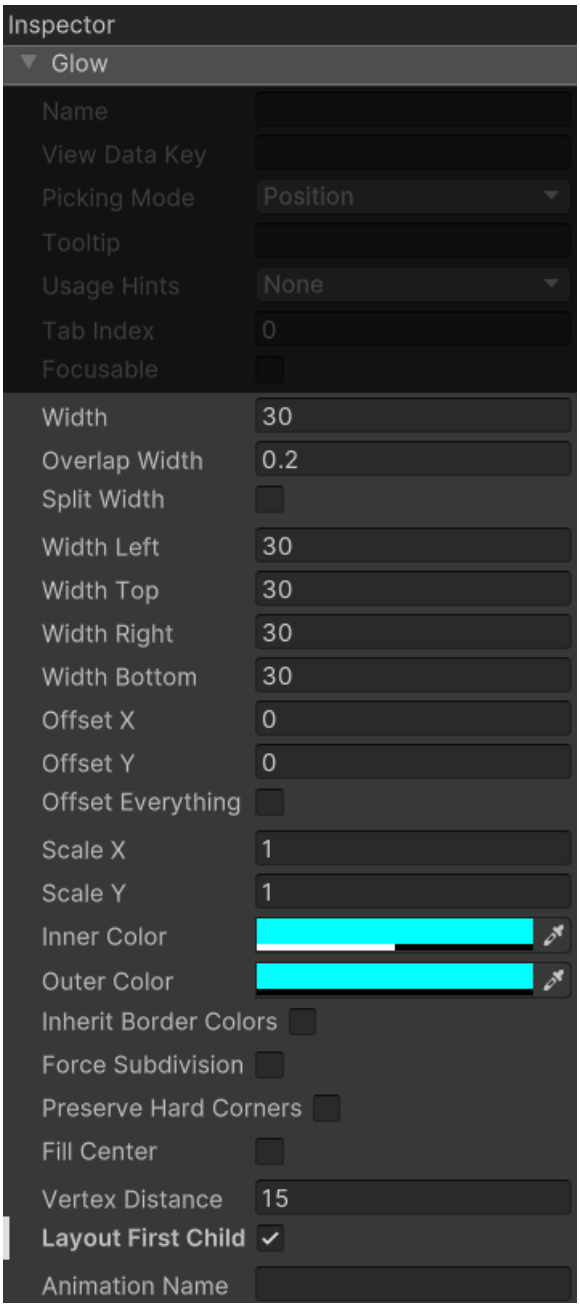


The first thing to understand about the glow effect is that it is achieved by generating a mesh around the element.



The mesh is then colored by tinting the vertices. If you set the inner tint to a color and the outer tint to transparent then you will have a glow effect. If you set both the the same color then you will get an outline effect.

In the UI Builder you can edit all these glow attributes:



Width

Width describes how big the glow / outline is.

Width = 2

Width = 10

Overlap Width

Overlap define how much the glow mesh will overlap on the inside.

Overlap = 0

Overlap = 10

Split Width

Splitting the widths allows you to specify the with in each direction (left, top, right bottom). If disabled then the same „Width“ will be used for all sides.

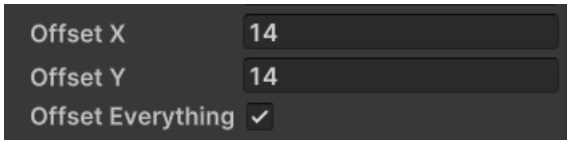
Width Left / Top / Right / Bottom

Offset X / Y

The offset pushes the outer vertices in one direction.

Offset Everything

By enabling this you can offset both inner and outer vertices which means that you are effectively pushing the whole glow mesh around.



Scale X / Y

Scaling allows you to scale the glow mesh. The scale origin is always the center.

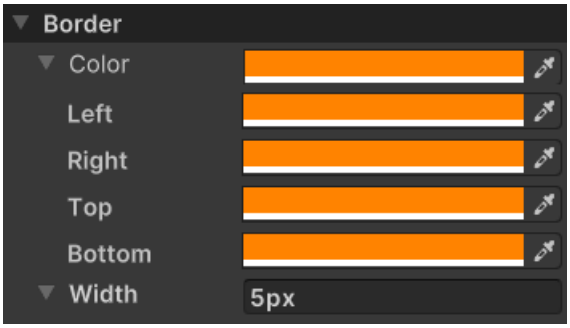
Inner / Outer Color

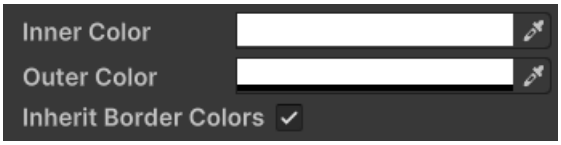
With these you define the color tine on the inside and outside of the glow mesh.

By choosing a transparent outer color you can create a glow effect since the inner color will fade out over distance.

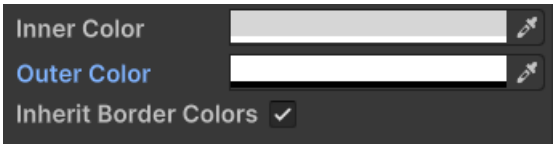
Inherit Border Colors

This automatically sets the colors of the glow mesh to the border color. Since the vertices of the glow mesh are shared between triangles for performance reasons this is best used only for uniform border colors, like this:



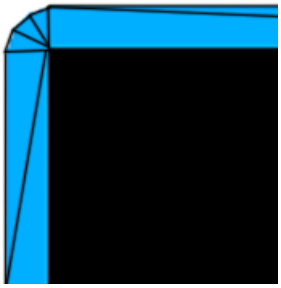


You can use the Inner/Outer colors to tint the inherited glow color.

