Advanced Input Field 1.4.3:

This plugin is a custom implementation of the Unity Input Field (it still inherits from Selectable base class). It adds several features and fixes multiple bugs that were in the original Unity Input Field. The logic has been split over multiple (platform specific) classes to improve readability.
For Mobile platforms (Android & iOS & UWP) a whole new implementation has been written, because these were the platforms with the most annoying bugs. Also new bindings for the native keyboards on Android, iOS & UWP have been added to control showing and hiding the keyboard more easily and to have more event callbacks.

Supported platforms: PC, Mac, Android, iOS & UWP (Smartphone, Tablet & PC)

WARNING: using the AdvancedInputField and the default Unity InputField together might cause unexpected behaviour.

Features:
- Default features of the original Unity InputField
- Process edited text on deselect (using PostProcessingFilters)
- Process text while input field is being edited (using LiveProcessingFilters)
- Next Input Field option to control which InputField should be selected when done editing. (Tab key on Standalone platforms and Done/Next key on Mobile platforms).
- Mobile only: Show ActionBar with cut, copy, paste and select all options
- Mobile only: Selection Cursors (Draws selection sprites for start and end of text selection to control the selected text more easily in large text blocks)
- Gestures for Mobile added: Single Tap: show ActionBar (if enabled)
  Double Tap: Select word
  Hold: Select word + show ActionBar (if enabled)
- Event for keyboard height changes in the new NativeTouchScreenKeyboard binding.
- Added a KeyboardScroller component to scroll content when TouchScreenKeyboard appears/hides. (requires a specific setup, see more info below)

Changelog:

1.4.3:

- Change Mobile: ActionBar will be drawn last in Canvas now
- Bugfix Mobile: Fixed some ActionBar actions that were acting weird
- Change: When using the “Decimal Number “ character validation the comma key will input a decimal point

1.4.1:

- Implemented character validation in the mobile bindings
- Change Mobile: Changed behaviour when ActionBar & Mobile Cursor is visible
- Bugfix Mobile: Fix for ActionBar not always working properly when Mobile Selection Cursors are disabled
- Change: SelectedText property is exposed now in AdvancedInputField class

1.4.0:

- Bugfix Mobile: Multilline submit & Multiline newline working now
- Change Mobile: Emoji characters will be blocked
- Bugfix Mobile: Fix for selection cursors not disappearing when selected text is deleted
- Bugfix Mobile: Character Validation “Name” working properly now
- Added scale option for mobile selection cursors
- Added option to enable/disable actions of the ActionBar
- Change: Allow smaller values for the caret width

1.3.2:

- New Feature: LiveProcessingFilters added to format the text in edit mode (see LiveProcessing scene in the Samples folder)
- Bugfix: Fix for caret jumping to first or last position when clicking above or belove the text of a single line input field.
- Bugfix Standalone: Fix for tab character getting inserted in the next input field when using tab to go to the next input field.
- Bugfix Android: Fix for crash when starting the app after Android OS has terminated the app.
- Bugfix Mobile: Fix for empty inputfield on first press after manual Text change.

1.3.0:

- General: Restructured script folders and renamed namespace to simplify declaration from "AdvancedInputField.AdvancedInputField inputField" to "using AdvancedInputFieldPlugin;" & "AdvancedInputField inputField".
- Bugfix iOS: Fixed bitcode errors when building for some iOS devices and archiving
- Bugfix Mobile: Fixed scaling issues of mobile caret sprites & ActionBar
- Added SelectionChangedHandler: An event has been added that will execute whenever the selection state (selected or deselected) changes.

