Advanced Input Field 2.0.9

The improved version of the original Advanced Input Field plugin. Rewritten to support more complex features.

This plugin provides a more Advanced Input Field that has a lot more features and properties than the official Unity InputField. It still inherits from the Selectable base class, so it can be used with the Unity EventSystem.

Also, it has it's own bindings for the native keyboards. This made it possible to provide a better user experience and to fix several issues that existed in the official Unity InputField.

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

Features:

- Default features of the official Unity InputField
- More event callbacks (OnSelectionChanged, OnCaretPositionChanged, OnSpecialKeyPressed,...)
- Filters to process, decorate, block or allow text changes
- 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).
- Show ActionBar with cut, copy, paste and select all options
- Touch Selection Cursors (Draws selection sprites for start and end of text selection to control the selected text more easily in large text blocks)
- Event for keyboard height changes in the new NativeKeyboard binding.
- KeyboardScroller component to scroll content when NativeKeyboard appears/hides.
- Support for TextMeshPro Text Renderers
- Support for full emoji range
- Support for rich text
- Multiple InputField Modes (indicates how to handle text bounds changes): SCROLL_TEXT, HORIZONTAL_RESIZE_FIT_TEXT and VERTICAL_RESIZE_FIT_TEXT
- And more...

Table of contents

Upgrade notes

Please read the following when upgrading from version 1.*.* to version 2.*.* of the plugin:

1. Remove the following files from the old plugin installation:

- Assets/Plugins/AdvancedInputField
- Assets/Plugins/Android/NativeKeyboard.aar
- Assets/Plugins/iOS/NativeKeyboard.a
- Assets/Plugins/WSA/NativeKeyboard.dll
- Assets/Plugins/WSA/NativeKeyboard.pdb
- Assets/Plugins/WSA/NativeKeyboard.pri

Everything has been moved to the Assets/AdvancedInputField directory in version 2.0.0 of the plugin

2. All enum values now use the same naming style (All uppercase). For example:

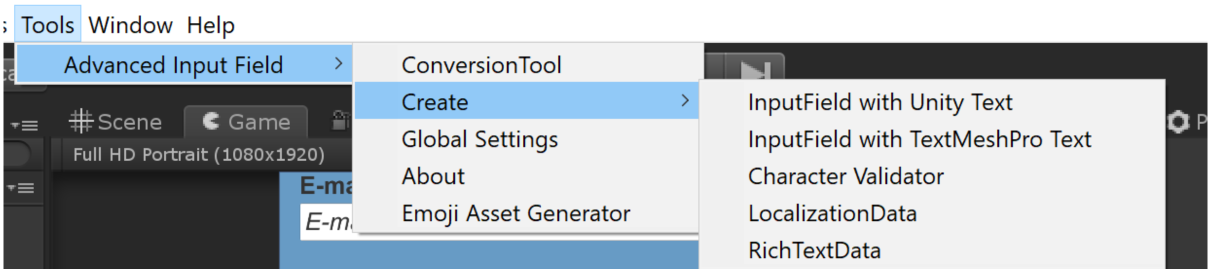
```
public enum LineType { SINGLE_LINE, MULTILINE_SUBMIT, MULTILINE_NEWLINE }
```

3. The old LiveProcessingFilter & PostProcessingFilter components have been renamed to LiveDecorationFilter & PostDecorationFilter to better suit their intended behaviour. The new LiveProcessingFilters are intended to intercept, modify or block changes from the native keyboard.

General

Basic creation

To create a new instance of AdvancedInputField use the Menu option:
TopBar: Tools => Advanced Input Field => Create => InputField with Unity Text (or InputField with TextMeshPro Text).

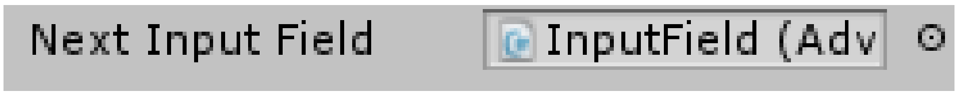


This will create a new Input Field with the required components and childs.
Then just drag to it into a Canvas and you can use the default RectTransform attributes to resize it to the desired size.

Examples

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

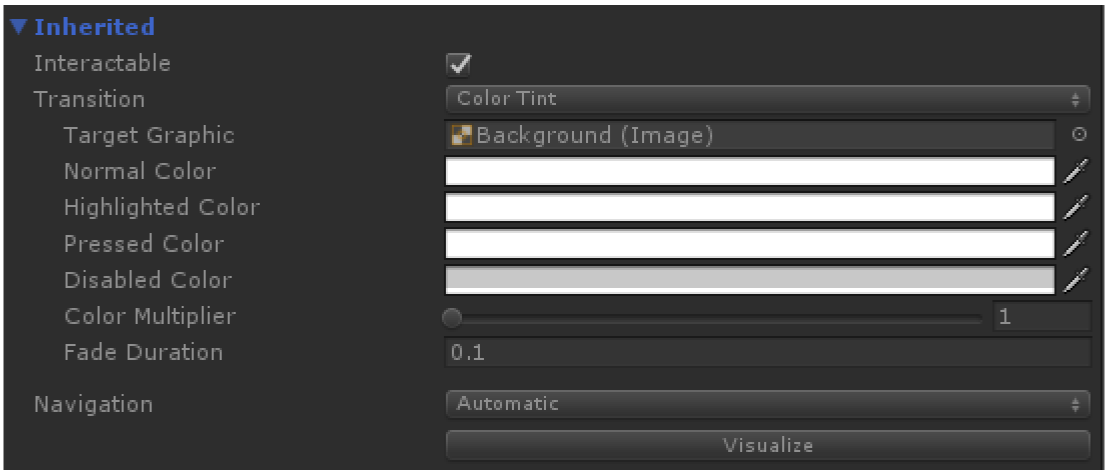
Using the Next Input Field option



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.

Inspector properties

Inherited

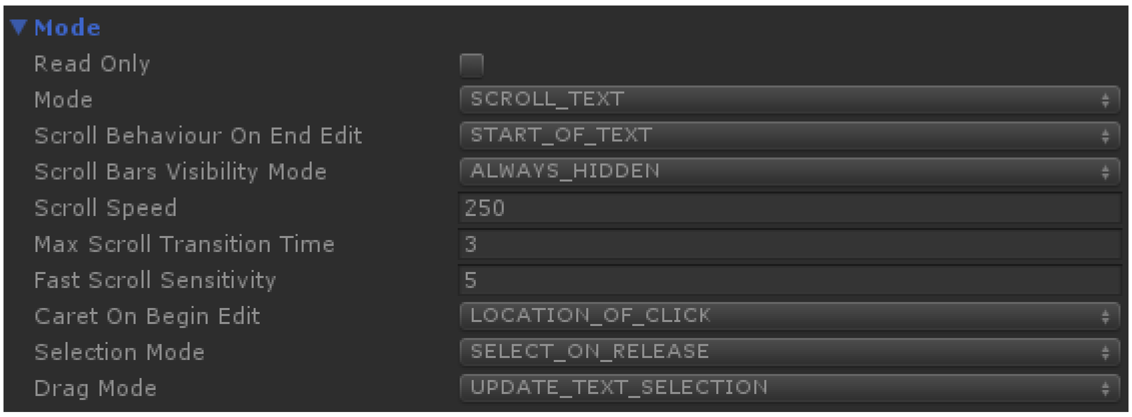


Inherited

Interactable	✓
Transition	Color Tint
Target Graphic	Background (Image)
Normal Color	⬜
Highlighted Color	⬜
Pressed Color	⬜
Disabled Color	▨
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic

This section contains the inherited properties from the Selectable base class. For more information about these properties see the official documentation of Unity: https://docs.unity3d.com/2019.1/Documentation/ScriptReference/UI.Selectable.html

Mode



Mode	
Read Only	☐
Mode	SCROLL_TEXT
Scroll Behaviour On End Edit	START_OF_TEXT
Scroll Bars Visibility Mode	ALWAYS_HIDDEN
Scroll Speed	250
Max Scroll Transition Time	3
Fast Scroll Sensitivity	5
Caret On Begin Edit	LOCATION_OF_CLICK
Selection Mode	SELECT_ON_RELEASE
Drag Mode	UPDATE_TEXT_SELECTION

Read Only: Indicates whether modifications to the text by the user should be blocked (for example: block keyboard input and paste actions). When enabled the input field is still focusable/selectable though, so the user could select and copy the text. If you don't want it to be focusable too set the "interactable" property to disabled.