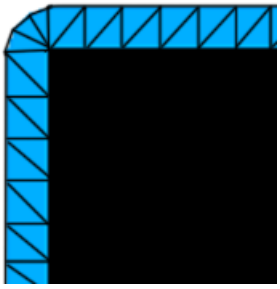


Force Subdivision

Usually the mesh is generated with as few vertices as possible. Like this



However, for animations you may want to have more vertices that are evenly distributed. If „Force Subdivision“ is enabled then even the sides are divided into multiple parts. Like this:



Preserve Hard Corners

If enabled then hard corners are kep hard on the outside. If disabled then they will be rounded.

OFF

ON

Fill Center

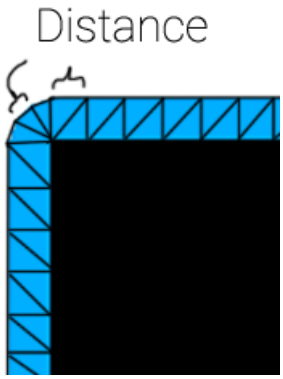
As the name suggests this will fill the center if enabled.
INFO: That's how the shadow element works ;-)

OFF

ON

Vertex Distance

This defines how far the vertices in the corners will be apart. The value is the distance between the vertices. If you have „Force Subdivision" enabled then this will also apply to the sides.



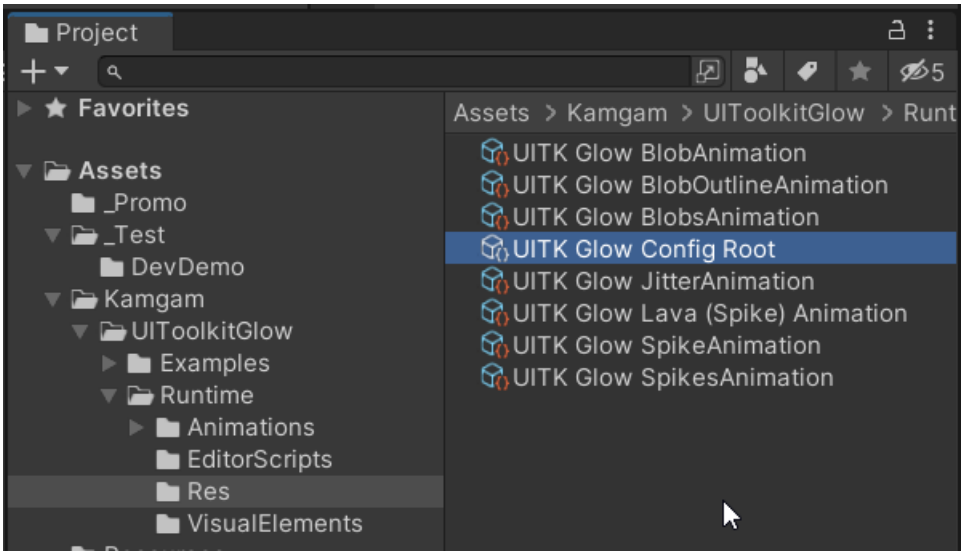
Layout First Child

This one only exists for the „Glow" and „Shadow" elements. If enabled then the first child of the elements will be set to grow and width/height 100%. It will also match the border radii of the parent. This is done so you do not have to copy these values manually.

Animation Name

This is the string name of the animation you want to use. You can find the Animations in the Glow Config Root in the „Runtime/Res“ folder. If you do not want to use an animation then simply leave this empty.

Please refer to the „Animations“ section for more infos.



NOTICE: If you use animations the you will have to add the GlowDocument to your UI Document or else the Glow element will not be able to find the animation data.

GLOW: The non-destructive way

If you are bringing in the glow asset into an already existing UI then you should consider this approach since it allows you to add a glow to any EXISTING element without the need for a custom control (like „Glow" or „Shadow").

NOTICE: This workflow only works for glows and outlines. For shadows you (sadly) need to use the Shadow element. There is no non-destructive workflow for shadows yet. As soon as Unity allows us to generate meshes behind the original mesh shadow support will be added.

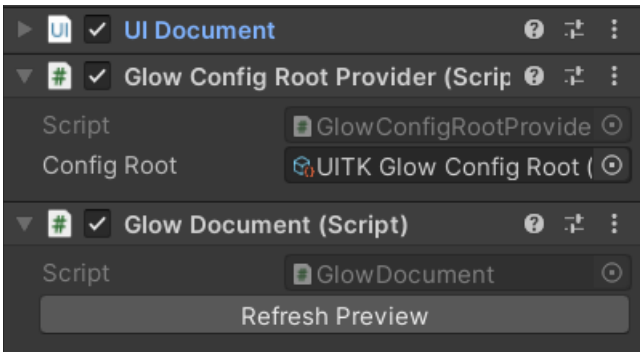
To add the glow in a non-destructive way requires you to do two things:

1) Add the GlowDocument to your UI Document.
2) Add the glow class to your elements.

1) Add the GlowDocument to your UI Document

Todo this first add the **GlowDocument** component to your UI Document.

You'll have to do this for any UI Document that should support non-destructive glow.



You will notice the GlowDocument automatically adds a **GlowConfigRootProvider**.

The GlowConfigRootProvider is just a wrapper around a GlowConfigRoot scriptable object. This is the data object where the non-destructive glow information is stored. It also contains a list of animation objects (more on that in the „Animations" section).

Why are GlowConfigRootProvider and GlowDocument not combined into one component?

The reason is order-of-execution. On the config provider we modify the execution order to ensure Awake() is called before UI Toolkit instantiates the UI Elements (which it does via the event system, which is set to execution order -1000 by default).

Execution order at application start:

1. GlowConfigRootProvider (-1001) – Provides the GlowConfigRoot

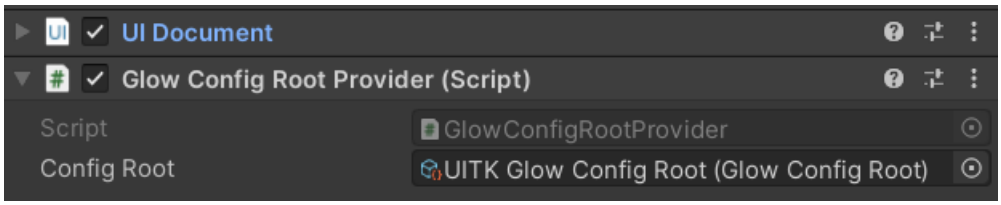
2. EventSystem (-1000) – UI Toolkit instantiates „Glow“ and „Shadow“ custom controls

3. GlowDocument (default) – Adds manipulators to the panel in OnEnable() to generate the meshes.

The provider needs to run BEFORE UI Toolkit but GlowDocument can only add manipulators after the UI elements have been created.