1.2.3:

- Bugfix: Initial position of caret is rendered at the right position now when InputField is empty
- Bugfix: Fixed NullReferenceException when loading an AdvancedInputField from a prefab
- Bugfix Android: Reloads native keyboard settings now when selecting InputField to avoid text copy from previous InputField
- Change Mobile: Disables mobile caret sprite when none of the "mobile only" options are enabled now

1.2.0:

- Single line mode: Horizontal scroll working correctly now
- Bugfix Mobile platforms: All selected characters are deleted now when pressing delete/backspace key (instead of only 1 character)
- Bugfix: If the Text property of the InputField is changed before initialization happens, it will update properly now
- Bugfix Android: AndroidManifest conflict error fixed when building
- Added ManualSelect() method to select the InputField programmatically (will set the caret to the end of the text)

1.1.0:

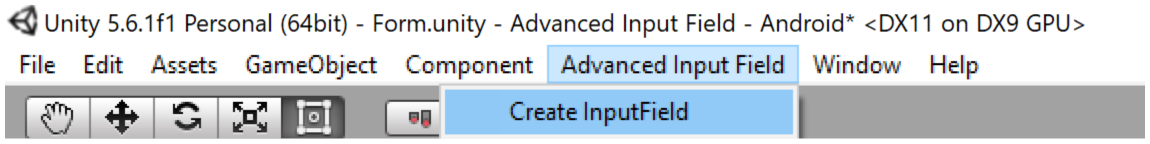
- Added support for UWP (Smartphone, Tablet and PC)
- Bugfix Mobile platforms: Added null check for OnKeyboardHeightChanged event
- Bugfix Android: keyboard is shown properly now in landscape mode
- Bugfix iOS: InputField accepts non-standard characters now
- Improvement: KeyboardScroller

1.0.2

- Changed event callback signature for OnEndEdit(String) to OnEndEdit(String, EndEditReason) EndEditReason is a enum that specifies the reason for ending edit mode(Deselect, Keyboard Cancel, Keyboard Done, Keyboard Next)
- Bugfix Android: Back key dismisses keyboard properly now

How to use:

Basic creation:
The create a new instance of AdvancedInputField use the Menu option: Advanced Input Field => Create Input Field.
This will create a new Input Field with the required components and childs.

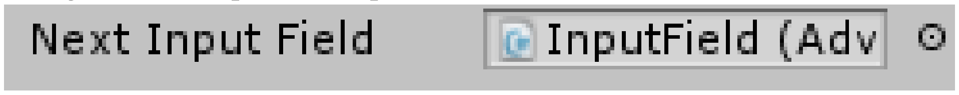

Unity 5.6.1f1 Personal (64bit) - Form.unity - Advanced Input Field - Android* <DX11 on DX9 GPU>
File
Edit
Assets
GameObject
Component
Advanced Input Field
Window
Help
Create InputField

Then just drag to it into a Canvas and you can use the default RectTransform attributes to resize it to the desired size.

Basic functionality:
For the basic functionality look at to the Unity manual:
https://docs.unity3d.com/Manual/script-InputField.html

Examples:
Check the Samples/Scenes directory for usage examples of this Plugin.

Using the Next Input Field option:



