Flexbox 4 Unity - v3.0 *(June 2020)*

Good choice! Flexbox will make you faster at building GUI/UI, and give you more responsive, adaptive designs that flow easily and layout more cleanly.

Table of Contents

Introduction

If you get stuck

You can get help on the official forum, or via Discord:

forum.unity.com

discord.gg

There are also tutorials and guides posted on the main website:

flexbox4unity.com

How does it work?

A GameObject can have a FlexContainer component, a FlexItem component, or both. You can mix and match Flexbox layout with Unity components (Button, Image, etc).

FlexItem

The FlexItem controls the size of itself – e.g. is it 400 pixels? or 50% of its parent? … does it have a maximum width? … does it have a minimum height? etc.

FlexContainer

A FlexContainer controls the layout of its children – e.g. are they positioned horizontally or vertically? … are they centered? Or left-aligned … or right-aligned? etc.

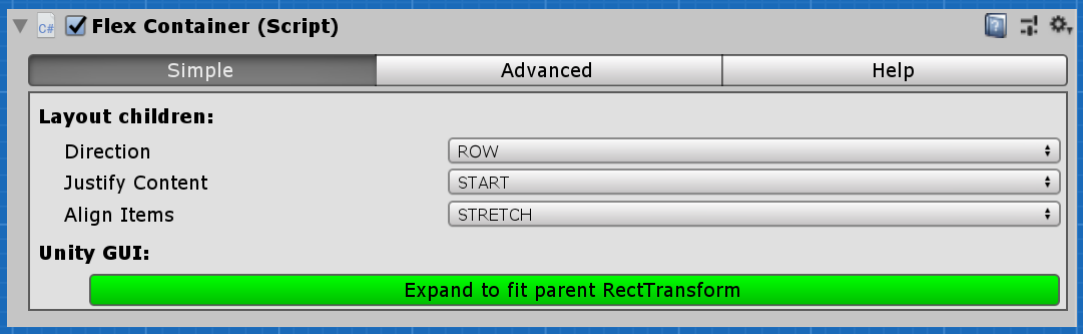
Getting Started

(also available online: http://flexbox4unity.com/2020/06/05/guide-using-flexbox-in-unity-2020/)

The minimum you need:

- a Unity Canvas
- a GameObject with a FlexContainer

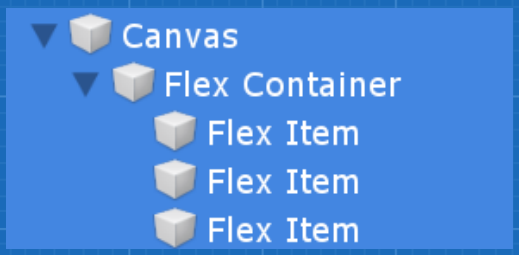
Usually you want your UI to occupy the full Canvas, and auto-resize whenever the Canvas resizes – it will detect if this is not the case, and offer you a button to automatically resize itself:



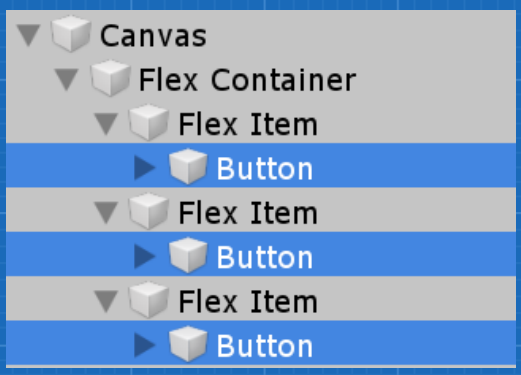
The simplest Flexbox UI

A very basic UI is a menu bar containing 3 buttons. To make Flexbox position and size the buttons, we need three FlexItems (one for each button), and we need to add a Unity Button to each FlexItem – that's it! Nothing else is needed.

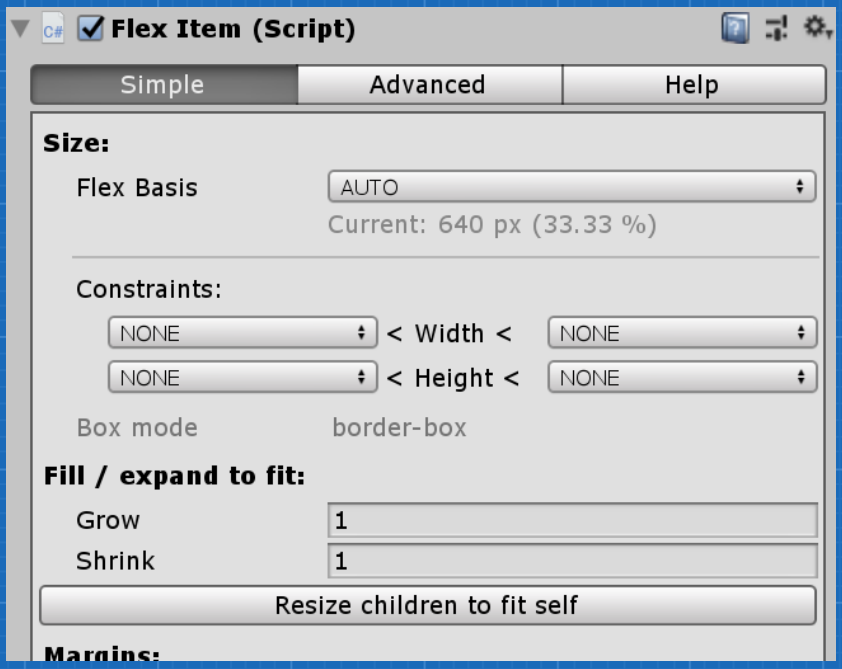
So, create three new GameObjects as children of your FlexContainer, with a FlexItem on each. You can do this quickly by right-clicking the FlexContainer in the Hierarchy view, and selecting "Flexbox > FlexItem" (it will create the GO and attach the FlexItem component).