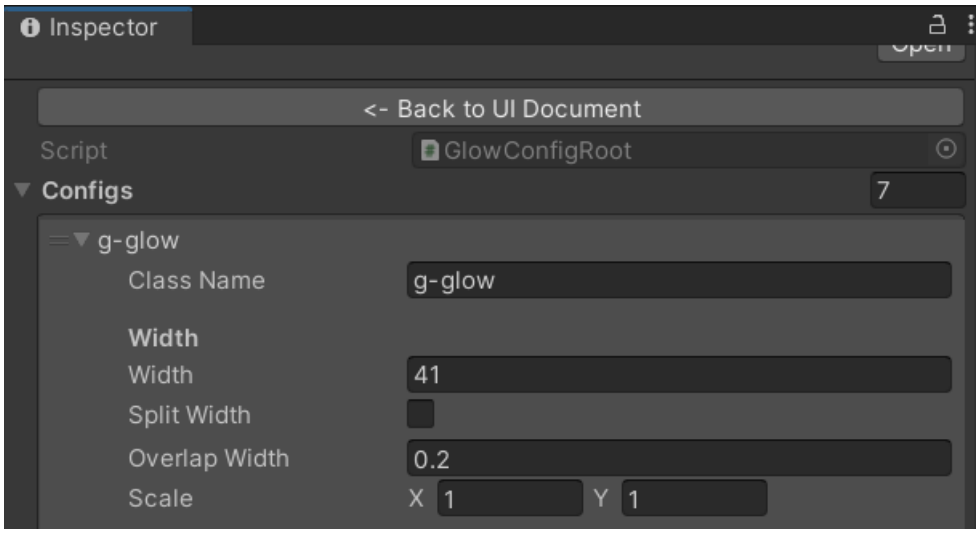
Glow Config Root

In the „Glow Document“ you will notice the „Config Root“. If you click on it you will see the configurations for ALL the flow classes. Here you can modify, add or remove glow classes.



NOTICE: You should only ever have ONE CONFIG ROOT asset in your project or else the Editor previews might get confused. At runtime it will use whatever config root your have linked in the provider.

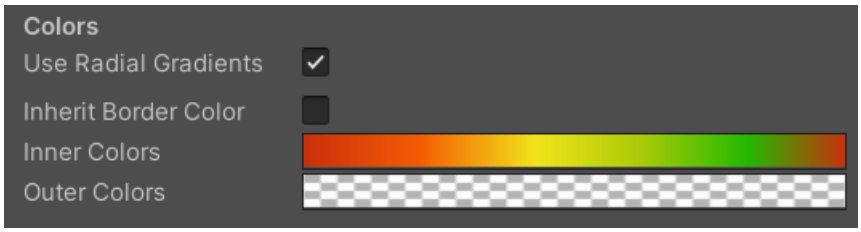
If you double click on the Config Root asset it will open.



In there you will see a list of „Configs“. Each of these stands for a glow class. To add a glow to an element you will have to add a USS class equal to name of the „Class Name“ property. These elements will then use that glow config.

HINT: There is a „Back to UI Document“ button at the top.

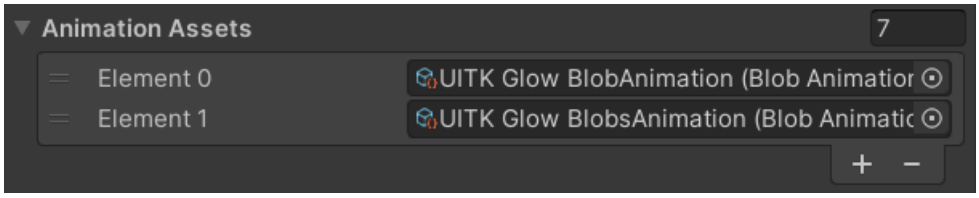
The properties on the glow config are the same as described in the „Glow“ custom controls above with the exception fo the „Radial Gradients“. If you enabled these you will get color gradient picker for the inner and outer colors.



With these you can add some more advanced coloring to your glow. The colors start at the top left and go around in a clockwise manner.

You may have also noticed the „Animation Assets“ section at the bottom. This is where the elements will look for animation to play if you have specified an animation name in the „Animation“ property. Here you can drag in all you custom animation assets.

INFO: Usually animation assets will add themselves automatically but it if you have an animation that is not playing then checking if it is in this list is one of the first things you should do.



Then there is the „Use Runtime Copy“ checkbox and the „Refresh Preview“ buttons.

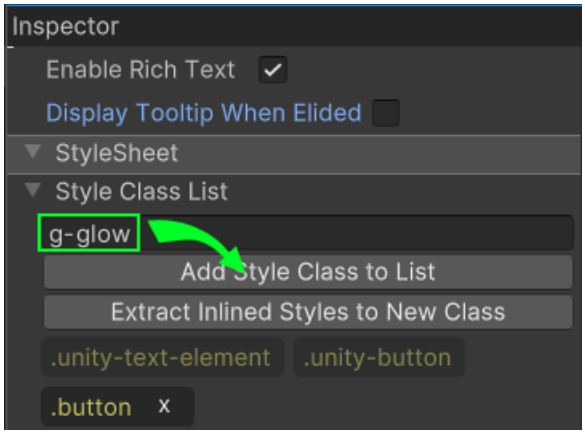
The „Use Runtime Copy“ will ensure each glow config is copied when used so you can not accidentally edit it via code at runtime in the editor.