Mode: Determines how changes in the text bounds should be handled. Following value can be used:

- SCROLL_TEXT: Scroll the text horizontally when it doesn't fit the input field bounds
- HORIZONTAL_RESIZE_FIT_TEXT: Expand the input field horizontally so the text fits in the input field bounds
- VERTICAL_RESIZE_FIT_TEXT: Expand the input field vertically so the text fits in the input field bounds

When using one of the resize modes you can also specify the minimal width/height and maximum width/height. Setting the max value to zero means that the input field can expand infinitely.

Scroll Behaviour On End Edit: The scroll behaviour when ending edit mode of the focused input field. Following values can be used:

- NO_SCROLL: Don't scroll the text
- START_OF_TEXT: Scroll back to start of text
- START_OF_TEXT: Scroll forward to end of text

Scroll Bars Visibility Mode: The visibility mode for the scroll bars. Following values can be used:

- ALWAYS_HIDDEN: Never show the scroll bars
- ALWAYS_VISIBLE: Always show the scroll bars
- IN_EDIT_MODE_WHEN_NEEDED: Only show the scroll bars when focused input field needs them to view the complete text
- ALWAYS_WHEN_NEEDED: Always show the scroll bars if they are needed to view the complete text

Scroll Speed: The speed to scroll text (in Canvas pixels per second)

Max Scroll Transition Time: The maximum time it can take to scroll to caret position (in seconds)

Fast Scroll Sensitivity: The additional scroll sensitivity when dragging out of bounds

Caret On Begin Edit: The caret position in the text when beginning edit mode. Following values can be used:

- LOCATION_OF_CLICK: When clicking the input field to focus it use the position of the click to determine the caret position
- START_OF_TEXT: Set the caret position to start of text when focusing the input field
- END_OF_TEXT: Set the caret position to end of text when focusing the input field

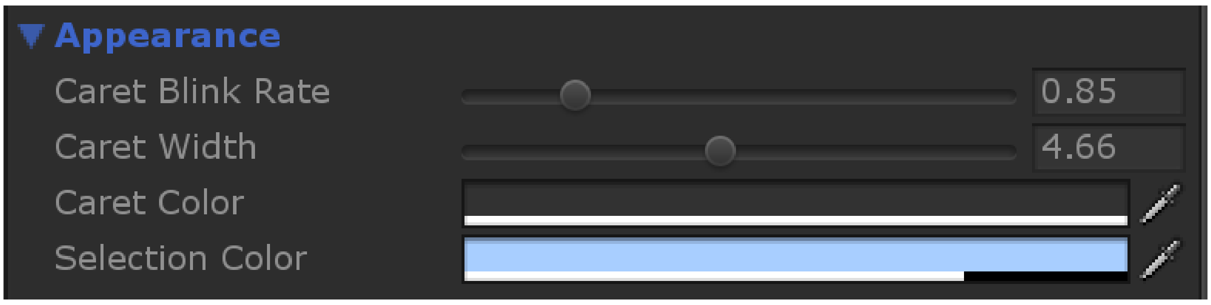
Selection Mode: Determines which input event to use to select the input field. Following values can be used:

- SELECT_ON_RELEASE: Select the input field on Pointer Release when the Pointer event is within the input field bounds
- SELECT_ON_PRESS: Select the input field on Pointer Press when the Pointer event is within the input field bounds

Drag Mode: Determines how to use drag events. Following values can be used:

- UPDATE_TEXT_SELECTION: When dragging on the input field update the text selection
- MOVE_TEXT: When dragging on the input field move/scroll the text

Appearance


▼ Appearance
Caret Blink Rate 0.85
Caret Width 4.66
Caret Color
Selection Color

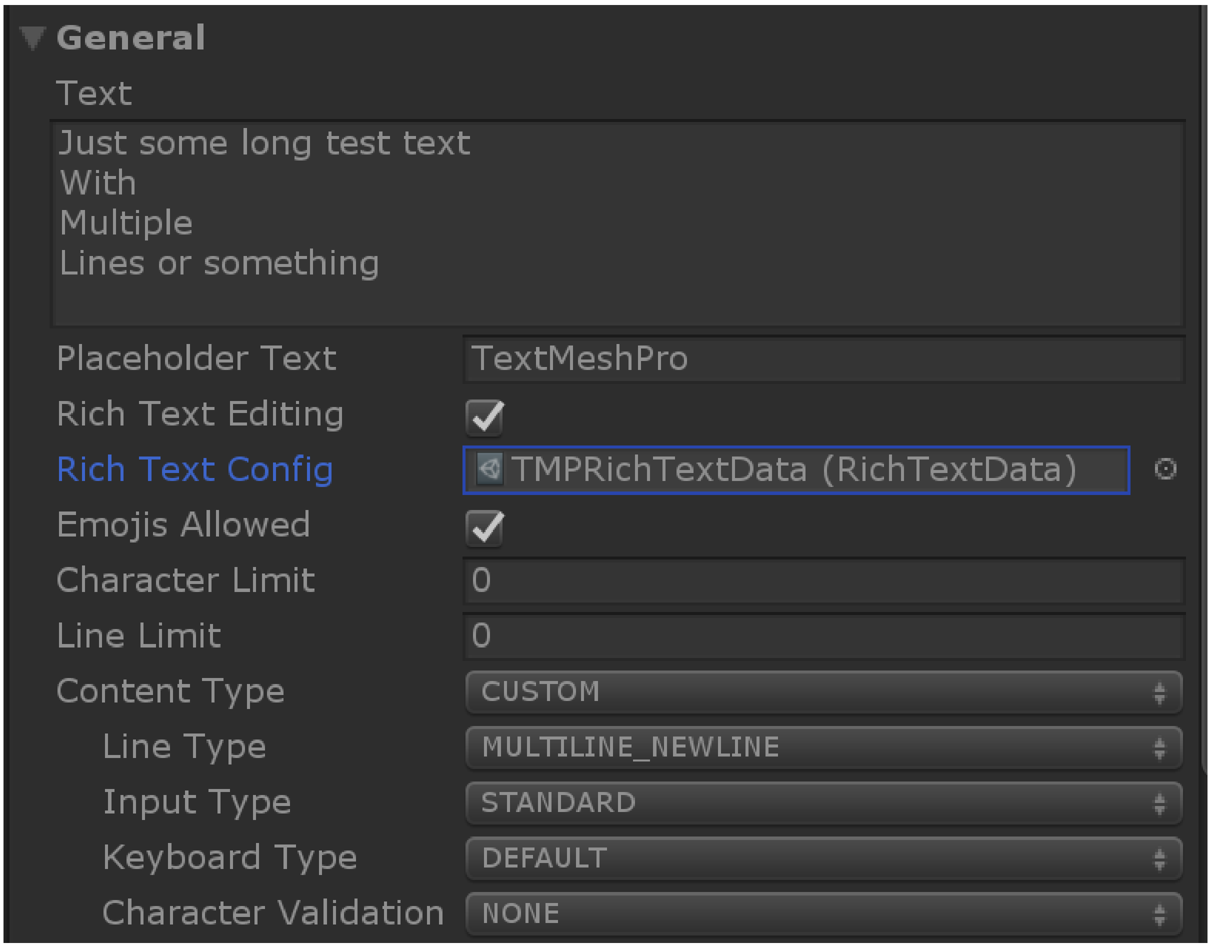
Caret Blink Rate: The blink rate of the caret (in seconds)

Caret width: The width of the caret (in Canvas pixels)

Caret color: The color of the caret

Selection color: The color of the text selection

General


General

Text

```
Just some long test text
With
Multiple
Lines or something
```

Placeholder Text	TextMeshPro
Rich Text Editing	✓
Rich Text Config	TMPRichTextData (RichTextData)
Emojis Allowed	✓
Character Limit	0
Line Limit	0
Content Type	CUSTOM
Line Type	MULTILINE_NEWLINE
Input Type	STANDARD
Keyboard Type	DEFAULT
Character Validation	NONE

Text: The main text string

Placeholder Text: The placeholder text string. Will be displayed if the main text string is empty

Rich Text Editing: Enables editing of text with rich text tags

Rich Text Config: The rich text configuration. Configuration hold a list of rich text tags that should be supported

Emojis Allowed: Indicates whether emojis should be allowed. Can be used in combination with rich text editing

Character Limit: The maximum amount of characters allowed, zero means infinite

Line Limit: The maximum amount of lines allowed, zero means infinite

Content Type: Configuration preset for the content of this InputField. Following values can be used:

- STANDARD: Preset for standard text input.
  - Input Type: STANDARD
  - Keyboard Type: DEFAULT
  - Character Validation: NONE

- AUTOCORRECTED: Preset for autocorrected text input.
  - Input Type: AUTOCORRECT

  - Keyboard Type: DEFAULT
  - Character Validation: NONE

- INTEGER_NUMBER: Preset for integer number text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: NUMBER_PAD
  - Character Validation: INTEGER

- DECIMAL NUMBER: Preset for decimal number text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: NUMBERS_AND_PUNCTUATION
  - Character Validation: DECIMAL

- DECIMAL NUMBER_FORCE_POINT: Preset for decimal number text input that automatically converts a comma character to a decimal point character.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: NUMBERS_AND_PUNCTUATION
  - Character Validation: DECIMAL

- ALPHANUMERIC: Preset for alphanumeric text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: ASCII_CAPABLE
  - Character Validation: ALPHANUMERIC

- NAME: Preset for name text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: DEFAULT
  - Character Validation: NAME

- EMAIL_ADDRESS: Preset for email address text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: EMAIL_ADDRESS
  - Character Validation: EMAIL_ADDRESS

- PASSWORD: Preset for password text input.
  - Line Type: SINGLE_LINE
  - Input Type: PASSWORD
  - Keyboard Type: DEFAULT
  - Character Validation: NONE

- PIN: Preset for pin text input.