Next, add a UI Button to each of the new GameObjects. You can use Unity's built-in shortcut for this: right click a GameObject in Hierarchy, and select "UI > Button".



Finally: when you add a UI Button, Unity makes it a fixed size, but we want Flexbox to control the size. You can do this manually using the RectTransform – but if you select each FlexItem, you will see it has auto-detected the child Button, and noticed it's not being auto-sized. Click the button "Resize children to fit self" to have it take control of the Button and autosize it from now on.



By default, Flexbox assumes you want objects to take up as much space as possible. If we had put Images in, that would be perfect. But for buttons, we want them to use a sensible size/shape. Flexbox4Unity has built-in autosizing for all core UnityUI elements, so all you need to do is tell it "stop maximizing everything in this container".

Select the FlexContainer, and change the "Align Items" setting to anything except "Stretch".

That's it. You're done! Flexbox was designed to automatically default to sensible values for everything, so it's already positioned and laid out all your objects.

CSS: Grow/Shrink

The power of Flexbox: Grow/Shrink

One of the most important features of Flexbox is “grow”. Unlike Unity’s complicated resizing system, you have a single number for each item that controls how it reacts to resizing. This may sound too simple – but most of the code in Flexbox4Unity (thousands of lines of source code) exists to make this work automatically.

When there is more space than the FlexItems need, Flexbox looks at their “grow” values to decide what to do next.

Simple growing:

If any item has a “grow” of 0, then it won’t grow at all: it will remain its default size (which might be zero, if the FlexItem has no button/text/image). If all the items have a value of 1, then they will each receive an equal share of the spare space, and grow to fit.

Complex growing:

But if they have a value more or less than 1, then they will grow by different amounts relative to their different grow values. Without Flexbox, Unity is incapable of doing this: UnityUI can only resize objects locally, so all your different buttons end up ignoring each other. But in Flexbox, each button (or image, or textfield, etc) is collaboratively working together to allocate space between them, and to maintain relative ratios/sizes.

Example

For instance, give the middle button a “grow” of 2, and the left and right buttons a “grow” of 1. The middle button will take up half the spare space (2 / (2 + 1 + 1)), and the left and right buttons will get one quarter each (1 / (2 + 1 + 1)).

Shrink

Shrink does the same, but is only used when the FlexContainer is too small to contain the FlexItems. A Shrink of 0 means the item won’t shrink – and if none of them shrink, they will overflow the edges.

(Incidentally: this is how Scrollview is implemented with FlexBox: you set all the rows in your scrolling area to shrink=0, and then Flexbox detects this and creates a large scrollable area. You still need to create a Unity ScrollView, but it will automatically size and position its scrollbars because of this).

CSS: Padding and Margins

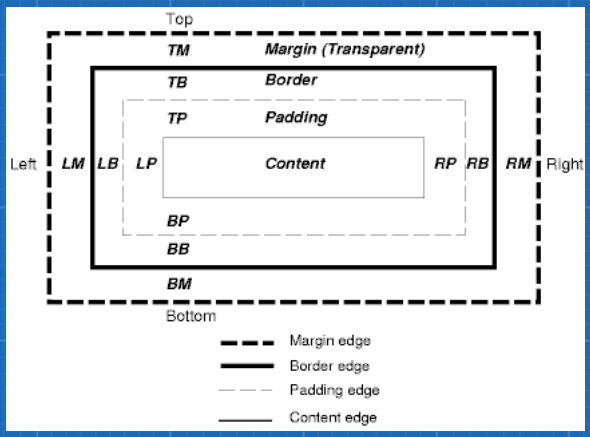
The power of Flexbox is its simplicity. The flex system makes it very easy for us to control the position and size of everything in our GUI (and in most cases we can leave Flexbox at its default settings, where it intelligently does the the layout for us).

But to get really good-looking GUIs, you need to control the empty space as well: the padding, the margins, the indentation.