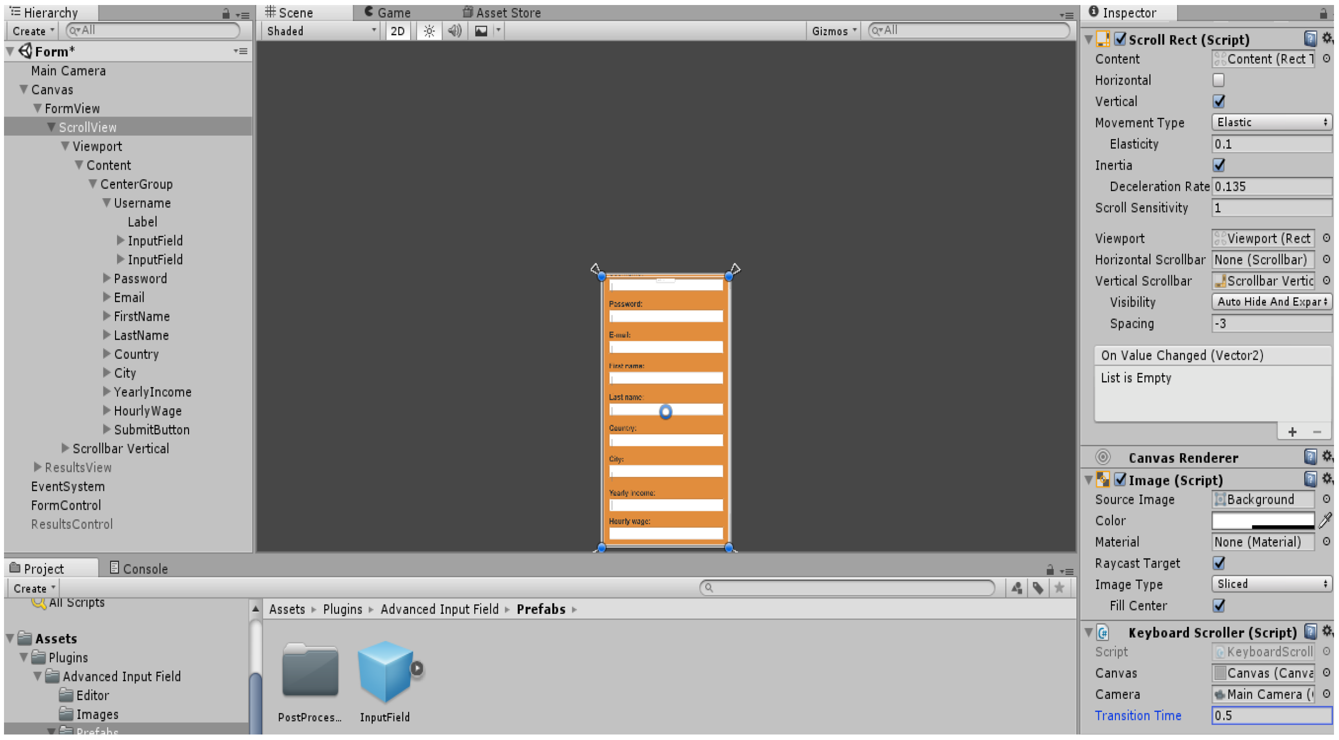
The "Next Input Field" option on the Advanced Input Field can be used to select a next Advanced Input Field instance when you're done with current one. Just drag another AdvancedInputField instance into it to use it. If you leave this field empty it won't be used (and will hide the keyboard normally on mobile platforms).
On Standalone platforms the Tab key is used to select the AdvancedInputField specified in this field.
On mobile platforms the Done/Next key is used to select the AdvancedInputField specified in this field.

Notes:

- Event callback signature for OnEndEdit is OnEndEdit(String, EndEditReason) instead of OnEndEdit(String) in the original Unity InputField.
  EndEditReason is a enum that specifies the reason for ending edit mode(Deselect, Keyboard Cancel, Keyboard Done, Keyboard Next)

Using the KeyboardScroller component:

As a starting point locate the ScrollRect in the hierarchy that contains the (Advanced)InputFields and add the KeyboardScroller component to it.