  - Line Type: SINGLE_LINE
  - Input Type: PASSWORD
  - Keyboard Type: NUMBER_PAD
  - Character Validation: INTEGER

- IP_ADDRESS: Preset for ip address text input.
  - Line Type: SINGLE_LINE
  - Input Type: STANDARD
  - Keyboard Type: PHONE_PAD
  - Character Validation: IP_ADDRESS

- SENTENCE: Preset for sentence text input.
  - Line Type: MULTILINE_NEWLINE
  - Input Type: STANDARD
  - Keyboard Type: DEFAULT
  - Character Validation: SENTENCE

- CUSTOM: Custom, set to properties manually

Line Type: The type of line. Following values can be used:

- SINGLE_LINE: Basic single line text input, no newline character allowed and the whole text will stay on one line
- MULTILINE_SUBMIT: Multiline text input, no newline character allowed and text will be wrapped over multiple lines if text doesn't fit completely on the first line
- MULTILINE_NEWLINE: Multiline text input, newline character allowed and text will be wrapped over multiple lines if text doesn't fit completely on the first line. Also, done key will insert a newline instead

Input Type: The type of input. Following values can be used:

- STANDARD: Standard text input
- AUTOCORRECT: Show suggestions bar on Mobile platforms. NOTE: A lot of Android devices just ignore this flag and always show the suggestions bar
- PASSWORD: Password input, masks the text input with the asterisk characters ('*')

Keyboard Type: The keyboard to use on Mobile platforms. Following values can be used:

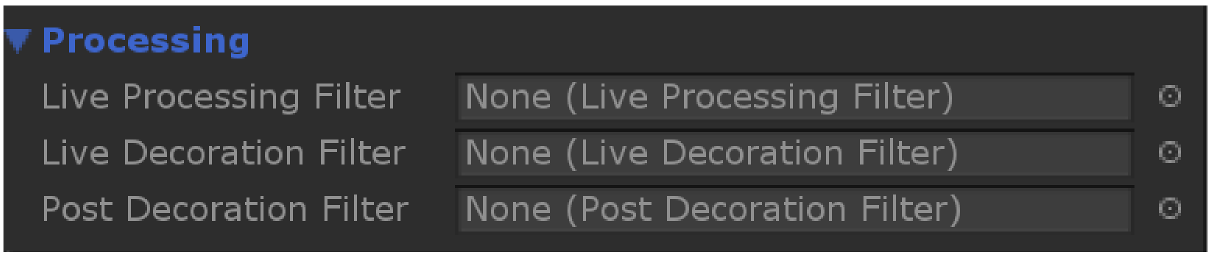
- DEFAULT: The default keyboard with all characters
- ASCII_CAPABLE: Keyboard that displays standard ASCII characters
- NUMBERS_AND_PUNCTUATION: Keyboard that displays number and punctuation characters
- URL: Keyboard that displays characters needed for url input
- NUMBER_PAD: Keyboard that displays number characters
- PHONE_PAD: Keyboard that displays number and special characters needed for phone number input
- EMAIL_ADDRESS: Keyboard that display characters needed for email address input

Character Validation: The validation to use for the text. Following values can be used:

- NONE: No validation
- INTEGER: Only allow number characters
- DECIMAL: Only allow number, minus, dot and comma characters
- DECIMAL_FORCE_POINT: Only allow number, minus, dot characters. Comma characters will automatically be converted to decimal point/dot character
- ALPHANUMERIC: Only allow alphanumeric characters
- NAME: Automatically uppercase and lowercase characters as the user is typing a name
- EMAIL_ADDRESS: Only allow characters that are valid in an email address and check if only 1 '@' character is used
- IP_ADDRESS: Only allow number and dot characters and check if each section of an ip address has the correct length
- SENTENCE: Automatically uppercase first letter when typing a dot followed by a space character
- CUSTOM: Use you own Character Validator (See Custom Character Validator section)

Next Input Field: The next input field to select when pressing the next key on the keyboard

Processing


Processing

Live Processing Filter: None (Live Processing Filter)

Live Decoration Filter: None (Live Decoration Filter)

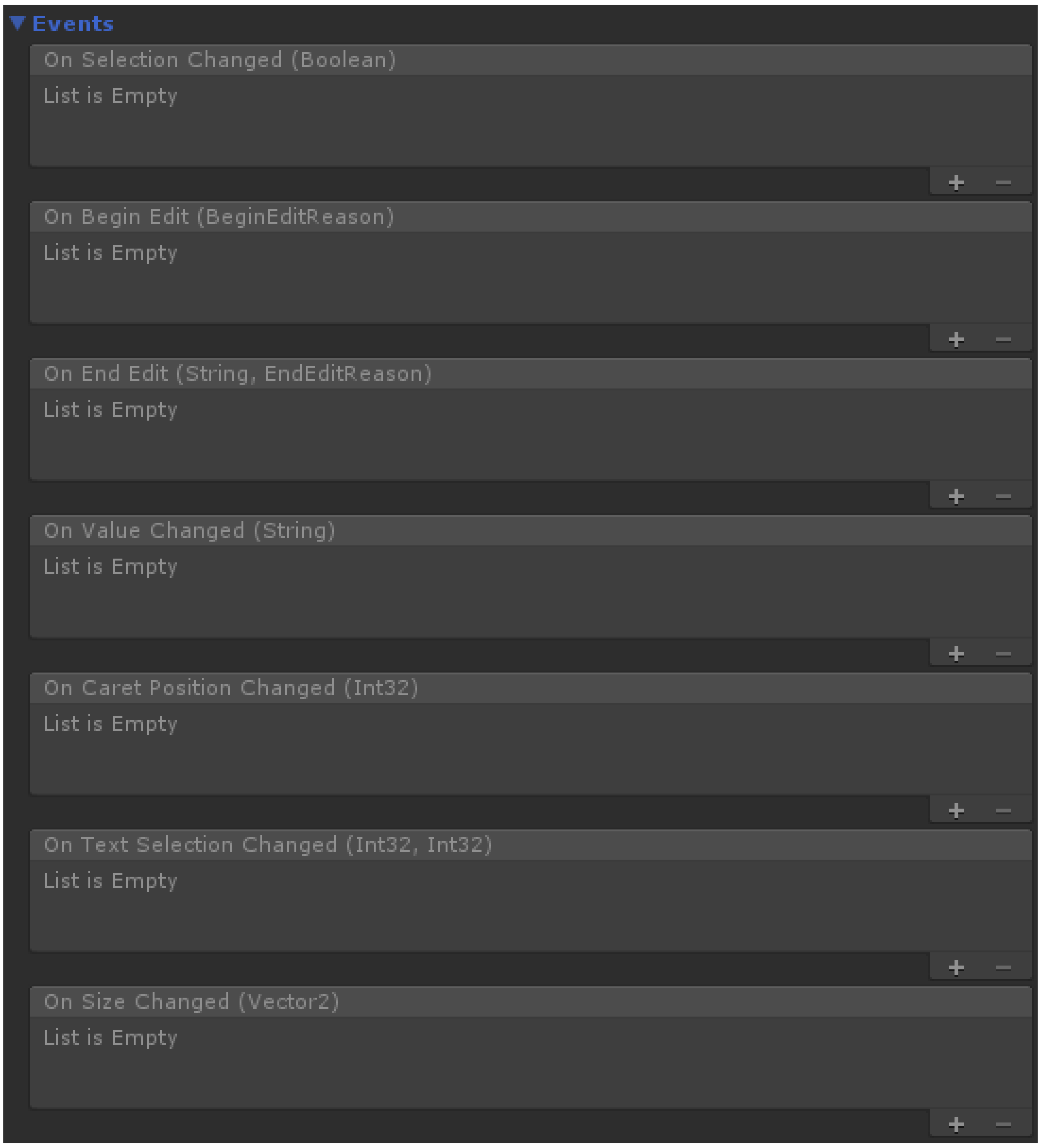
Post Decoration Filter: None (Post Decoration Filter)

Live Processing Filter: Use a filter to process any modifications in text and text selection made by the native keyboard. Can be used to block/allow/modify these “TextEditFrames”.