CSS-3's Flexbox is tightly integrated with CSS's margins/padding system, so I have also implemented that part of CSS – you cannot fully run Flexbox without giving it detailed information about the margins/padding of every item (and Unity does not support margins/padding itself, so we have to add it).

Padding vs Margins

Here is the official image from CSS-3 showing how they relate to each other:



To recap:

- Padding – extra space INSIDE the UI element, but AROUND/OUTSIDE its main content
- Margin – extra space OUTSIDE the UI element

The main differences in practice:

- If a UI element has text – e.g. a button, with text – then Padding increases the space around the text (but inside the button image), while Margin increases the space outside the button (transparent background).
- The background of an element covers the Padding but doesn't cover the Margin

Using Margin with Flexbox4Unity

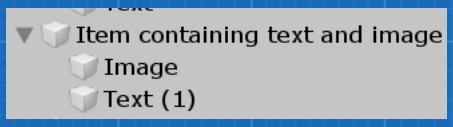
Margins **can be applied by any FlexItem**, and affect any attached UI elements / RectTransforms, and any children.

Margins tell the parent FlexContainer to put empty space around the FlexItem and its children.

Using Padding with Flexbox4Unity

Padding **can only be applied by a FlexItem that is also a FlexContainer**, and only affects the child UI elements / RectTransforms.

The classic setup is: a GameObject with FlexItem, and a child GameObject with Text/Button/etc, with the child RectTransform expanded to fit the full size of the parent. Each child also needs a FlexItem component (so that the parent FlexContainer can control its size), but you usually leave these at default settings and/or set them to “basis = CONTENT” (or basis = AUTO).

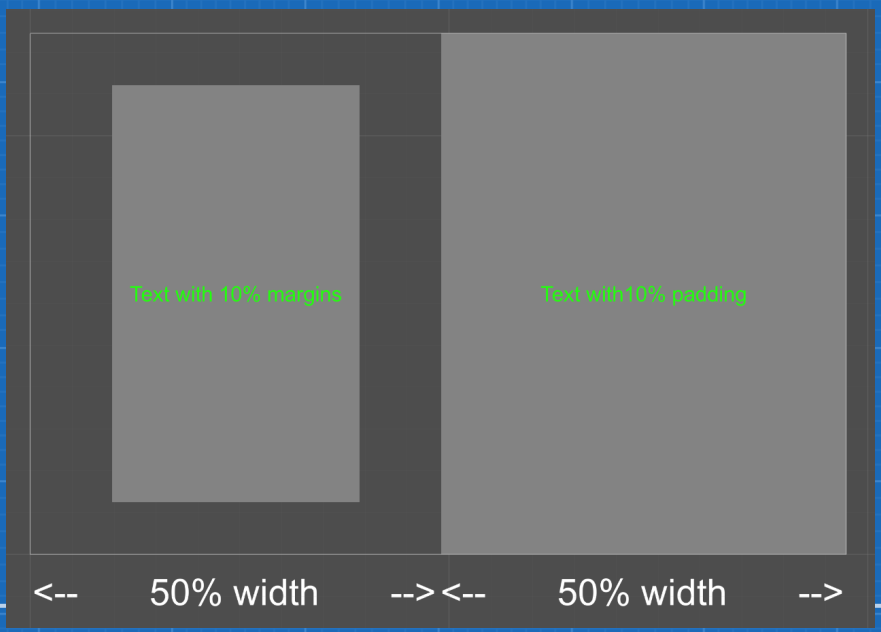


Note: if you attach the FlexItem directly to a UI element (e.g. a Button or an InputField), it will be ignored – this is because Unity’s UI system doesn’t support Padding internally, and only Flexbox4Unity understands how to use it. The UnityUI built-in components cannot handle padding on their own.

(this is why you need to make the item with padding both a FlexItem and a FlexContainer – it allows Flexbox4Unity to implement the missing features from UnityUI).

DemoScene: “margins and padding”:

This scene contains a simple example of two identical FlexItems, but one using margins, the other using padding.