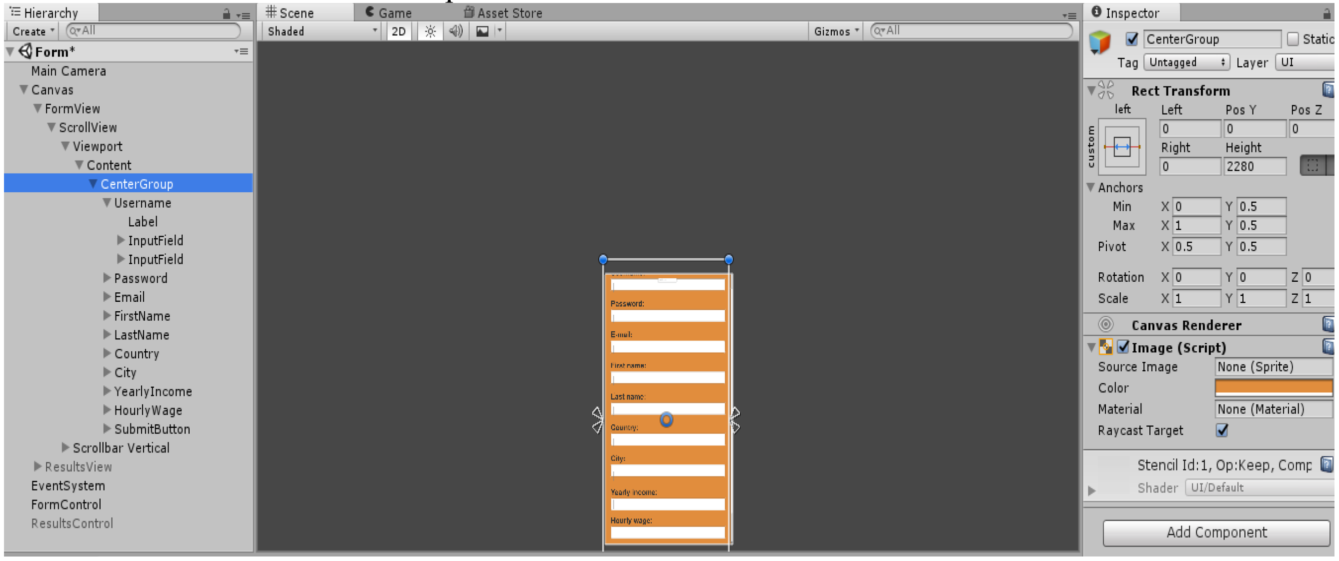
Scroll Rect (Script)
Content: Content (Rect T
Horizontal: ☐
Vertical: ✓
Movement Type: Elastic
Elasticity: 0.1
Inertia: ✓
Deceleration Rate: 0.135
Scroll Sensitivity: 1
Viewport: Viewport (Rect
Horizontal Scrollbar: None (Scrollbar)
Vertical Scrollbar: Scrollbar Vertic
Visibility: Auto Hide And Expar
Spacing: -3
On Value Changed (Vector2)
List is Empty

Canvas Renderer

Image (Script)
Source Image: Background
Color:
Material: None (Material)
Raycast Target: ✓
Image Type: Sliced
Fill Center: ✓

Keyboard Scroller (Script)
Script: KeyboardScroll
Canvas: Canvas (Canva
Camera: Main Camera (
Transition Time: 0.5

The KeyboardScroller works by adding more scrolling space when the keyboard appears. To make sure everything stays anchored correctly we're going to add a new RectTransform anchored at the vertical center to the Content Transform of the ScrollRect. (Content transform needs to anchored at the vertical center too. It's named Center Group in the screenshot below.



CenterGroup ☐ Static
Tag: Untagged Layer: UI

Rect Transform
custom / left
Left: 0 Pos Y: 0 Pos Z: 0
Right: 0 Height: 2280
Anchors
Min X 0 Y 0.5
Max X 1 Y 0.5
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer

Image (Script)
Source Image: None (Sprite)
Color:
Material: None (Material)
Raycast Target: ✓

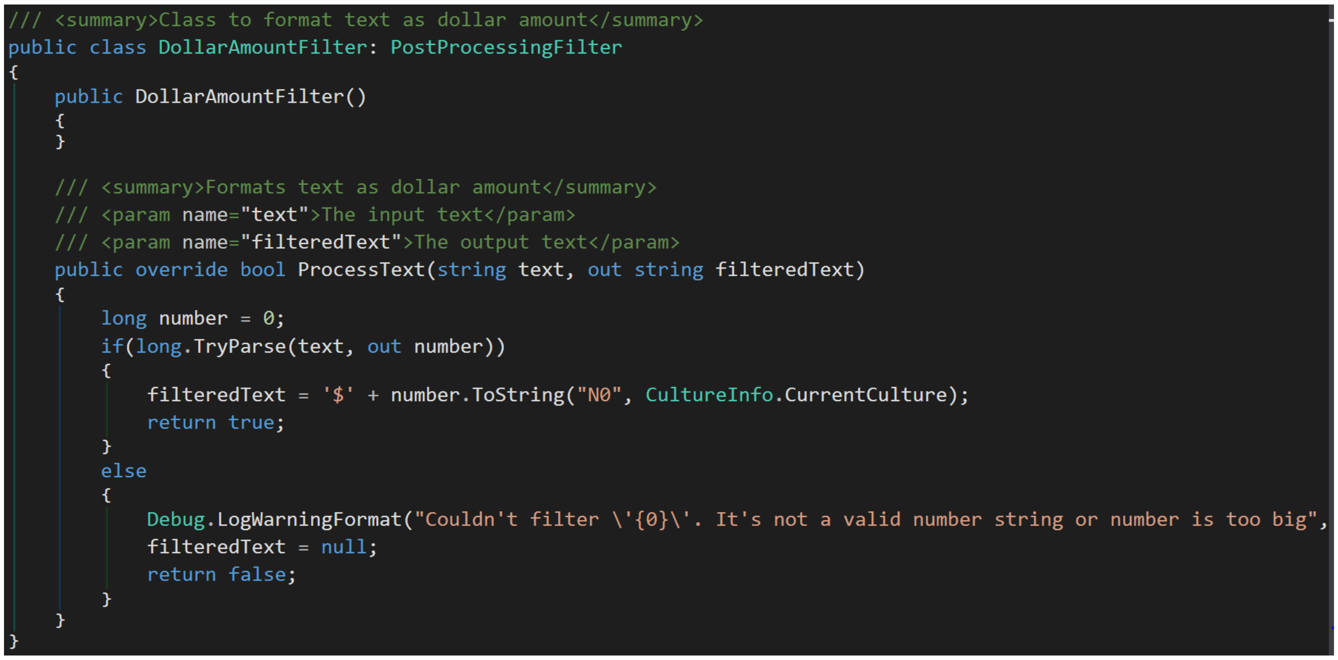
Stencil Id:1, Op:Keep, Comp
Shader: UI/Default