Live Decoration Filter: Use a filter to add extra “decoration” characters such as slashes, spaces & underscores when editing the text. For example: a DateFilter to automatically add slashes between numbers. The source text string will stay the same (without the decoration characters).

Post Decoration Filter: Use a filter to add extra “decoration” characters such as slashes, spaces & underscores when finishing editing the text. For example: a DollarFilter to automatically add a dollar sign when editing is finished. The source text string will stay the same (without the decoration characters).

Events


▼ Events

On Selection Changed (Boolean)
List is Empty

On Begin Edit (BeginEditReason)
List is Empty

On End Edit (String, EndEditReason)
List is Empty

On Value Changed (String)
List is Empty

On Caret Position Changed (Int32)
List is Empty

On Text Selection Changed (Int32, Int32)
List is Empty

On Size Changed (Vector2)
List is Empty

OnSelectionChanged(bool selected): Event called when this input field gets selected or deselected. Includes the selected state as a boolean

OnBeginEdit(BeginEditReason reason): Event called when beginning to edit the text. Includes the reason as a BeginEditReason. BeginEditReason can have following values:

- USER_SELECT: The user has tapped the input field to select it
- KEYBOARD_NEXT: The input field got selected automatically, because the next key was pressed when editing previous input field. (Also, see the “Next Input Field” property)
- PROGRAMMATIC_SELECT: The input field got selected, because some code selected this input field

OnEndEdit(string result, EndEditReason reason): Event called when ending the text editing. Includes the current text as a string and the reason as a EndEditReason. EndEditReason can have following values:

- USER_DESELECT: The user deselected the input field because it tapped/focused some other UI element
- KEYBOARD_CANCEL: The cancel key on the keyboard was pressed to end the text editing
- KEYBOARD_DONE: The done key on the keyboard was pressed to end the text editing
- KEYBOARD_NEXT: The next key on the keyboard was pressed to end the text editing (and select the next input field)
- PROGRAMMATIC_DESELECT: The input field got deselected, because some code deselected this input field

OnValueChanged(string text): Event called when the text has changed. Included the current text as a string

OnCaretPositionChanged(int caretPosition): Event called when the caret position has changed. Includes the caretPosition as an integer.

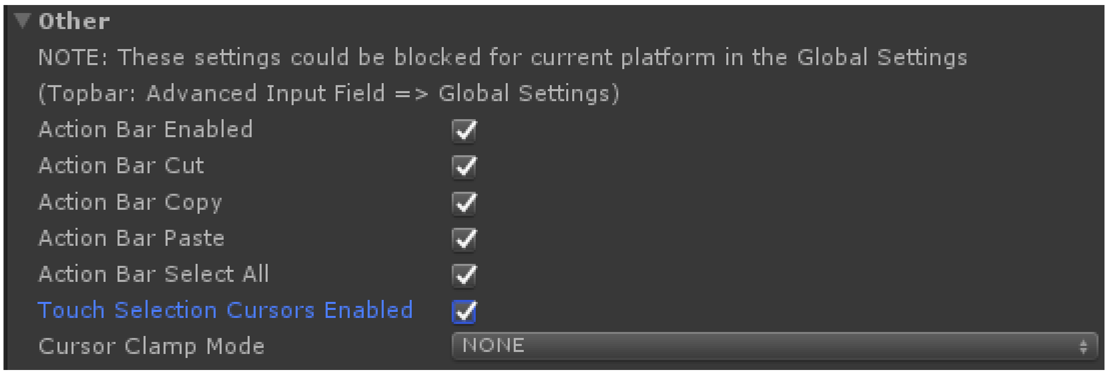
OnTextSelectionChanged(int selectionStartPosition, int selectionEndPosition): Event called when the selected text has changed. Includes the selectionStartPosition and selectionEndPosition as integers.

OnSizeChanged(Vector2 size): Event called when the size of the input field. Includes the current size as a Vector2. This only gets called when using one of the resize modes of the “Mode” property.

OnSpecialKeyPressed(SpecialKeyCode specialKeyCode): Event called when a special key has been pressed on the keyboard. Included the special key pressed as a SpecialKeyCode. SpecialKeyCode can have following values:

- BACK: The back key/button of Android
- BACKSPACE: The backspace key of the keyboard
- ESCAPE: The escape key of the keyboard

Other

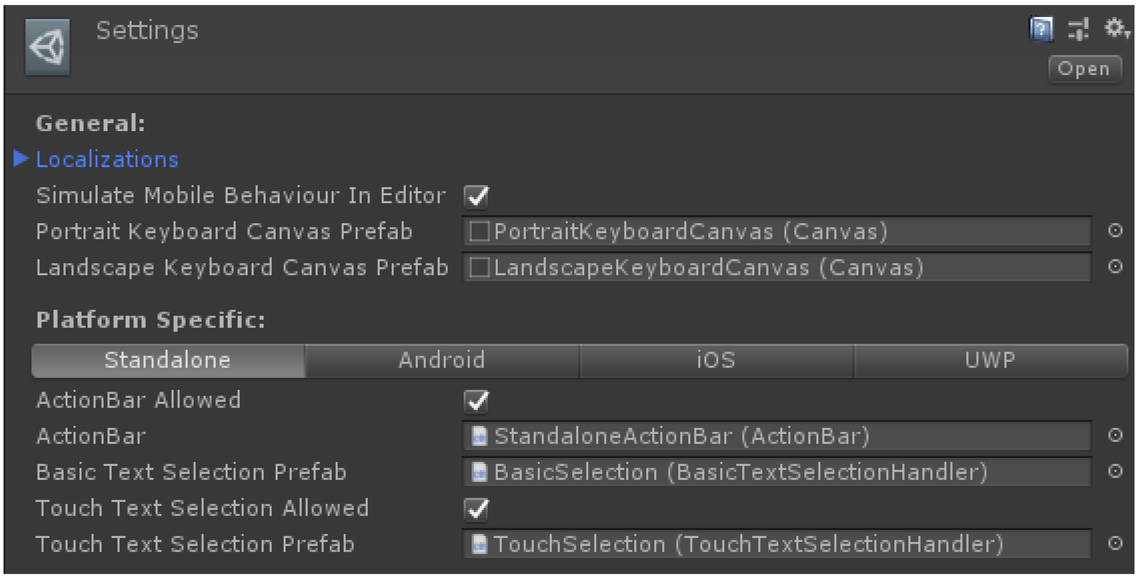

Other

NOTE: These settings could be blocked for current platform in the Global Settings (Topbar: Advanced Input Field => Global Settings)

Action Bar Enabled	✓
Action Bar Cut	✓
Action Bar Copy	✓
Action Bar Paste	✓
Action Bar Select All	✓
Touch Selection Cursors Enabled	✓
Cursor Clamp Mode	NONE

The settings can be blocked for current platform in the Global Settings. The Global Settings can be accessed from the TopBar: Tools => Advanced Input Field => Global Settings.

Global settings:



General:

Localizations

Simulate Mobile Behaviour In Editor	✓
Portrait Keyboard Canvas Prefab	PortraitKeyboardCanvas (Canvas)
Landscape Keyboard Canvas Prefab	LandscapeKeyboardCanvas (Canvas)

Platform Specific:

ActionBar Allowed	✓
ActionBar	StandaloneActionBar (ActionBar)
Basic Text Selection Prefab	BasicSelection (BasicTextSelectionHandler)
Touch Text Selection Allowed	✓
Touch Text Selection Prefab	TouchSelection (TouchTextSelectionHandler)

If you toggle “ActionBar Allowed” or “Touch Text Selection Allowed” to OFF here for a specific platform, they will be blocked for that platform even if the “ActionBar Enabled” or “Touch Selection Cursors Enabled” are set to ON on a specific AdvancedInputField instance.

Action Bar Enabled: Indicates if the ActionBar should be used

Action Bar Cut: Indicates if the cut option should be enabled in the ActionBar. If “InputType” property is set to “PASSWORD” this option will be blocked automatically.

Action Bar Copy: Indicates if the copy option should be enabled in the ActionBar. If “InputType” property is set to “PASSWORD” this option will be blocked automatically.