The „Refresh Preview“ button exists to convince Unity to actually refresh the UI Builder preview of the glows. Usually it should not be necessary to press this button but if you are adding, removing or renaming classes in the list then this might come in handy.

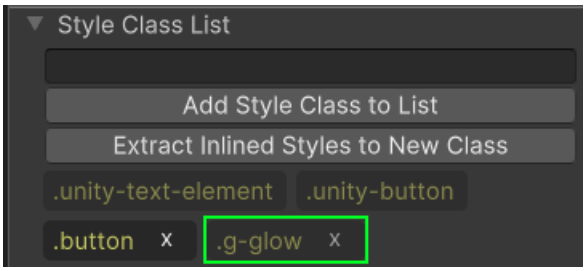
2) Add the glow class to your element

Once you have added and configured the GlowDocument you can use the glow classes on your elements.

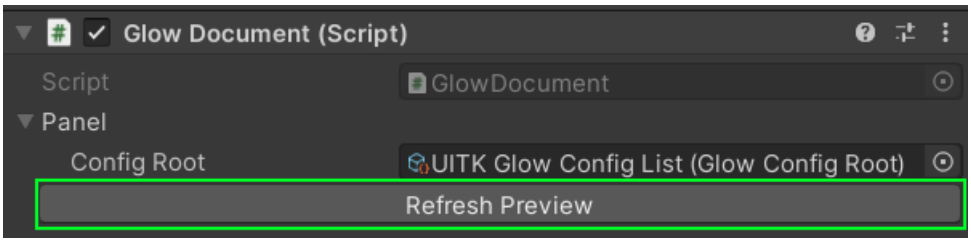
Simply add them to your existing style classes and you are ready to go.



After the glow class has been added it will already work at runtime BUT ..



.. to make it show up in the UI Builder you may have to press the „Refresh Preview" button.



Usually you'll have to do this only once per session.

As soon as Unity adds an event callback for class changes to the editor this will be automated (let's hope Unity does add one soon).

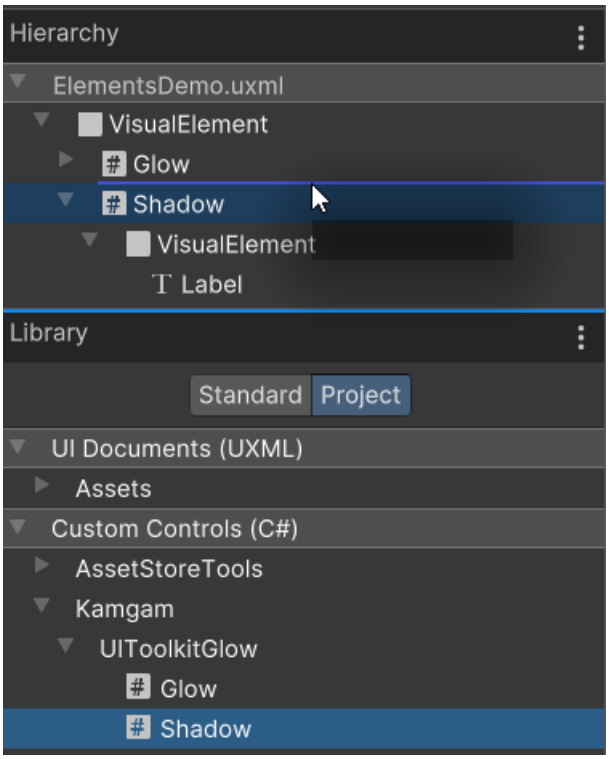
USS Attributes

You can configure most of the attributes via USS too. Check out the „GlowStylesDemo.uss“ in the project. It looks like this:

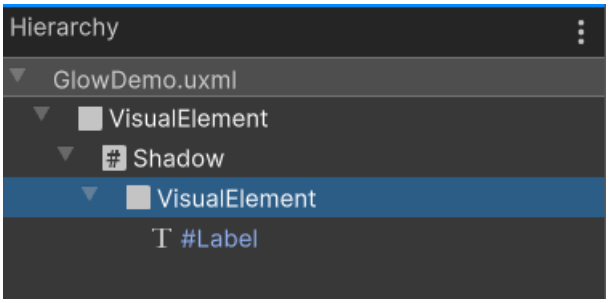
```
.g-glow
{
    // These are all the supprted custom styles für glows. You can also use
    // them on shadows.
    // Notice that none of the values have a dimension (no 'px').

    --glow-width: 60;                /* float */
    --glow-overlap-width: -4;        /* float */

    --glow-split-width: false;       /* bool */
    --glow-width-left: 10;           /* float */
    --glow-width-top: 30;            /* float */
    --glow-width-right: 60;          /* float */
    --glow-width-bottom: 90;         /* float */

    --glow-offset-x: 10;             /* float */
    --glow-offset-y: 30;             /* float */

    --glow-offset-everything: false; /* bool */

    --glow-scale-x: 1;               /* float */
    --glow-scale-y: 1;               /* float */

    --glow-inner-color: red;                /* color */
    --glow-outer-color: rgba(255,255,0,0);  /* color */

    --glow-inherit-border-colors: false;  /* bool */
    --glow-force-subdivision: false;      /* bool */
    --glow-preserve-hard-corners: false;  /* bool */
    --glow-fill-center: false;            /* bool */

    --glow-vertext-distance: 15;        /* float */
}
```

SHADOW: Using the Shadow element

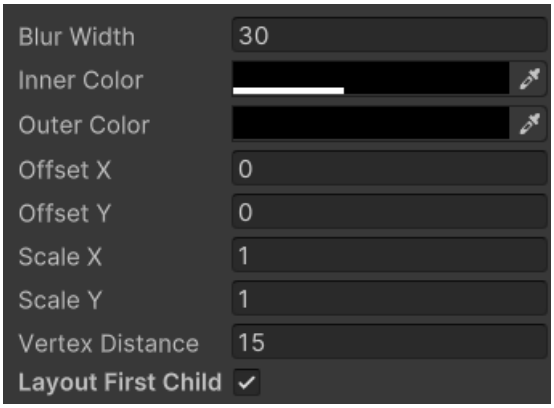
Currently only way to use the shadow effect is to wrap your elements in a Shadow control. You can find it in the **UI Builder Library** under **Kamgam > UIToolkitGlow > Shadow**.