Flexbox4Unity: Working with UnityUI

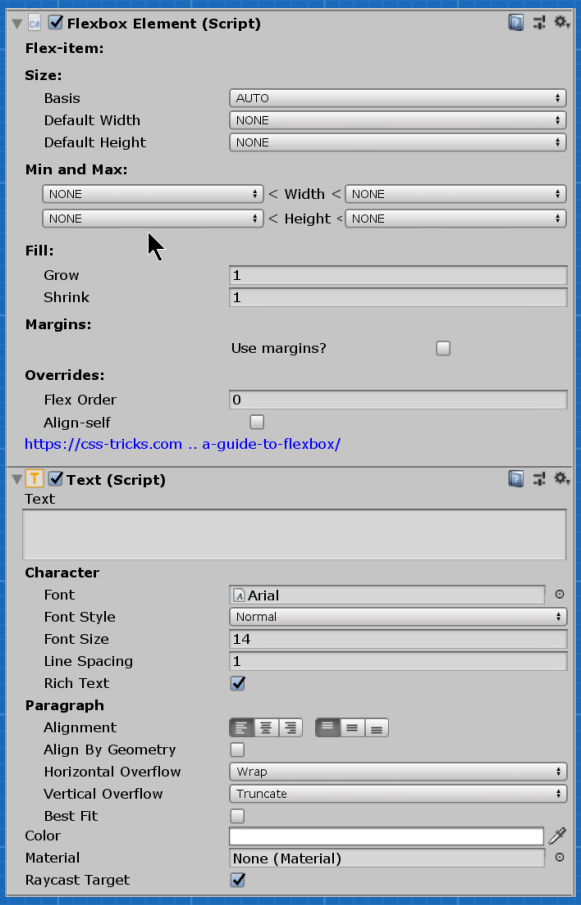
Flexbox4Unity is fully integrated with UnityUI. This means that it automatically detects core UnityUI items (and even your custom UnityUI controls) and sizes them intelligently.

Example: using Flexbox to size a Unity Text

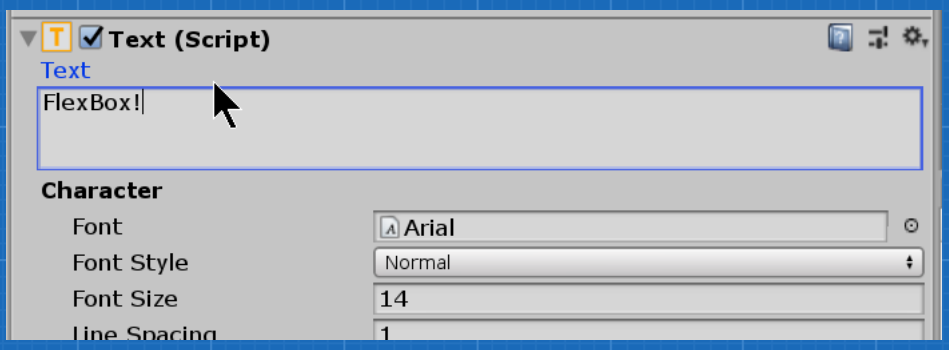
First, setup a basic scene:

1. Make sure you have a Canvas in your scene (create one if you don't already)
2. Create a FlexContainer as child of the Canvas
3. Create a FlexItem as child of the FlexContainer

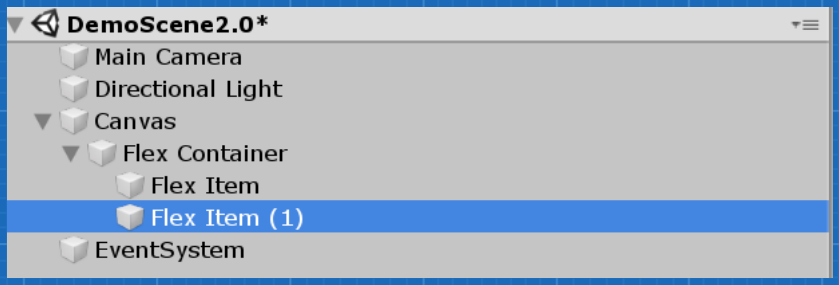
Now add a Text component to the flex item:



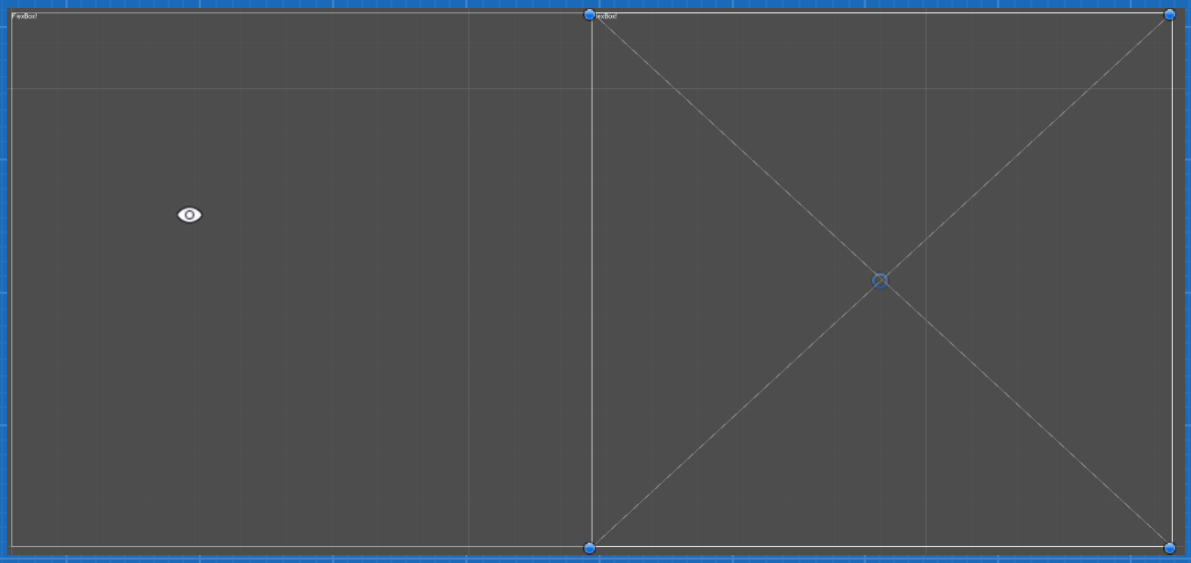
Edit the text to say something:



Select the FlexItem in hierarchy and duplicate it:



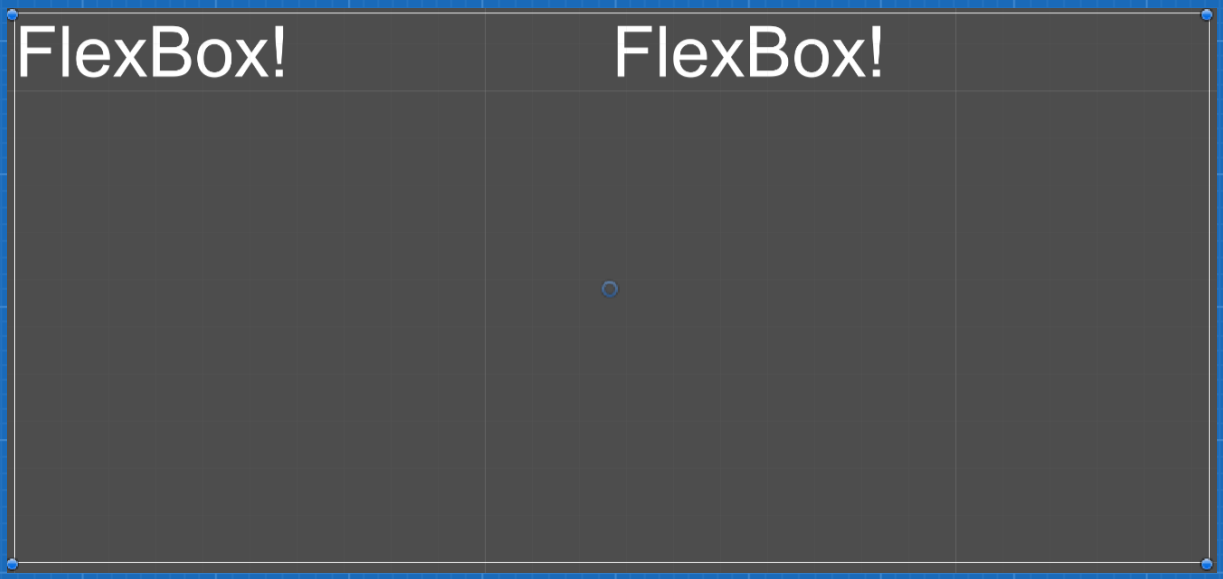
Flexbox4Unity will now automatically layout your two text items, and it should look something like this in the SceneView:



Edit the two Text components, and set:

1. Color = white
2. Font-size = 150

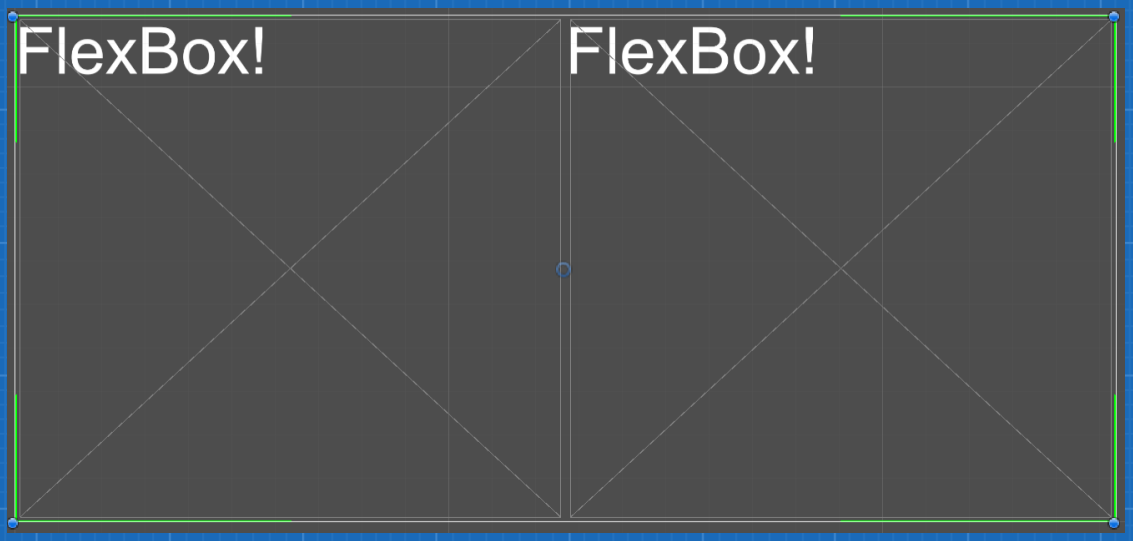
...and your SceneView should now look like this:



Flexbox4Unity: Live Previews

In the Hierarchy view, try selecting the main Flex Container you created. This will show the live-preview of your Flexbox hierarchy, making it easier to rapidly edit and debug complex layouts.

