Advanced Input Field 1.7.1:

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/format edited text on deselect (using PostProcessingFilters)
- Process/format text while input field is being edited (using LiveProcessingFilters)
- Validate characters when text changes in native code (using CustomCharacterValidators)
- 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)
- Mobile only: Support for (bluetooth) hardware keyboards
- 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 NativeKeyboard binding.
- Added a KeyboardScroller component to scroll content when NativeKeyboard appears/hides. (requires a specific setup, see more info below)
- Support for TextMeshPro Text Renderers
- Experimental Emoji support (requires TextMeshPro)
- Multiple InputField Modes (indicates how to handle text bounds changes): SCROLL_TEXT, HORIZONTAL_RESIZE_FIT_TEXT and VERTICAL_RESIZE_FIT_TEXT

Table of Contents

Changelog

1.7.1:

- Improvement: Text Validation speed boost
- Bugfix: Sometimes incorrect text bounds on password fields
- Bugfix Mobile: Incorrect initial mobile cursor position when using KeyboardScroller component
- Bugfix: Incorrect text bounds when RectTransform size got changed

1.7.0:

- Added Custom Character Validator option
- Added Drag Modes (UPDATE_TEXT_SELECTION, MOVE_TEXT)
- Added Selection Modes (SELECT_ON_RELEASE, SELECT_ON_PRESS)
- Bugfix iOS: Fix incorrect keyboard height on newer iOS devices
- Bugfix: Input issues with some of the Pointer Events
- Change: Added tap threshold for begin edit mode
- Bugfix iOS: Errors when running in the iOS Simulator

1.6.3:

- Improvement: Emoji support more stable
- Bugfix UWP: Keyboard input not working on PC
- Bugfix: Autoresize issues
- Bugfix: Caret position different from click position when text alignment is not upper left.
- Bugfix Android: Keyboard stuck on pause
- Bugfix Android: Password field showing suggestions
- Bugfix Mobile: ActionBar not reparenting to current Canvas properly (when there are multiple Canvases in the scene)
- Bugfix Mobile: Manual Text change not updating native text properly
- Bugfix Mobile: ActionBar reference missing in runtime created AdvancedInputField

1.6.1:

- Added autoresize (horizontal/vertical) modes to make the InputField resize automatically to fit the text content.
- Bugfix: Other text alignment options than top left work properly now with to new AdvancedInputField format.
- Bugfix Mobile: ActionBar will move to the bottom of the InputField when it can't be displayed completely at to the top of the screen
- Change iOS: Replaced -ObjC flag with -force_load flag to improve compatibility with other plugins

1.6.0:

- Added TextMeshPro support
- Change: Format of an AdvancedInputField instance has changed to support both Unity and TextMeshPro Text Renderers
- Added ConversionTool to convert deprecated AdvancedInputField instances (= made with an earlier version than 1.6.0) to the new format.
- Change: Uses a custom ScrollView now instead of shifting the text characters
- Added Multiple Text Scroll properties
- Change: There is a dedicated Text Renderer now for the placeholder, Processed Text Renderer isn't used for the placeholder anymore
- Added (Experimental) Emoji support
- Bugfix iOS: Split keyboard getting stuck
- Bugfix UWP: Il2Cpp build works now

1.5.7:

- Added additional mode to KeyboardScroller component to scroll only when keyboard is in front of focused input field
- Fix: KeyboardScroller component not scrolling correctly when Canvas mode is overlay
- Fix: Rendered caret position at wrong position when text alignment is not left
- Fix: Disabled Interactable property still allowing focus of input field
- Bugfix Mobile: Hold not working correctly to show ActionBar
- Bugfix Android: Native binding not handling offset of NavigationBar properly when disabling fullscreen on some devices
- Bugfix iOS: Russian characters incorrectly being allowed when CharacterValidation is Email
- Bugfix iOS: Keyboard doesn't show when Splashscreen is disabled

1.5.5:

- Bugfix iOS: Fixed undefined symbols for architecture error

1.5.4:

- Added post build script to add required flag for iOS (Xcode) build automatically
- Added “AutocapitalizationType” option to give the native keyboard a hint to use a specific type of autocapitalization behavior.
- Change: TouchscreenKeyboard stays visible now when user has a inputfield selected and selects another inputfield
- Bugfix: Various minor bug

1.5.2:

- API Changes: Marked some methods as internal and added more public methods and events in AdvancedInputField class
- Added "CaretOnBeginEdit" option to control where the caret should be located when beginning edit mode
- Change: Added a global settings file (Resources/AdvancedInputField/Settings.asset) and moved some mobile only settings to that file
- Fix: Caret position was rendered at wrong position when using some fonts
- Fix: Single line InputField was sometimes rendering multiple lines when caret was set to end of text
- Bugfix Android: A crash occurred when starting the app after Android OS had terminated the app
- Bugfix Mobile: Native bindings were incorrectly sending cancel events when switching to other keyboard input methods from the touchscreenkeyboard
- Bugfix iOS: Workaround for not receiving hardware keyboard events properly with Unity's EventSystem
- Bugfix iOS: Fixed Character limit
- Bugfix iOS: Fixed Duplicate symbols errors when compiling using -ObjC flag and using other plugins in the same project

1.5.0:

- Improvement Mobile: Rewritten mobile bindings for more stability
- Added "Sentence" Character Validation type to autocapitalize first letter of each sentence
- Added "IPAddress" Character Validation type to validate a IPv4 Address
- Mobile: Added support for (bluetooth) hardware keyboards
- Bugfix: Initial caret position was at wrong place
- Bugfix: Pasting text with newline characters caused OutOfRangeExceptions
- Bugfix: Keyboard was reappearing when cancel key was pressed
- Bugfix Mobile: Suggestions will be hidden when autocorrect is disabled now
- Bugfix Standalone: Scrolling with arrow keys caused OutOfRangeExceptions
- Bugfix iOS: Name Character Validation allowed some invalid characters (currency signs)
- Bugfix iOS: Undo button caused a crash sometimes

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

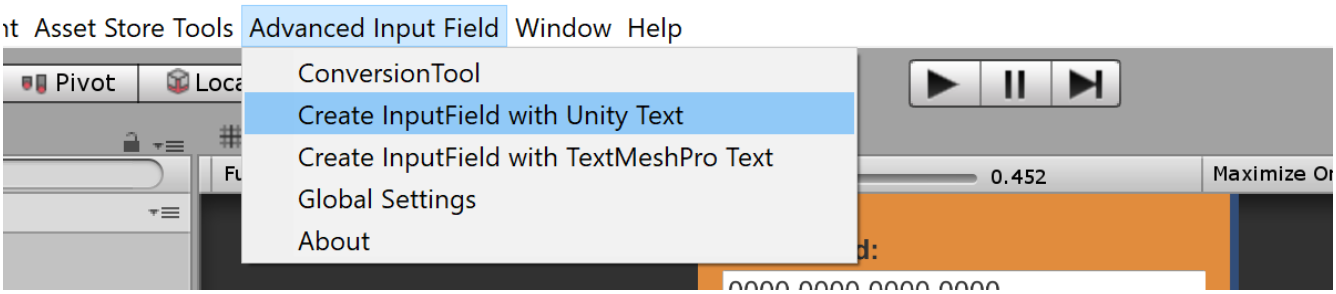
1.0.2

- 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

General

Basic creation

To create a new instance of AdvancedInputField use the Menu option: Advanced Input Field => Create



Input Field with Unity Text.
This will create a new Input Field with the required components and childs.
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)
- For UWP build:
  Please use XAML as build type, D3D seems to give issues.

TextMeshPro

Using TextMeshPro Text Renderers (requires Unity 2018.1.0 or newer)

To use TextMeshPro Text Renderers in combination with this plugin use the following steps:

1. Install the TextMeshPro package from the PackageManager window in Unity:
(Window => Package Manager)
2. Find the TextMeshPro package in the list and install it
3. Open the Player Settings in Unity
(Edit => Project Settings => Player)
4. In “Scripting Define Symbols” add the following value:
ADVANCEDINPUTFIELD_TEXTMESHPRO
5. Recompile (normally Unity does this automatically)