NOTICE: You should not add content to the shadow element itself but instead add a child (visual element) and then add the content to that child.



HINT: Enable the „Layout First Child“ property.



Blur Width

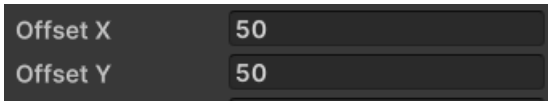
The width of the blur (here 10 and 30).

Inner / Outer Color

Just like the glow you can choose and inner and outer color.

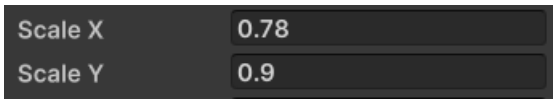
Offset X / Y

Moves the shadow ond the x or y axis.



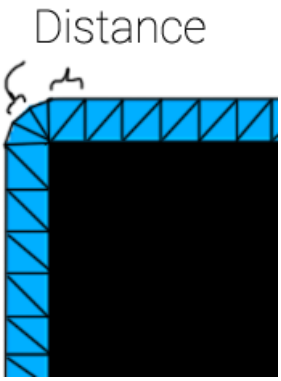
Scale X / Y

Scales the shadow on the x or y axis.



Vertex Distance

This defines how far the vertices in the corners will be apart. The value is the distance between the vertices.



Layout First Child

This one only exists for the „Glow“ and „Shadow“ elements. If enabled then the first child of the elements will be set to grow and width/height 100%. It will also match the border radii of the parent. This is done so you do not have to copy these values manually.

Animations

Take a look at the „AnimationDemo.cs“ file in the Examples. It contais a simple vertex animation.

NOTICE: Since we can not (yet) use custom materials in UI Toolkit it is also difficult use custom shaders. That's why these animations are done on the CPU. **Use animations with caution. One or two animations at the same time won't matter much but dozens certainly will.**

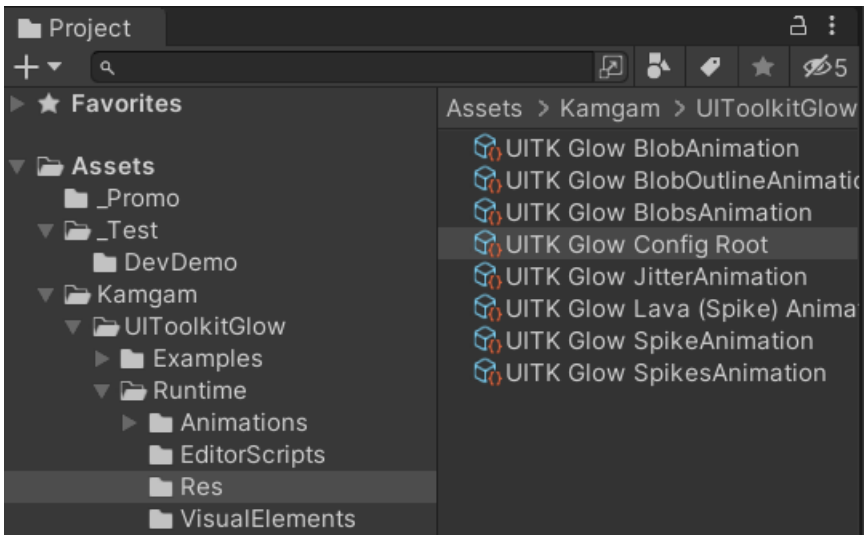
There are two ways to add an animation:

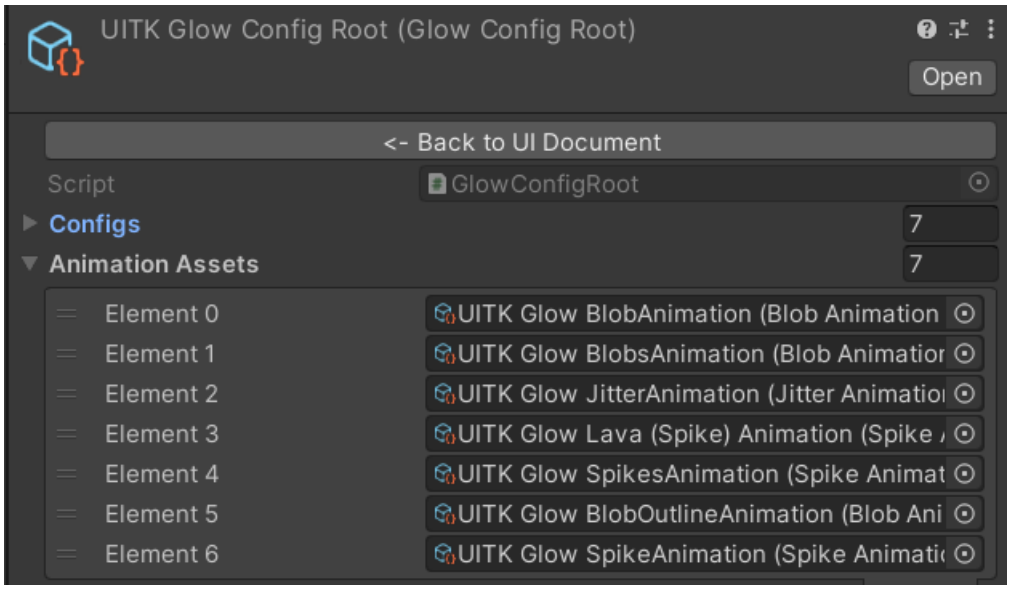
Either you use (or create) an **animation asset** OR you do some coding in a **MonoBehaviour**.

Technically there is a third way by using an Animation object directly but that's just the animation asset way without the asset.

