UI Flex Layout

Documentation

by W7

1. Introduction

1.1 Component Overview

This UI Flex Layout package contains 3 main scripts to provide UI layout functionality:

- **UI Flex Container**

The container determines how items should be laid out. Items are **GameObjects** with the **UI Flex Item** component which are of direct descendant within the hierarchy.

- **UI Flex Item**

Supplies information for the container for it to determine its **RectTransform** position and size.

- **UI Flex Linker**

Controls size information of the **UI Flex Item** component when there's also an **UI Flex Container** component. Similar to a **Content Size Fitter**, this component takes information from all items within the container to determine the size of this item.

1.2 Dual Sizes

All size information are a combination of relative **percentage** and a constant in **pixels**. These are added together to form the final size. Most percentage fields are relative to the container's **content size**. Currently, width and height are relative to their their own axis, meaning percentage width is relative to the width, and likewise for height. There's currently no option to easily maintain aspect ratio.

Negative Input

Both relative percentage and constant in pixels can have negative inputs. Resulting final size cannot be negative when computing.

All Size Fields:

- Flex Basis **(Item)**
- Minimum Size **(Item)**
- Maximum Size **(Item)**
- Margin **(Item)**
- Minimum Gap **(Container)**
- Padding **(Container)** (relative to Transform size)

1.3 Padding and Margins

Padding

The **UI Flex Container** component has a padding field. Padding subtracts area from the container's transform size edges to form the **content size**. The padding field is subdivided into sizes for its 4 edges and a field that applies to all edges. **Percentages** are relative to the transform size. Percentages applied to top and bottom edges are relative to height. Left and right edges are relative width. Padding is shown with an orange outline gizmo for selected containers.

Margin

Margins can be set for **UI Flex Item** components. Similar to padding, this field is subdivided for each edge and a field that applies to all edges. Margins define required space around the item to be empty. Marginal percentages are relative to the item's container content size. Margins are shown with an yellow outline gizmo for selected items.

1.4 Item Dynamic Sizes

Flex Grow

The flex-grow value can be set for flex-items. By default flex-grow is disabled with a value of zero. When enabled by setting a positive value, items will grow in size when the container has more available space than the combined basis-sizes of its items. Items with higher growth value take more of the available space than items with a lesser growth value. Items only grow in the main direction of the container. Stretching in the cross-axis is not affected by the flex-grow value.

Flex Shrink

Flex-shrink is enabled by default with a value of one. Similar to flex-grow, shrinking only happens in the main container direction. Items shrink in size when the container's content size cannot fit all items with their basis-sizes. Different than flex-grow, the shrinking factor is multiplied by the basis size internally. This makes all items retain their relative sizes when shrinking. When items' shrink factor are equal, bigger objects shrink with bigger portions than smaller objects, such that all items reach zero size at the same time. Items with double the shrink rate will reach zero twice as fast compared to others.

When wrapping is enabled, it takes priority over shrinking. Shrinking then only works when there's just a single item in a line.

1.5 CSS' Online Guide

There's and extensive visual guide about CSS Flexbox which shows more visually how wrapping and alignments work. Please visit the following link to get a good sense of how this all works. Most functionality of this package is directly related to CCS Flexbox.

A Complete Guide to Flexbox
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

- Note: the **baseline** function is not available.

Minor Alignment Improvement

There's a minor difference in how **Align-Content** influences **Align-Items** alignments in this package. When **Align-Content** is not set to **Stretch**, line's cross-axis size is determined by the largest item in that line. **Align-Items** then aligns items within this line. You could have a centered content with items aligning to the bottom of that centered line. This could only be produced in CSS with wrapping enabled, in this package this is not required. Allowing the option for shrinking to still take effect.

In CSS, **Align-Content** behaves like **stretched** whenever wrapping is disabled, regardless of what it is set to. This package allows for more options regarding this, where **Align-Content** values create different layouts.

2. User Guide

2.1 Component fields

UI FLex Container

Field	Script Access	Description
Direction	*direction*	Main direction axis.
Reversed	*reversed*	Reverse main direction.
Wrap	*wrap*	Enables wrapping to new lines instead of shrinking items.
Wrap Reversed	*wrapReversed*	Reverses the wrap direction.
Justify Content	*justifyContent*	Alignment of items in main axis lines. Determines what to do with spacing when flex-grow is limited.
Spacing Distribution	*spacingDistribution*	Custom distribution for Justify Content
Align Content	*alignContent*	Behavior or Positioning of lines in the cross axis.
Align Items	*alignItems*	Item alignment in the cross axis within lines. Items can override their alignment with the Align-Self property.
Minimum Gap	*minimumGap*	Minimum required gap between items.
Padding	*padding*	Reduces Content Size inwards from edges.