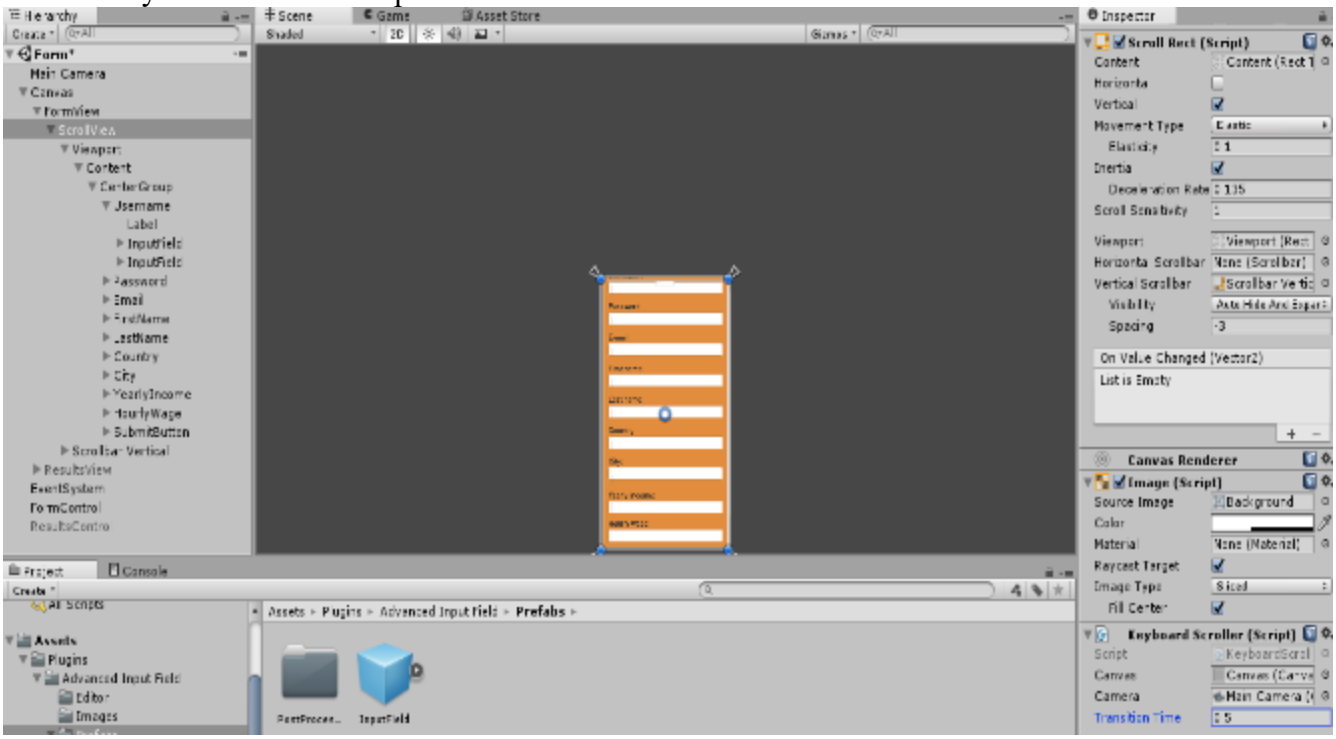
You can either convert from an existing Input Field to an AdvancedInputField instance with TextMeshPro Text Renderers using the ConversionTool (Advanced Input Field => ConversionTool) or create a new instance using the menu option (Advanced Input Field => Create InputField with TextMeshPro Text)

Enabling Emojis (requires TextMeshPro Text Renderers)

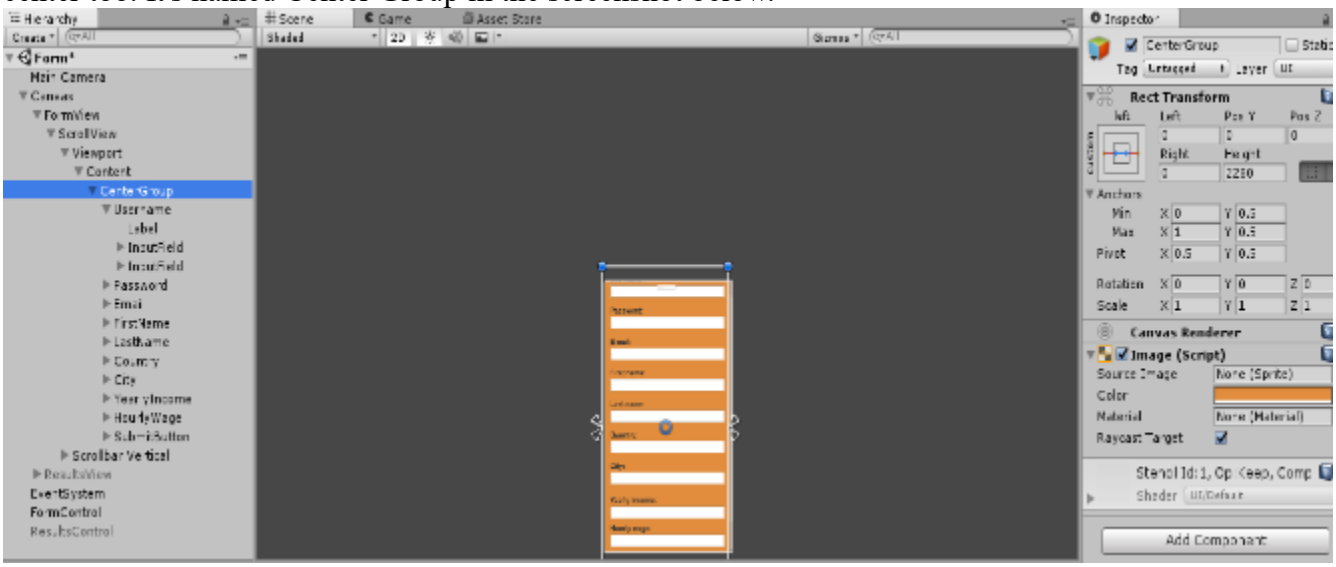
IMPORTANT: This feature is currently experimental and may have weird behavior.
To allow emojis change the ContentType of the AdvancedInputField instance to “Custom” and enable the “(Experimental) Emojis Allowed” property.