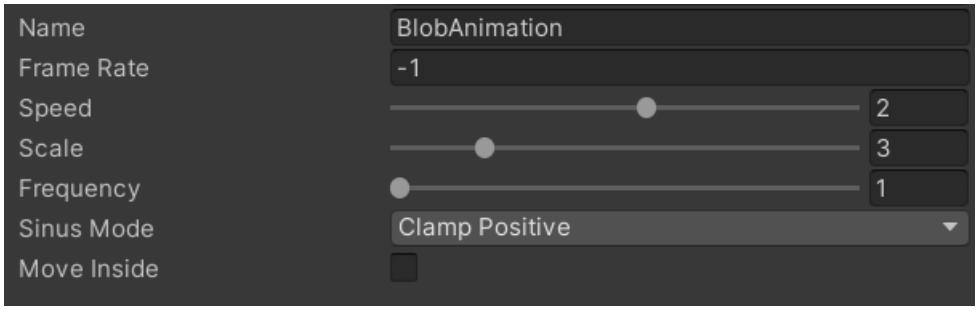
Animation Assets

There are some premade animation assets. Those are used in the demos. You can find them under Assets/Kamgam/UIToolkitGlow/Runtime/Res/.





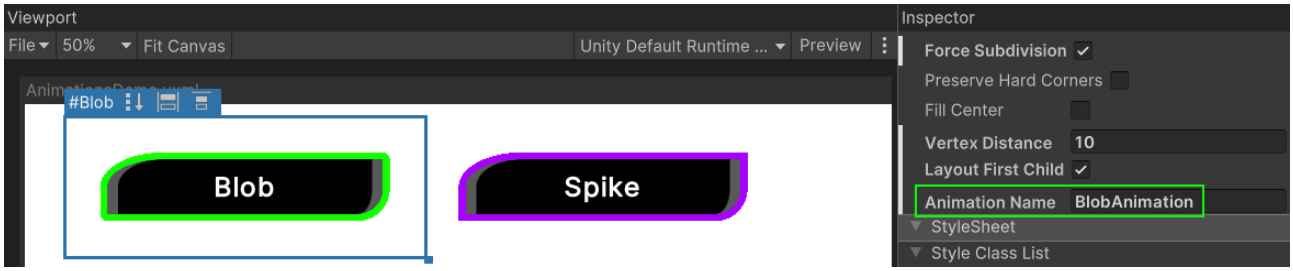
An animation asset is just a wrapper around an animation object that exposes some parameters in a Scriptable object. The advantage of this is that you can copy and paste these objects and thus reuse them easily.



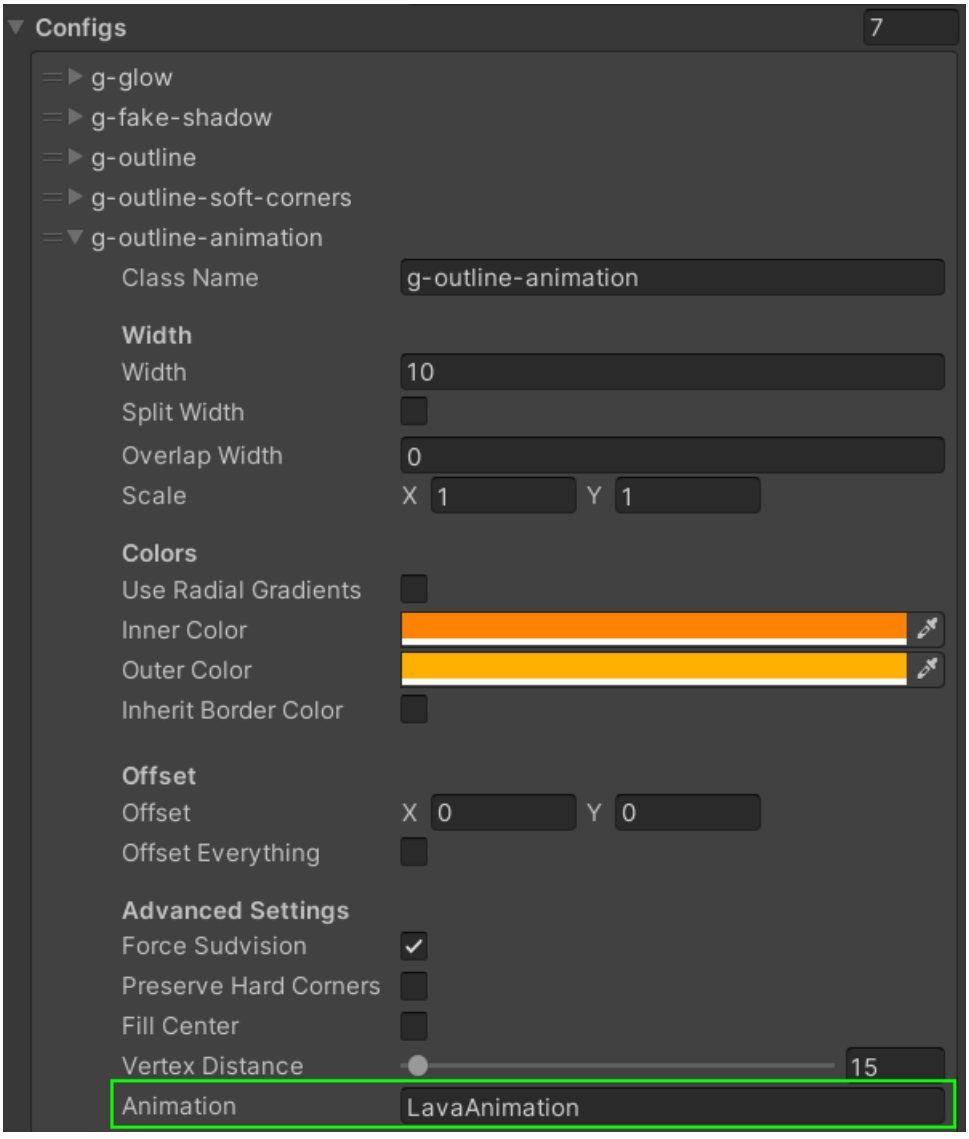
Animations are always referenced by NAME. In the example above the name is „BlobAnimation".