Action Bar Paste: Indicates if the paste option should be enabled in the ActionBar

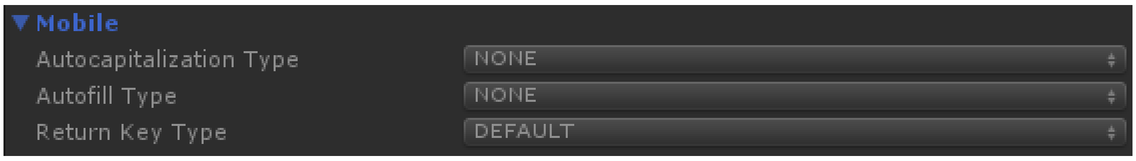
Action Bar Select All: Indicates if the select all option should be enabled in the ActionBar

Touch Selection Cursors Enabled: Indicates if the Touch Selection Cursors (handles for selection start and end) should be used

Cursor Clamp Mode: The clamp mode of the touch selection cursors. Following values can be used:

- NONE: No clamping, you can drag the cursors everywhere on the screen
- TEXT_BOUNDS: Keep the cursors within the text bounds
- INPUTFIELD_BOUNDS: Keep the cursors within the input field bounds

Mobile


▼ Mobile

Autocapitalization Type	NONE
Autofill Type	NONE
Return Key Type	DEFAULT

The properties only apply to the mobile platforms (Android & iOS).
NOTE: These properties are basically hints to tell the native input code to do something. That means that they should be applied, but some devices might not support them or simply ignore them.

Autocapitalization Type: The type of autocapitalization. Following values can be used:

- NONE: No autocapitalization is applied
- CHARACTERS: Autocapitalize new characters
- WORDS: Autocapitalize the first letter of new words
- SENTENCES: Autocapitalize the first letter of a new sentence

Autofill Type: The type of autofill. Following values can be used:

- NONE: No autofill suggestions should be shown
- USERNAME: Show username suggestions
- PASSWORD: Show password suggestions
- NEW_PASSWORD: Show new password suggestions
- ONE_TIME_CODE: Show one time code suggestions (Also, see the OneTimeCode sample scene)

Return Key Type: The type of return key to display on the keyboard. Following values can be used:

- DEFAULT: Show the default key
- GO: Show the go key
- SEND: Show the send key
- SEARCH: Show the search key

AdvancedInputField properties

Most of the properties visible in the inspector can be modified from code too.
There are some properties only accessible from code:

RectTransform: The RectTransform of the InputField

Size: The size of the InputField RectTransform

TextAreaTransform: The RectTransform of the TextArea. The TextArea is the viewport of the actual rendered text.

TextContentTransform: The scrollable content RectTransform of the actual text rendered text.

TextRenderer: The main text renderer

ProcessedTextRenderer: The text renderer for processed text (when using Live or PostDecorationFilters)

PlaceholderTextRenderer: The text renderer used as placeholder

VisiblePassword: Indicates whether the password should be visible. Can be used to make a password visibility toggle

CanUseActionBar: Indicates if the ActionBar can be used on this inputfield on this platform

CanUseTouchSelectionCursors: Indicates if the Touch Selection Cursors can be used on this inputfield on this platform

Initialized: Indicates if this input field is initialized

ShouldBlockDeselect: Indicates whether next deselect event should be blocked. This can be used to automatically reselect the Input Field when the Unity EventSystem wants to focus some other Selectable. (for example: a Button). Don't forget to set this property back to false when you are done editing the Input Field.

HasSelection: Indicates if some text is currently selected

SelectedText: The selected text string

RenderedText: The text that is actually rendered

ActionBar: The ActionBar if enabled

Selected: Indicates if the input field is currently selected/focused

AutoCorrection: Indicates if autocorrection should be used

Secure: Indicates is input should be secure

Multiline: Indicates if line type is multiline

HasNext: Indicates if a next inputfield has been set

LiveProcessing: Indicates if LiveProcessing is active

LiveDecoration: Indicates if LiveDecoration is active

PostDecoration: Indicates if PostDecoration is active

ShouldSubmit: Indicates if the Enter/Done key should submit

IsPasswordField: Indicates if input field is a password field

AdvancedInputField methods

Clear(): Clears the InputField text

ManualSelect(): Manually selects this inputfield (call this to select this inputfield programmatically)

ManualDeselect(): Manually deselects this inputfield (call this to deselect this inputfield programmatically)

UpdateCaretPosition(): Updates the visual position of the caret

GetCaretTransform(): Gets the RectTransform of the caret (when the input field is selected)

Following methods only have an effect when “Rich Text Editing” is enabled:
ToggleBold(): Toggles bold in current text selection

ToggleItalic(): Toggles italic in current text selection

ToggleLowercase(): Toggles lowercase in current text selection

ToggleNonBreakingSpaces(): Toggles non-breaking spaces in current text selection

ToggleNoParse(): Toggles no parse in current text selection

ToggleStrikethrough(): Toggles strikethrough in current text selection

ToggleSmallCaps(): Toggles small caps in current text selection

ToggleSubscript(): Toggles subscript in current text selection

ToggleSuperscript(): Toggles superscript in current text selection

ToggleUnderline(): Toggles underline in current text selection

ToggleUppercase(): Toggles uppercase in current text selection

ToggleAlign(string parameter): Toggles align with given parameter in current text selection

ToggleAlpha(string parameter): Toggles alpha with given parameter in current text selection

ToggleColor(string parameter): Toggles color with given parameter in current text selection

ToggleCharacterSpace(string parameter): Toggles character space with given parameter in current text selection

ToggleFont(string parameter): Toggles font with given parameter in current text selection

ToggleIndent(string parameter): Toggles indent with given parameter in current text selection

ToggleLineHeight(string parameter): Toggles line height with given parameter in current text selection

ToggleLineIndent(string parameter): Toggles line indent with given parameter in current text selection

ToggleLink(string parameter): Toggles link with given parameter in current text selection

ToggleMargin(string parameter): Toggles margin with given parameter in current text selection

ToggleMark(string parameter): Toggles mark with given parameter in current text selection

ToggleMaterial(string parameter): Toggles material with given parameter in current text selection

ToggleMonospace(string parameter): Toggles monospace with given parameter in current text selection

TogglePosition(string parameter): Toggles position with given parameter in current text selection

ToggleSize(string parameter): Toggles size with given parameter in current text selection

ToggleStyle(string parameter): Toggles style with given parameter in current text selection

ToggleVerticalOffset(string parameter): Toggles vertical offset with given parameter in current text selection

ToggleWidth(string parameter): Toggles width with given parameter in current text selection

TextMeshPro

Using TextMeshPro Text Renderers

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

Unity 2018.4:

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)

6*. If it still doesn't work, it probably means that the Unity Package manager didn't add the reference to TextMeshPro properly to Visual Studio (this unfortunately sometimes happen). You can check if it got added correctly by trying to access a class from TextMeshPro in code. Removing and reinstalling TextMeshPro from the PackageManager usually works.

7*. If the TMProTextRenderer still doesn't load, try the option "reimport all"

8*. If you want to use TextMeshPro 1.5.* you also need to add "TEXTMESHPRO_1_5" to the "Scripting Define Symbols".

Unity 2019.1 or newer:

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.
   (The assembly definition file of the plugin should automatically add the necessary defines)

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

Emojis (requires TextMeshPro Text Renderers)

To allow emojis enable the “Emojis Allowed” property on the AdvancedInputField instance. Also, assign an emoji asset file to the “Default Sprite Asset” property in Project Settings => TextMeshPro in the Unity Editor.
Due the licensing issues of emoji spritesheets, there isn’t an emoji asset included by default in the plugin. See following section for more information on how to create an emoji asset file.