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.



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.



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);
```

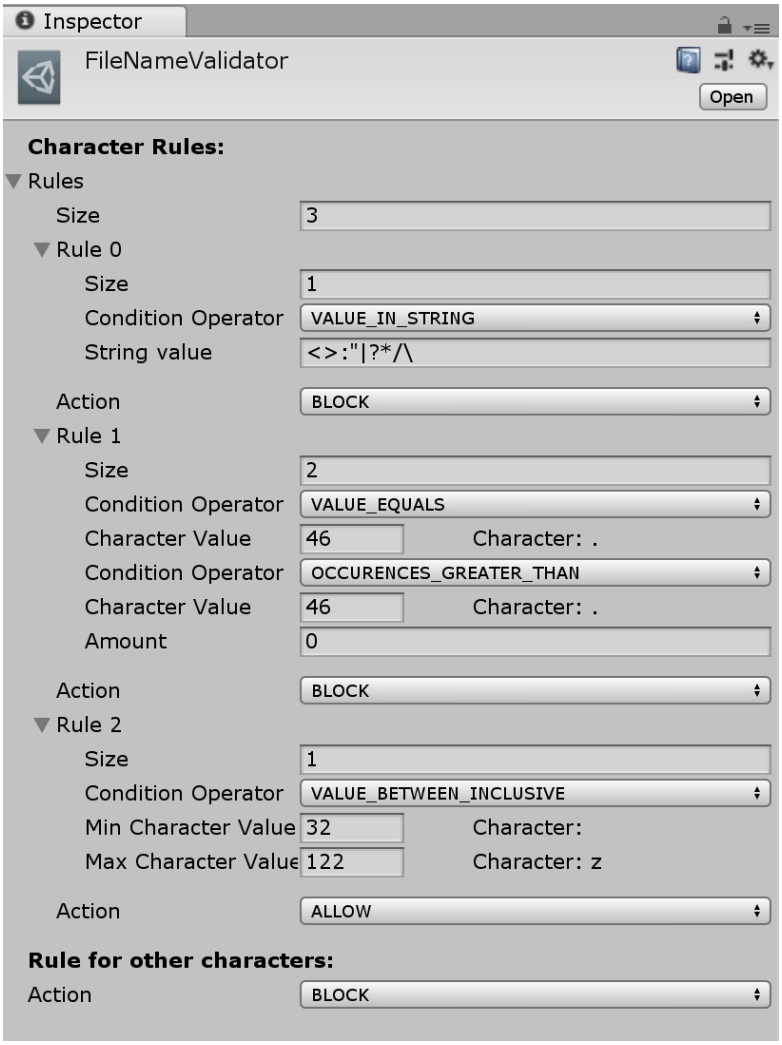
The 3 required methods for a Live Processing Filter are basically:
1. Convert current text value to processed text value
2. Determine processed caret position (character index in the processed text value) based on given input parameters.
3. Determine caret position (character index in the text value)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

Using (Custom) CharacterValidators



CharacterValidators can be used to block/allow certain characters whenever text changes in the native InputField code for each platform. The format is serializable so it can be executed in native code as soon as the native text changes.

The format works as follows:
CharacterRules:
- Holds all character rules to check when validating a character

CharacterRule:
- Each rule can have multiple conditions. When multiple conditions are defined in a rule they all must be true for the action to be executed.
- The rules will be checked from first to last. If one of the rules gets executed (his conditions are met) it won't check the remaining rules anymore, so keep in mind that the order of rules matters.

Rule for other characters
- If none of the rules apply, then the action defined at "Rule for other characters" will be executed

<u>The above example roughly means the following:</u>

```
for each character (ch) check:
if("<>:\"|?*/\\".Contains(ch)) //Check if value string holds current character
{
        Block current character
}
else if((int)ch == 46 && Occurences((char)46)  >  0) //Check if current character is '.' and if the amount of occurrences of '.' in current text is greater than 0
{
        Block current character
}
else if((int)ch >= 32 && (int)ch <= 122) //Check if character value is between 32 and 122
{
        Allow current character
}
else
{
        Block current character
```

}