NOTICE: If an animation is not in the list of animation assets then it will not be found.

If you check the animations example you will notice that the animations are referenced by name either on the Glow element ..



.. or on the config.



If you want to make your own animation assets then please take a look at the code in the examples. It's best to just copy one of them and then add in your own code.

HINT: You do not have to use animation assets. You can also just add Animation via code.

Animations via MonoBehaviour

To do an animation via custom code on a MonoBehaviour you will have to:

1) Get the manipulator

2) Register to OnBeforeMeshWrite

3) Pump the update loop via MarkDirtyAnimation() to trigger OnBeforeMeshWrite every frame

4) Modify the vertices in OnBeforeMeshWrite

Example:

```
protected GlowManipulator _manipulator;

public void OnEnable()
{
    GlowDoc.RegisterOnEnable(OnGlowEnabled);
}

public void OnGlowEnabled()
{
    var e = GlowDoc.Document.rootVisualElement.Q<VisualElement>(ElementName);
    _manipulator = GlowPanel.GetManipulator(e);
    _manipulator.OnBeforeMeshWrite -= updateMesh;
    _manipulator.OnBeforeMeshWrite += updateMesh;
}

public void Update()
{
    // Here we mark the element for a repaint with animation.
    // This triggers the mesh generation cycle: OnBeforeMeshWrite -> updateMesh.
    _manipulator.MarkDirtyAnimation();

    _progress += Time.deltaTime * Speed;
}

protected void updateMesh(
    GlowManipulator manipulator,
    List<Vertex> vertices,
    List<ushort> triangles,
    List<ushort> outerIndices,
    List<ushort> innerIndices,
    Dictionary<ushort, ushort> outerToInnerIndices)
{
    // Some code to modify the „vertices“ list.
    // Look into the existing Animations for reference code.
}
```

Things to be aware of

Don't use „GlowPanel.GetManipulator(..)" in Awake() or OnEnable()

The reason is that the GlowDocument adds the glow manipulators in OnEnable().

Awake() and OnEnable() may be called BEFORE the GlowDocument OnEnable(). So it can happen that the manipulators are not added yet (GetManipulator() will always return null).

If you want to use GetManipulator() then use it in Start() or Update() – OR – use the „`RegisterOnEnable`" method of the GlowDocument. It is called right after the regular OnEnable(). Like this:

```
public void Awake()
{
    var glowDoc = GetComponent<GlowDocument>();
    glowDoc.RegisterOnEnable(OnLateEnable);
}

private void OnLateEnable()
{
    var element = doc.rootVisualElement.Q<VisualElement>(name: ElementName);
    var manipulator = GlowPanel.GetManipulator(element);

    Debug.Log(manipulator); // Now it is ensured this is found.
}
```

Don't use too many animations

Since we can not (yet) use custom materials in UI Toolkit it is also difficult use custom shaders. That's why these vertex animations are done on the CPU. Use animations with caution. One or two animations at the same time won't matter much but dozens certainly will.

Avoid animating the config values

Changing the config values of a glow will update any (non-destructive) glow that is using the config. However, this will trigger a complete regeneration of the glow mesh, which is expensive. If you want to animate the vertex positions or the vertext colors then please use a animation.

While animations should be used with caution too they are more performant than regeneration the mesh every frame.

However, having said all that. It usually is no problem if you have only a few elements with animations or config changes per frame.

Execution Order (why we need a GlowConfigRootProvider)

The provider is just a wrapper around a glow config object so we can use it in UI Toolkit custom elements and on GlowDocuments. On the provider we modify the execution order to ensure Awake() is called before UI Toolkit instantiates the UI Elements (which it does via the event system, which is set to execution order -1000 by default).

Execution order at application start:

1. GlowConfigRootProvider (-1001) – Provides the GlowConfigRoot.

2. EventSystem (-1000) – UI Toolkit instantiates Glow and Shadow controls.

3. GlowDocument (default) – Adds manipulators to the panel in OnEnable() to generate the meshes.

Frequently Asked Questions

Here are some common issues that have been reported.

If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

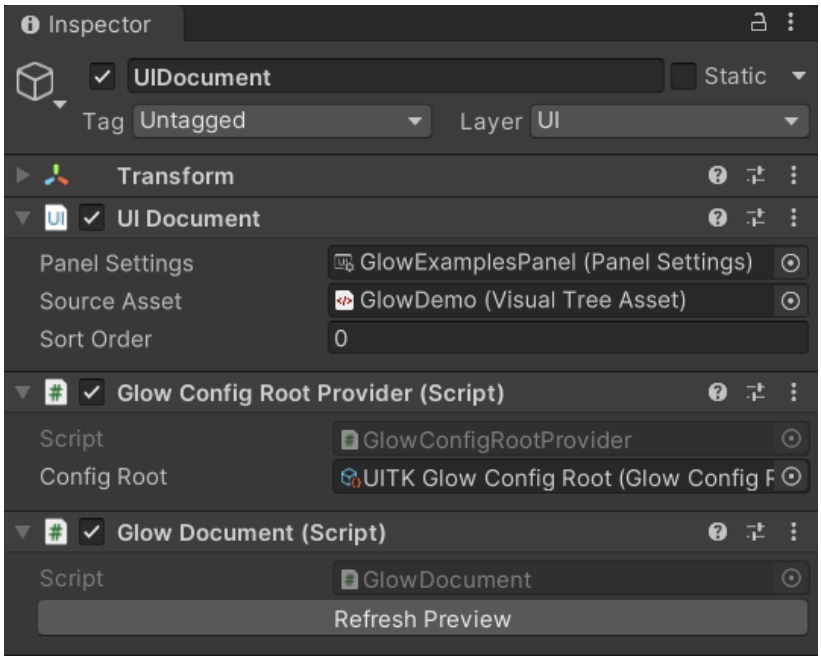
Keep in mind, UI Toolkit as a whole it is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

I have added a glow class to my element but there is no glow in play-mode (but it shows in edit-mode, how strange?!?).