Creating an emoji asset file:
Follow the following steps to create you own emoji asset file:

1. Download the emoji.json file and the spritesheet(s) you want to use from:
   https://github.com/iamcal/emoji-data
   Please note the licenses for the different spritesheets if you want to use them in your final product. (See “Image Sources” in above GitHub repo)
2. Import them in your Unity project.
   For example: Assets/Emoji/emoji.json file as TextAsset and Assets/Emoji/sheet_google_32.png as Sprite
3. In the TopBar of the Unity Editor click on: Tools => Advanced Input Field => Emoji Asset Generator
4. Reference the json file in the “Emoji list” field
5. Reference the spritesheet file in the “Spritesheet” field
6. Optionally change the “Grid size”, “Padding” and “Spacing” when the default values are not correct for configured spritesheet. The default values are configured for 32x32 pixels emojis. So, for example when you use the spritesheet: sheet_google_32.png or sheet_twitter_32.png
7. Press the “Generate Emoji Asset” button and save the result somewhere in your project directory.
8. Assign the emoji asset to the “Default Sprite Asset” property in Project Settings => TextMeshPro in the Unity Editor.
9. Optionally assign different spritesheets for the different platforms in the platform settings of the Emoji Asset.
10. (The sample scene: TextMeshPro/Chat should work now after assigning the emoji asset in the TextMeshPro Settings)

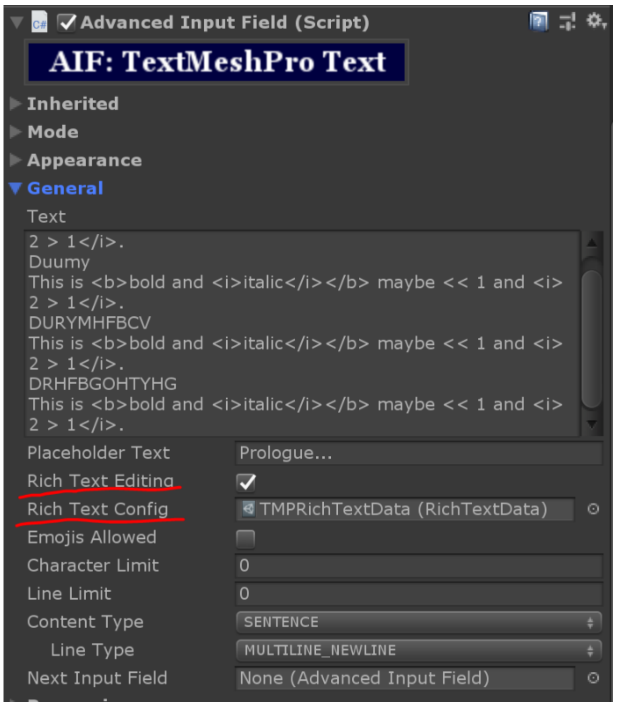
Rich Text

General

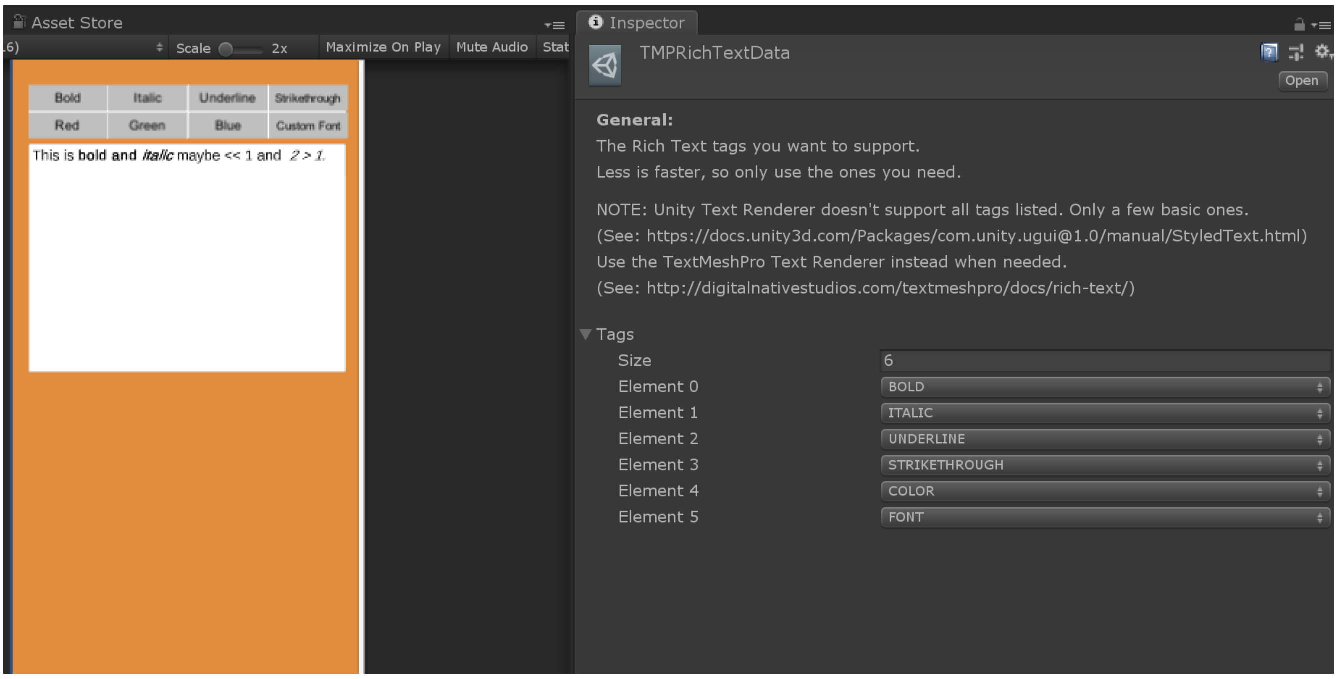
You can use Rich Text tags with both the Unity and TextMeshPro Text Renderers, although the Unity Text renderers support only a few Rich Text tags.
See the following documentation pages:
Unity Text Renderer:
https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/StyledText.html
TextMeshPro Text Renderer: http://digitalnativestudios.com/textmeshpro/docs/rich-text/

At the moment only tag pairs are supported. This means that there needs to be a start and a corresponding end tag for each effect. Tag pairs with no parameter, such as bold (<b>test</b> => **test**) and tag pairs with one parameter, such as color (<color=red>test</color> => test) are supported.

Basic setup


Advanced Input Field (Script)
AIF: TextMeshPro Text
Inherited
Mode
Appearance
General
Text
```
2 > 1</i>.
Duumy
This is <b>bold and <i>italic</i></b> maybe << 1 and <i>
2 > 1</i>.
DURYMHFBCV
This is <b>bold and <i>italic</i></b> maybe << 1 and <i>
2 > 1</i>.
DRHFBGOHTYHG
This is <b>bold and <i>italic</i></b> maybe << 1 and <i>
2 > 1</i>.
```
Placeholder Text	Prologue...
Rich Text Editing	✓
Rich Text Config	TMPRichTextData (RichTextData)
Emojis Allowed	
Character Limit	0
Line Limit	0
Content Type	SENTENCE
Line Type	MULTILINE_NEWLINE
Next Input Field	None (Advanced Input Field)

Support for Rich Text can be enabled on an AdvancedInputField instance by setting the property “Rich Text Editing” to enabled.
Also, there needs to be config file assigned to the “Rich Text Config” property. This config will specify which Rich Text tags you want to support on this instance.
You can create a new config file using the following TopBar option:
Tools => AdvancedInputField => Create => RichTextData



This is **bold** and *italic* maybe << 1 and *2 > 1*

TMPRichTextData

General:
The Rich Text tags you want to support.
Less is faster, so only use the ones you need.

NOTE: Unity Text Renderer doesn't support all tags listed. Only a few basic ones.
(See: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/StyledText.html)
Use the TextMeshPro Text Renderer instead when needed.
(See: http://digitalnativestudios.com/textmeshpro/docs/rich-text/)

Tags

Size	6
Element 0	BOLD
Element 1	ITALIC
Element 2	UNDERLINE
Element 3	STRIKETHROUGH
Element 4	COLOR
Element 5	FONT

This will create a file that looks like the above in the inspector:
In the “Tags” property you can specify which tags you want to support. Less tags to support will be faster/better for performance, so only add the ones you actually need.
When you’re done configuring this file, you can assign it to the “Rich Text Config” of the AdvancedInputfield instance.

In code you can toggle an effect on current text selection using the “Toggle” methods on an AdvancedInputField instance.
For example:
- inputField.ToggleBold() will toggle the bold effect on the currently selected text in the inputfield.
- inputField.ToggleColor(“red”) will toggle the “red” color effect on the currently selected text in the inputfield.

Also, see the sample scenes “TextMeshPro/RichText” or “RichText” in the Samples directory of the plugin if you’re still unsure how to configure Rich Text properly.

Autofill (Android & iOS)

Using the autofillType property of an AdvancedInputField instance you can specify what kind of text that inputfield expects from an autofill service. Please note that this is a hint value and some autofill services might not provide suggestions for it.
(See the “Autofill” sample scene for an example implementation)

There are also a few methods related to the autofill functionality:

- **NativeKeyboardManager.ResetAutofill():** Resets the AutofillManager state (Android only). Sometimes if an autofill suggestion has already has been applied for a specific autofill type on Android it will not show new suggestions the next time you select it. You can call this method to reset it. For example: When returning to the LoginScreen after logging out.
- **NativeKeyboardManager.SaveCredentials(string domainName):** Saves the current text values of the inputfields with the username and password autofill type set as login credentials.
  The domain name parameter is only required for iOS and should be the same as the configured associated domain (see Domain setup for more info).