Basic Preview

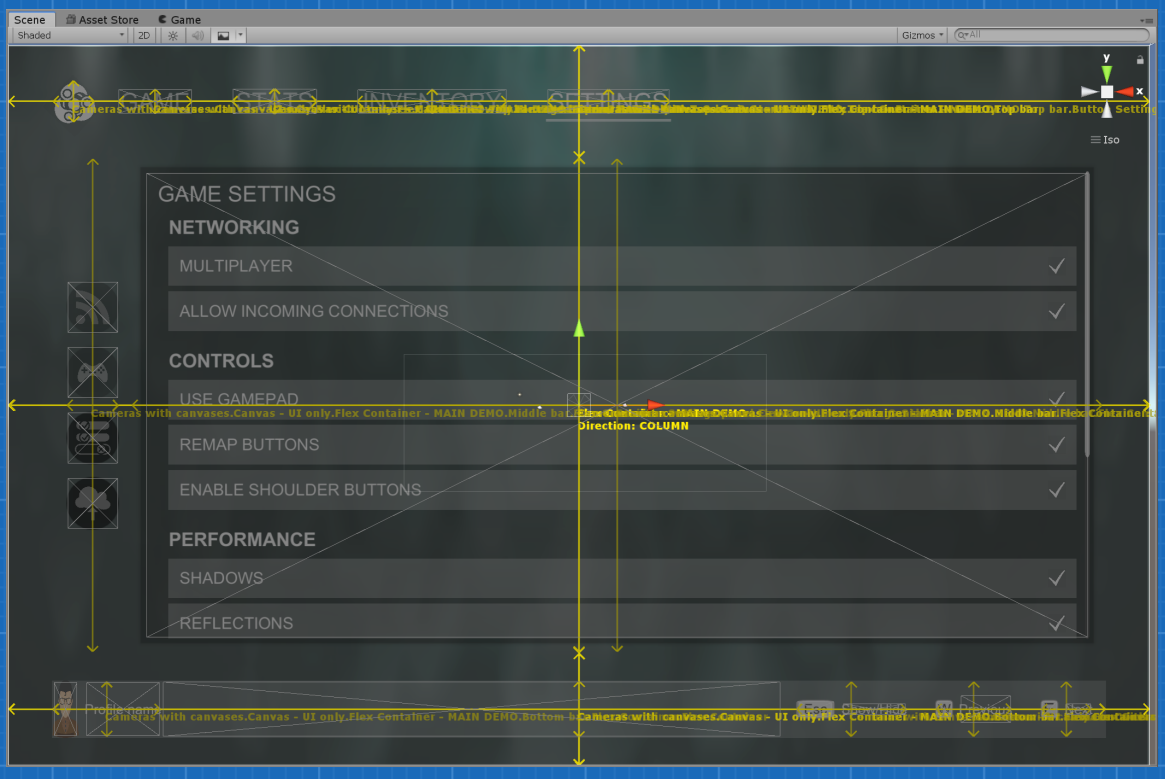


The grey crossing lines show where child FlexItem objects are.

The green corner lines show where nested FlexContainer objects are (try adding child FlexContainers, and you'll see additional sets of green-corners appear).