The most common cause of this is a missing GlowDocument. **Please make sure you have a GlowDocument component added to your UI Document.**

Why? → In order for the glow to show up at runtime some code has to run on the UI Document. That code is inside the GlowDocument component.

INFO: There is some editor code that ensures the glow is shown even if you are not in play mode. That code searches for the glow configs in the whole asset database. That's why you will almost always see the glow in edit-mode. However, at runtime there is no asset database so you need to add a GlowDocument to provide the glow configs to the UI Document.



INFO: If you use the „Shadow" or „Glow" custom controls from the UI Builder Library then you do not need a GlowDocument (except if you want to use animations, then you'll always need it).

Changes to the config asset are not updated in the game view while animating

Sadly that's a know limitation at the moment. Simply resize the game view a little and it will refresh. You may also want to hit that „Refresh Preview" button every now and then.

Why are the edges not anti-aliased?

Unity does not (yet) support anti aliasing for user generated meshes in UI Toolkit. As soon as they add it the edges should be smooth. Sadly there is no ETA yet from Unity as to when they will add it.

„Inherit Border Colors" gives strange results if border colors are not all the same.

It may look like this:

This is due to some shared vertices. While sharing vertices is efficient for rendering it is not so nice for blending colors around corners.

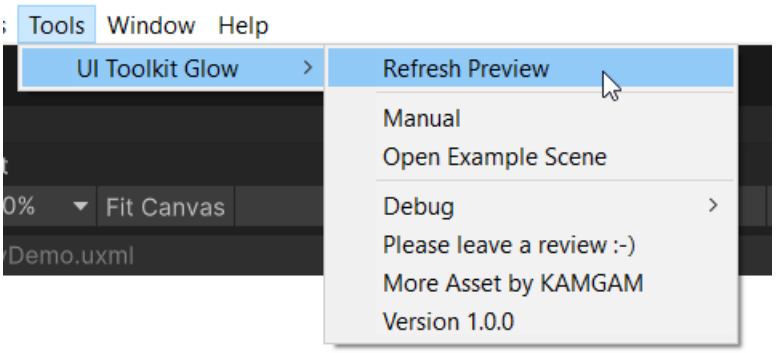
At the moment this is considered an edge case. I may look into this at some point in the future but don't get your hopes up, it's pretty low on the to-do list (maybe never).

Radial Colors can not be set in the UI Builder

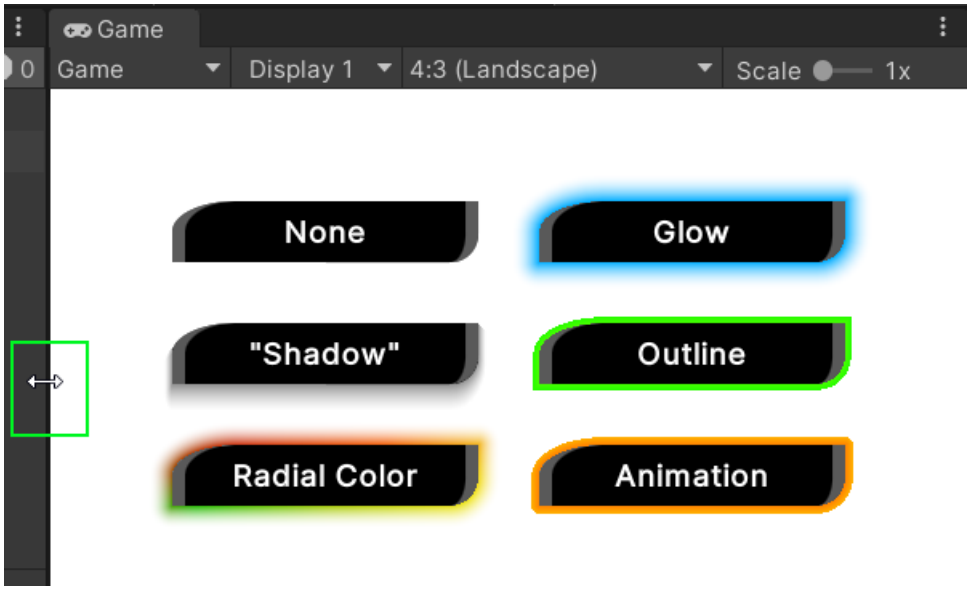
Sadly this is an issue that needs to be resolved (supported) by Unity. The UI Builder is pretty limited in terms of what it can do with attributes. It is getting better in newer Unity versions (2023+) so maybe in the future this will be doable.

The Preview in the UI Builder does not refresh

This is under investigation. In some Unity versions the auto refresh is sometimes cancelled. To restart it please call „Tools > UI Toolkit Glow > Refresh Preview".



Also in Unity 2021 the game view sometimes needs a little convincing to update the UI Elements. To do this simply resize it a little.



Btw.: If anyone knows how to make the game view refresh the UI then please let me know (I have exhausted the abvious solutions like Repaint(), RepaintAllViews(), queue play, ...).

Why are animations not previewed in the UI Builder or the Game View?

It simply would be too much load for the editor. Usually the UI Builder and the Game View are only updated if needed. Forcing animations on them would mean they would have to update 60 times per second which would certainly slow down your editor. It's not only the animation that would be redrawn but the whole window.

The „.glowAnimation“ property of the „Glow“ element is null in OnEnable()

If you try to use code like below you will notice that this throws a NullReferenceException.

```
public void OnEnable()
{
    var element = Document.rootVisualElement.Q<Glow>("Animation");
    element.animation.FrameRate = 5;
}
```

The reason is that to create the animation (and thus set the „glowAnimation“) the GlowDocument and (in there) the glow animations have to be loaded. This is done in OnEnable(). You can avoid this by using the GlowDocumet.RegisterOnEnable() method to register a custom onEnable method – OR – you use Start() instead of OnEnable().

```
public void Start()
{
    var element = Document.rootVisualElement.Q<Glow>("Animation");
    element.animation.FrameRate = 5;
}
```