Domain setup

iOS:
View the official documentation of iOS to configure your domain for your app:
https://developer.apple.com/documentation/safariservices/supporting_associated_domains
Basically there are 2 important steps: Creating & hosting the association file and adding the domain as an Associated Domain Capability in Xcode.

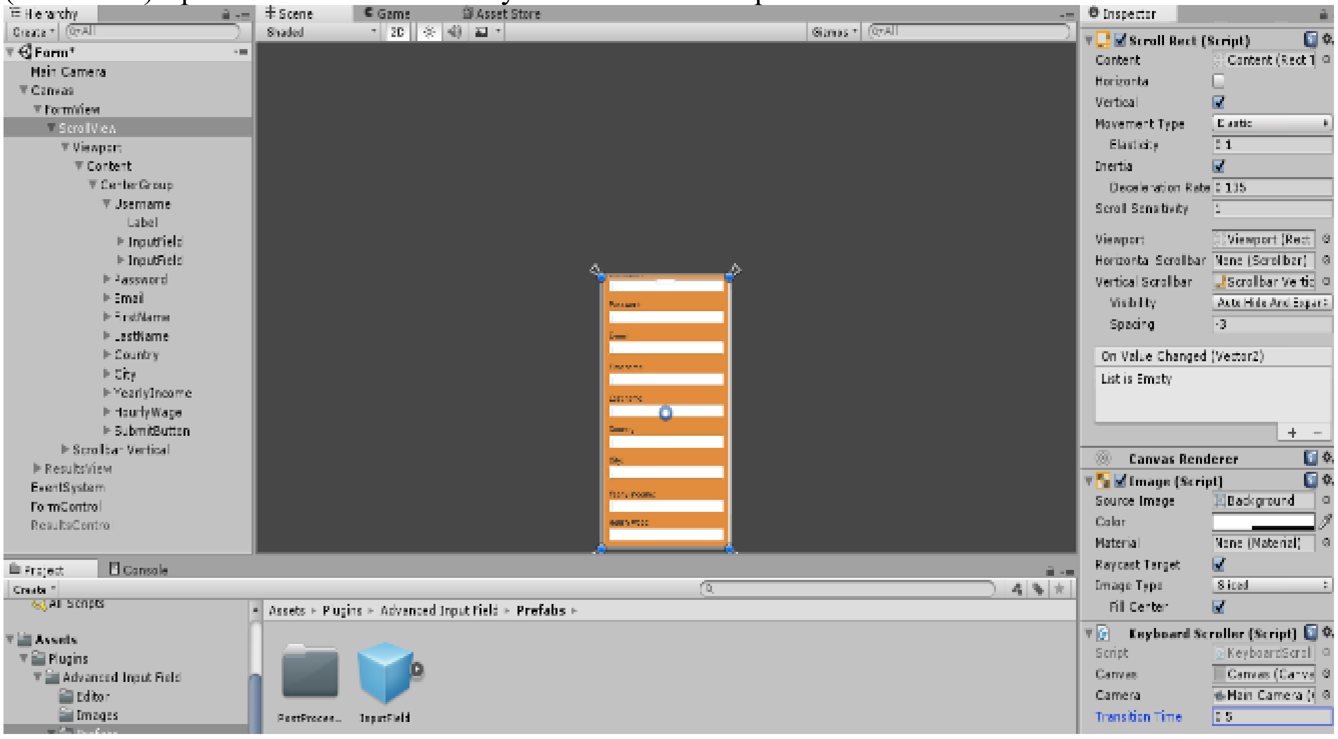
Android:
While it is not required to setup a domain to store credentials, you can find the documentation to do this here:
https://developers.google.com/identity/smartlock-passwords/android/associate-apps-and-sites

Known issues

On iOS username and password fields are not filled in at the same time. You have to tap the username and password fields separately and select the login credential to use.
The native iOS autofill service tries to guess if a login view is shown to determine if it should fill in both fields, but I guess something in Unity code is blocking this feature.

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.



Inspector

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 Expan
Spacing	-3

On Value Changed (Vector2)
List is Empty

Canvas Renderer

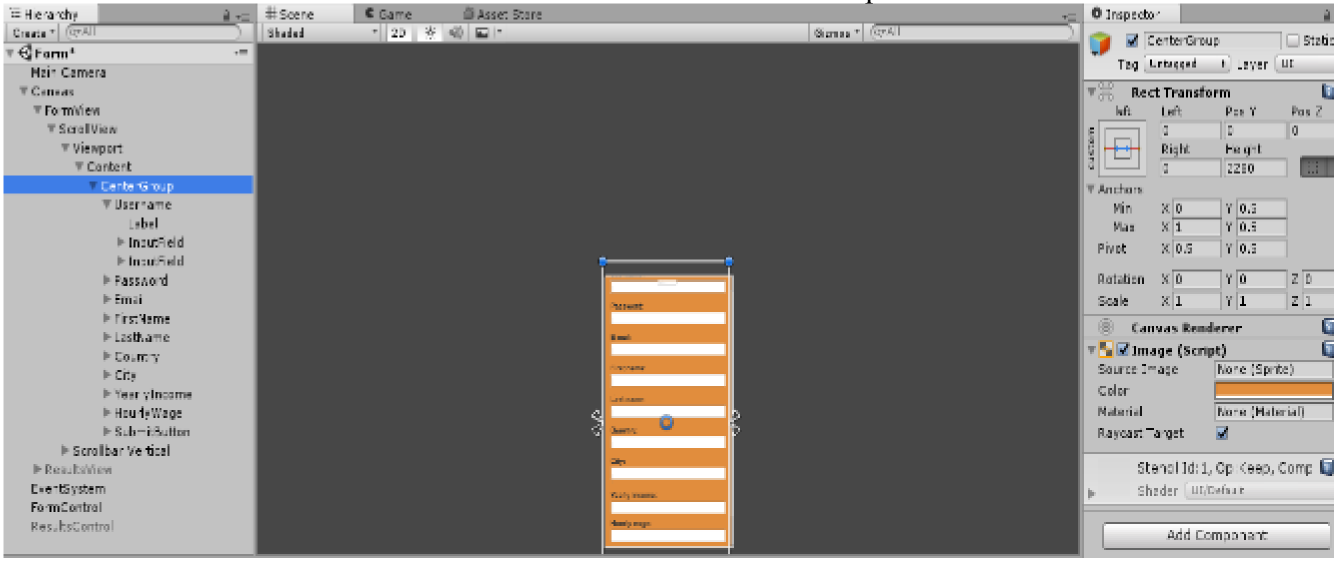
Image (Script)

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

Keyboard Scroller (Script)

Script	KeyboardScrol
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.



Inspector

CenterGroup ✓ — Static ☐ — Tag Untagged — Layer UI

Rect Transform (custom)

Left	Pos Y	Pos Z
0	0	0
Right	Height	
0	2250	

Anchors

	X	Y	Z
Min	0	0.5	
Max	1	0.5	
Pivot	0.5	0.5	
Rotation	0	0	0
Scale	1	1	1

Canvas Renderer

Image (Script)

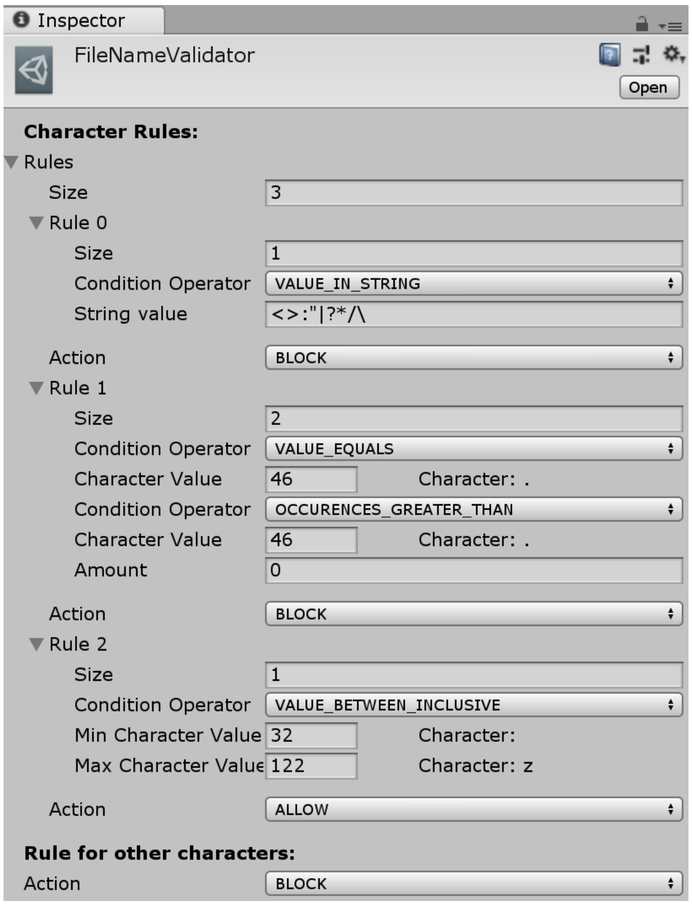
Source Image	None (Sprite)
Color	🟧
Material	None (Material)
Raycast Target	✓

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.
Also, there is a helper component called “SyncContentTransform” to automatically sync the “CenterGroup” and “ContentTransform” sizes when changed.