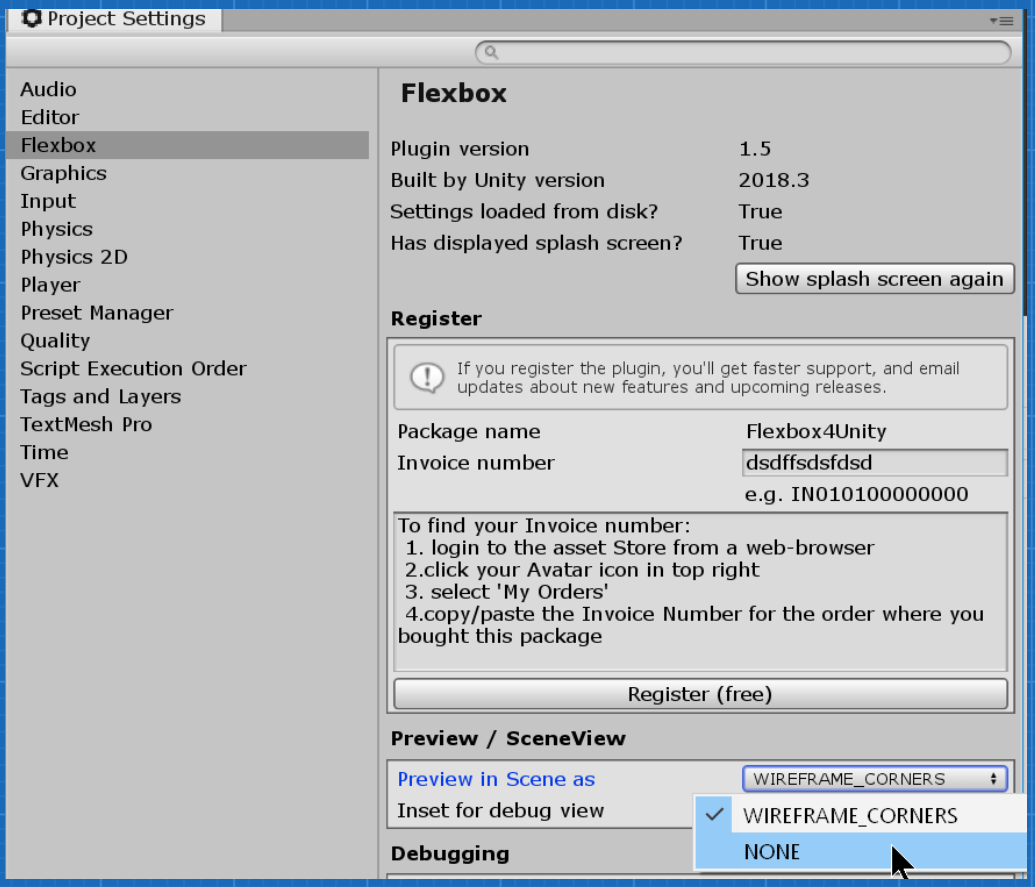
New for Flexbox v3: Advanced Preview

The advanced preview shows different info, focusing on the relative layout positions of each item. This is a work-in-progress, currently available to alpha-testers and beta-testers.



Note: you can disable preview mode at any time by going to your Unity Project Settings:

Edit > Project Settings > Flexbox



Flexbox4Unity: Mixing and matching UnityUI with Flexbox

Embedding FlexContainers inside Unity's default GUI / RectTransforms

Use Unity's anchors and handles to position and control the flex-container however you want. e.g. using the RectTransform tool put the anchors in the four corners of screen, and drag the blue dots to the four corners, making it auto-size to fit your screen.

Unity will control the size and position of the parent container, and Flexbox4Unity will control layout inside the container.

Embedding UnityUI inside FlexContainers / FlexItems

You can use any default UnityUI control within Flexbox4Unity, and it will be automatically resized by Flexbox.

There are two main ways of doing this, with pros/cons of each. Both require a FlexItem component to control the layout.

Option 1: Attach a FlexItem component to a UnityUI GameObject

1. Select the gameobject with the FlexItem
2. (recommended): Set the flex-basis to "CONTENT"
3. (recommended): Set the flex-grow to "0"

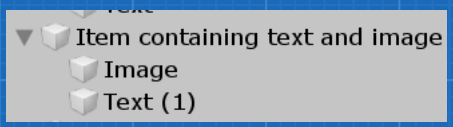
...the FlexItem will now intelligently resize itself to the best size for the UnityUI element (e.g. a Unity.UI.Button will try to wrap the text inside it, but will allow the text to flow to multiple lines if necessary).

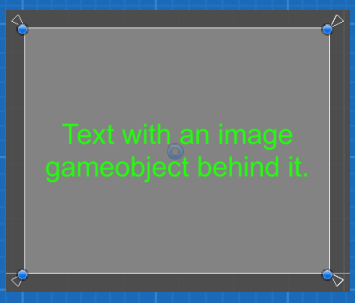
Option 2: Parent multiple UnityUI GameObjects inside a FlexItem GameObject

1. Select the gameobject with the FlexItem
2. (recommended): Set the flex-basis to "CONTENT"
3. (recommended): Set the flex-grow to "0"
4. Create a child object for each UnityUI item
5. Select each child object and drag the anchors + handles to auto-fill the parent

…in this case, the FlexItem will choose the most important UnityUI object among its children, and use that as the basis of the size. The order of importance is currently: Button > Toggle > Inputfield > Text > (others).

e.g. here is a Unity Image background with a Unity Text on top, both of them resized to always stay together, but automatically resizing and fitting the text:





Flexbox4Unity: Maintaining Aspect Ratios

ASPECT_FIT: fixed-aspect GUIs inside variable-aspect GUIs

The CSS-3 specification of FlexBox was designed for web browsers, and does not automatically adapt to different sized/shaped screens (CSS has a different mechanism for that – @screen – which you could implement in Unity but is much more complex to use).

The only major missing feature that we need in Games/Apps is control of the AspectRatio of parts of the GUI. We want this for mobile games, desktop games, embedded RenderTextures, etc. I have extended the CSS-3 Flexbox specification to include an extra value of flex-basis: "ASPECT_FIT".