UI Flex Item

Field	Script Access	Description
Order	*order*	Order of items within the container. Hierarchy order is maintained with equal order value.
Grow	*flexGrow*	Growth factor with respect to other items. Growth starts when there is more space than the Flex-Basis sizes in a line.
Shrink	*flexShrink*	Shrink Factor. Basis size is factored in to the final shrink factor. Shrinking starts when there is less space than the Flex-Basis sizes in a line.
Flex Basis	*flexBasis*	Base size. Determines when items should shrink or grow in the container.
Minimum Size	*flexMinimum* *flexMinimumEnabled*	Minimum Size.
Maximum Size	*flexMaximum* *flexMaximumEnabled*	Maximum Size.
Align Self	*alignSelfEnabled*	Override the containers Align-Items property.
Alignment	*alignSelf*	Self alignment value.
Margin	*margin*	Required spacing around this item.

UI Flex Linker

The linker component contains 3 check-boxes in the inspector for linking Flex Basis, Minimum Size and Maximum Size. Component not designed for scripting. Further information about this component is section 2.3.

2.2 Setting Up A Layout

Getting Started

To start creating a layout, add a **UI Flex Container** component to any **GameObject** within a canvas. Alternatively, you can right click in the hierarchy and find **UI > Flex Container** to create a new object with this component.

To add items to the container, either create a new **GameObject** with the **UI Flex Item** component attached, or find **UI > Flex Item** in the hierarchy to do this. Item objects should be placed as direct descendant of the container.

When both container and item are active and enabled, the layout management should be working correctly.

Adding Visuals

Flex items and containers are not visible to start with. You may add an image component to fill its size with color. An procedural UI image component from the asset store may be a very useful tool with flexible components like these. Flex Layout only controls **RectTransform** components.

Containers within Containers

It may be common you would like an item within a container to also be a container of items. Generally, this can be done without problems. Containers compute their layout from the top of the hierarchy to the bottom of the hierarchy. Meaning inner containers depend on the size of the outer containers. Without minimum sizes and flex-shrink enabled, this may not be a problem. However, when this is not the case, combined minimum size of items can start to become larger than the container itself. This is when the inner container may overflow to other items in the outer container and cause a problem.

This can be prevented with the **UI Flex Linker** component. Simply add the component to the inner container object, which has both a container and an item component, and **enable minimum size**. It will now compute the minimum size such that it matches the combined minimum size of items in the container. The inner container will no longer overflow. Linkers compute their sizes from bottom to the top of the hierarchy before containers start their layout. More about this component in the next section.

2.3 Flex Linker Component

The **UI Flex Linker** component requires both flex item and flex container components. When enabled, this component relays information about the container's items to the item component. It can determine the minimum size of this item, such that it is equal to the combined minimum size of items within the container, with padding, margin and minimum gaps included. This behaviour can be enabled for Flex Basis size, Minimum size and Maximum size. This can be used to **prevent overflow** of the container or be used as a **content size fitter**.

Valid computed links are indicated with a blue link icon at the respective item component field. Note that horizontal container affect widths, and vertical containers affect heights.

Invalid Computation

When the linker is unable to compute sizes for the item component, a red link icon is shown at the item's fields. This can happen when the combined items/margins and gaps have:

- Greater or equal to 100% percentage.
- A negative constant in pixels.
- (for maximum size) Growth of an item is enabled without maximum size specified.

Also padding along the main axis with greater or equal to 100% percentage is considered invalid. Minimum and maximum size fields are disabled when their computation is invalid. While flex basis fields are simply no longer controlled.

Reverse Computation

Reverse computation of the linker might be counter-intuitive when percentages are involved. An item taking 95% of the content size paired with an item taking only 30 pixels will result in a content size of 600 pixels. This is because the 30 pixel item should be taking only the remaining 5%. Advised is to use moderate percentage values when using the linker. This includes item sizes, margins and minimum gaps.

3. Scripting Guide

To access components, include **using FlexLayout;** namespace. Component fields are publicly accessible and are thus not encapsulated. Changing and reading field values are not done with functions, but instead with a chain of publicly accessible fields. After each change in an **UIFlexItem** component, call **item.Revalidate()** to update the container it is in. Changes to **UIFlexContainer** values should me met with a call to **container.MarkRebuild()**. For performance reasons, try avoiding these update function calls for every frame when values haven't changed.

UIFlexItem Exmaple:

```
item.flexBasis.width.percentage = 0f;
item.flexBasis.width.pixels = 100f;
item.margin.uniform.pixels = 10f;
item.order = 20;
item.Revalidate();
```

UIFlexContainer Example:

```
container.direction = UIFlexContainer.Direction.Horizontal;
container.wrap = false;
container.minimumGap.width.pixels = 10f;
container.MarkRebuild();
```

There are also example scripts/scenes available in the **Examples** sub-folders.

4. Contact

For questions, feedback, requests and/or bug reports, feel free to send a mail. I hope to respond in time. And I hope this layout system helps with your development in games!

Greetings,

W7
w7assets@gmail.com