The size of the RectTransform of ScrollRect Content and the CenterGroup needs to be the same and be large enough so it all InputFields and other childs attached to it fit.

When you have configured them correctly the KeyboardScroller should scroll the Content by the amount of the keyboard height. Above example is also included in the Samples/Scenes/Form.unity scene if you're still unsure how to configure it.

Using Post Processing Filters:

Post processing filters will format the text in the InputField when editing is done or inputfield gets deselected. It's only the visual text that changes, the text value of the InputField will stay the same and will be visible again when selecting the InputField again.
You can use this for example to a format a number or a decimal to a specific currency:
For example you have an InputField set to number or decimal input only, after deselecting the InputField the visual value changes from 134 to $134 (or $134.00 if you want decimals).
The basic filter for this behaviour is available in the Scripts/PostProcessing directory (see DollarAmountFilter.cs).

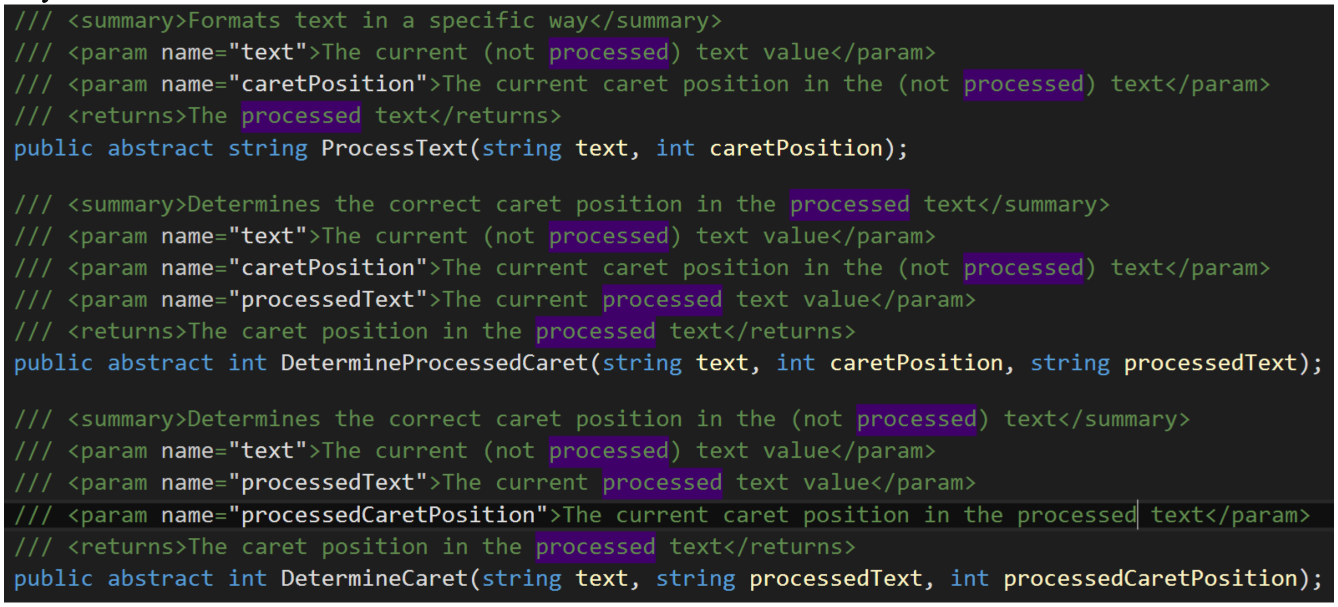


```
/// <summary>Class to format text as dollar amount</summary>
public class DollarAmountFilter: PostProcessingFilter
{
    public DollarAmountFilter()
    {
    }

    /// <summary>Formats text as dollar amount</summary>
    /// <param name="text">The input text</param>
    /// <param name="filteredText">The output text</param>
    public override bool ProcessText(string text, out string filteredText)
    {
        long number = 0;
        if(long.TryParse(text, out number))
        {
            filteredText = '$' + number.ToString("N0", CultureInfo.CurrentCulture);
            return true;
        }
        else
        {
            Debug.LogWarningFormat("Couldn't filter \'{0}\'. It's not a valid number string or number is too big",
            filteredText = null;
            return false;
        }
    }
}
```