To use ASPECT_FIT:

1. Create a FlexContainer (or choose an existing one)
2. Add a child object, with a FlexItem component
3. Set the child's flexBasis to "ASPECT_FIT"
4. In the field for flexBasis, type the aspect ratio as X/Y.
    1. e.g. to force 1:1 ratio, type "1"
    2. e.g. to force 3:2 ratio, type "1.5"
    3. e.g. to force 3:4 ratio, type "0.75"
5. The FlexItem will now always maintain that Aspect ratio, and will fit itself as large as possible within the parent FlexContainer.

Demo Scene: "**DemoScene – imacs**"

Flexbox4Unity changes vs CSS3

Removed/Not implemented: CSS-3 Anonymous FlexItems

CSS's "Anonymous FlexItems" make it impossible to fully integrate with Unity's UI / RectTransform / other UI libraries and plugins.

(In CSS, every child of a FlexContainer is automatically a FlexItem – you cannot have non-flexitems as children)

In CSS, the workaround is to use other CSS systems – e.g. “position: absolute” – to “un-flex” the automatically-flexed items; to keep things simple in Unity, I have instead removed this feature. In Unity/Flexbox4Unity, **only GameObjects that have a FlexItem component will ever be resized**.

(note: if a GameObject only has a FlexContainer component, **it will not be resized**; FlexContainers control how their children are resized, they never resize themselves)

Percentages are applied to parent width OR height

For historical reasons, CSS applies any “PERCENT” relative to the **width** (even if it’s a height!). Except … some parts of CSS don’t, and apply height to height, width to width!

This is far too confusing, and it makes layout much more difficult when designing GUIs, so I ignored the CSS specification. In Flexbox4Unity, **percent-widths apply to the width** and **percent-heights apply to the height**, always.

Experimental: flex-wrap

Flex-wrap is now included in the core layout algorithms, but to access it you need to use the ADVANCED mode (select a FlexContainer component, press the ADVANCED button, and one of the options lets you set the flex-wrap to “NONE” or “WRAP”).

In a future 3.x release, this will be moved to the BASIC tab.

Troubleshooting

… “It's not working! What do I do?!”

FAQ: Support forum & Discord

Forum:

https://forum.unity.com/threads/released-flexbox-fast-easy-layout-from-html-css-in-unity-2017-2018-2019.699749/

Discord:

https://discord.gg/umXJq4c

FAQ: My GUI scales weirdly when I change resolution

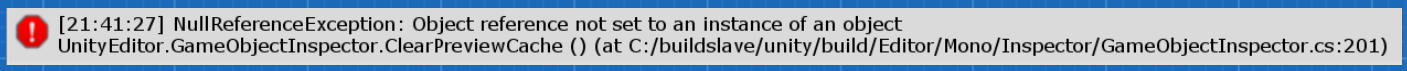
Unity version 5.x upwards requires that when you instantiate Prefabs and parent them, you use this code:

```
GameObject go = // anything here, e.g.: Instantiate( prefab );
go.transform.SetParent( parentTransform, false ); // this line is critical
```

Unity has some old bugs in their SetParent code that are only resolved using the extra, optional, flag “false”. They cannot fix them because old games depend on the old behaviour, so they recommend all new games use the new behaviours by sending “false” as a second parameter.

FAQ: “NullReferenceException: Object reference not set to an instance of an object”

If you see this every time you re-compile your project’s C# files:



...you need to upgrade your version of Unity. This is a bug in UnityEditor that they fixed, and Unity officially no longer supports any developers using the old version any more. The minimum supported by Unity is currently 2019.4 (LTS).

Recent changes

3.0: Major new layout algorithm, first Flex-templates and Flextensions

2.4: Improvements to VR support, new Experimental flex-wrap

- FIXED: player builds weren't exporting settings, were freezing layout on build
- Added auto-integration of UnityUI.RawImage

2.3: Complete re-write of the Layout engine (fixed many bugs + edge-cases)

2.2: Improved editing GUI

- Greatly improved the Inspector window for FlexItem and FlexContainer
- Added Help tab and "Advanced" inspector for direct access to all flex params

2.1: New layout-algorithms

- Adopted CSS3's names for classes and variables: FlexItem and FlexContainer
- Created modular, user-replaceable, Layout algorithm system

2.0: Big improvements for Unity 2018, Unity 2019

- Fixed: auto-refresh stopped working due to changes internally in Unity
- Added: Live-preview system to visualise complex layouts in Scene View
- Added: integrated Unity's new ProjectSettings system for storing and editing settings
- Added: free registration system for faster customer-support

1.5: Replaced UnityUI's slow algorithm with a custom fast one

- Improved: integration with Unity 2018.x editor versions

1.4: Added "padding" and improved auto-sized content

1.3: Performance optimizations

1.2: Greatly improved "margins" support

1.1: Fix Unity prefabs error, improve Aspect-Ratio Fill

1.0: First public release