To create your own filter create a new code file to inherits from PostProcessingFilter and override the ProcessText() method. The “text” parameter is the input value and the “filteredText” is the output value that will be shown to the user. The next step is to create a new GameObject and attach your script to it. Then drag this GameObject to a directory of choice (so a Prefab will be created). After that you can destroy that GameObject in the scene.
That's it, now you can just drag that Prefab into to “Post Processing Filter” field of a Advanced Input Field.

Using Live Processing Filters:
Live processing filters will format the text in the InputField while an InputField is selected/in edit mode). Just as Post Processing Filters it will modify the visual text, the text value of the InputField will stay the same.



```
/// <summary>Formats text in a specific way</summary>
/// <param name="text">The current (not processed) text value</param>
/// <param name="caretPosition">The current caret position in the (not processed) text</param>
/// <returns>The processed text</returns>
public abstract string ProcessText(string text, int caretPosition);

/// <summary>Determines the correct caret position in the processed text</summary>
/// <param name="text">The current (not processed) text value</param>
/// <param name="caretPosition">The current caret position in the (not processed) text</param>
/// <param name="processedText">The current processed text value</param>
/// <returns>The caret position in the processed text</returns>
public abstract int DetermineProcessedCaret(string text, int caretPosition, string processedText);

/// <summary>Determines the correct caret position in the (not processed) text</summary>
/// <param name="text">The current (not processed) text value</param>
/// <param name="processedText">The current processed text value</param>
/// <param name="processedCaretPosition">The current caret position in the processed text</param>
/// <returns>The caret position in the processed text</returns>
public abstract int DetermineCaret(string text, string processedText, int processedCaretPosition);
```

The 3 required methods for a Live Processing Filter are basically:
1. Convert current text value to processed text value
2. Determine processed caret position (character index in the processed text value) based on given input parameters.
3. Determine caret position (character index in the text value) based on given input parameters.

You can use this for example to a format a credit card number in groups of 4 digits separated by a space character or to add '/' characters to format a number field as '01/10/2017'.
To create your own filter create a new code file to inherits from LiveProcessingFilter and override the abstract methods.

Following steps are mostly the same as with Post Processing Filters:
Create a new GameObject and attach your script to it. Then drag this GameObject to a directory of choice (so a Prefab will be created). After that you can destroy that GameObject in the scene.
That's it, now you can just drag that Prefab into to “Live Processing Filter” field of a Advanced Input Field.

There are example implementations for credit card and date filters. See the scripts in the Scripts/LiveProcessing folder and the sample scene “LiveProcessing” in the Samples directory.