Using (Custom) CharacterValidators



```
Inspector
FileNameValidator                                   Open

Character Rules:
▼ Rules
    Size                         3
  ▼ Rule 0
      Size                       1
      Condition Operator         VALUE_IN_STRING
      String value               <>:"|?*/\

    Action                       BLOCK
  ▼ Rule 1
      Size                       2
      Condition Operator         VALUE_EQUALS
      Character Value            46          Character: .
      Condition Operator         OCCURENCES_GREATER_THAN
      Character Value            46          Character: .
      Amount                     0

    Action                       BLOCK
  ▼ Rule 2
      Size                       1
      Condition Operator         VALUE_BETWEEN_INCLUSIVE
      Min Character Value        32          Character:
      Max Character Value        122         Character: z

    Action                       ALLOW

Rule for other characters:
Action                           BLOCK
```

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

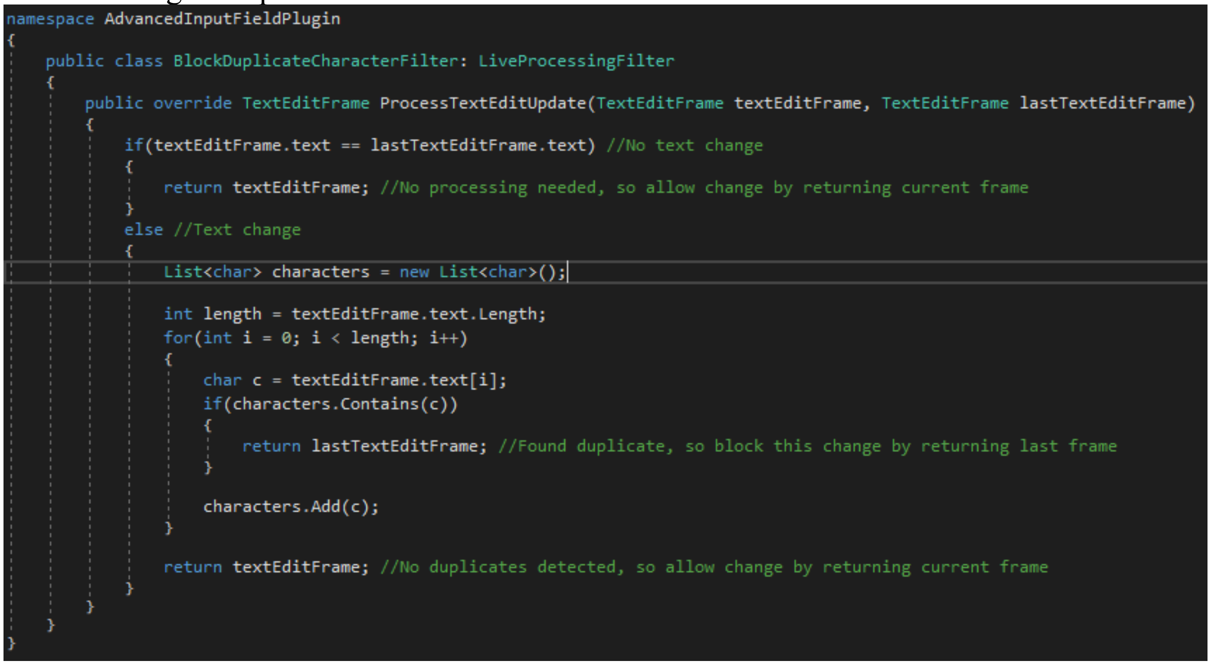
- If none of the rules apply, then the action defined at "Rule for other characters" will be executed

<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
}
```

Using Live Processing Filters

Live processing filters can be used as a kind of man in the middle between the NativeKeyboard and the AdvancedInputField instance. All modifications in native code are send to Unity code using the TextEditFrame format. TextEditFrames hold a text value and values for the selection start position and selection end position. You can use these LiveProcessingFilters to allow, block or modify the TextEditFrame that actually gets processed by the AdvancedInputField.
When using a LiveProcessingFilter you will get a callback with last TextEditFrame and current TextEditFrame.
See following example:



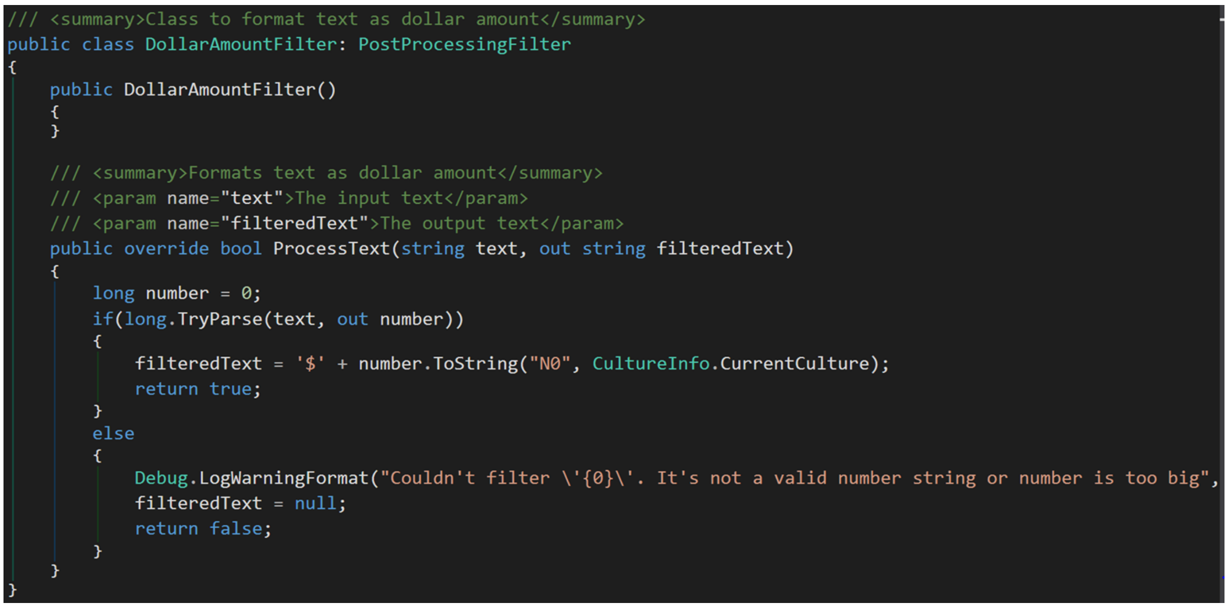
```
namespace AdvancedInputFieldPlugin
{
    public class BlockDuplicateCharacterFilter: LiveProcessingFilter
    {
        public override TextEditFrame ProcessTextEditUpdate(TextEditFrame textEditFrame, TextEditFrame lastTextEditFrame)
        {
            if(textEditFrame.text == lastTextEditFrame.text) //No text change
            {
                return textEditFrame; //No processing needed, so allow change by returning current frame
            }
            else //Text change
            {
                List<char> characters = new List<char>();

                int length = textEditFrame.text.Length;
                for(int i = 0; i < length; i++)
                {
                    char c = textEditFrame.text[i];
                    if(characters.Contains(c))
                    {
                        return lastTextEditFrame; //Found duplicate, so block this change by returning last frame
                    }

                    characters.Add(c);
                }

                return textEditFrame; //No duplicates detected, so allow change by returning current frame
            }
        }
    }
}
```

The filter in above example will block duplicate characters. First it checks if the text has changed. If it has not changed, it will allow the modification by returning the current TextEditFrame. If it is has changed it will check if the current text has duplicates. When a duplicate character has been found it will block the modification by returning last TextEditFrame. If no duplicate has been found it will the allow the modification by returning the current TextEditFrame.
So, basically you can use these filters when you need as a more advanced text validator when (Custom) CharacterValidators doesn’t suffice (since CustomCharacterValidators work per character).
To create your own filter create a new code file to inherits from LiveProcessingFilter and override the ProcessTexEditUpdate() method.

NOTE: RichText or Emoji support rely heavily on detecting changes that have happened in native code, so they could start acting strange when making large changes in the returned TextEditFrame when using these filters.

Using Post Decoration Filters

Post decoration 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/PostDecoration directory (see

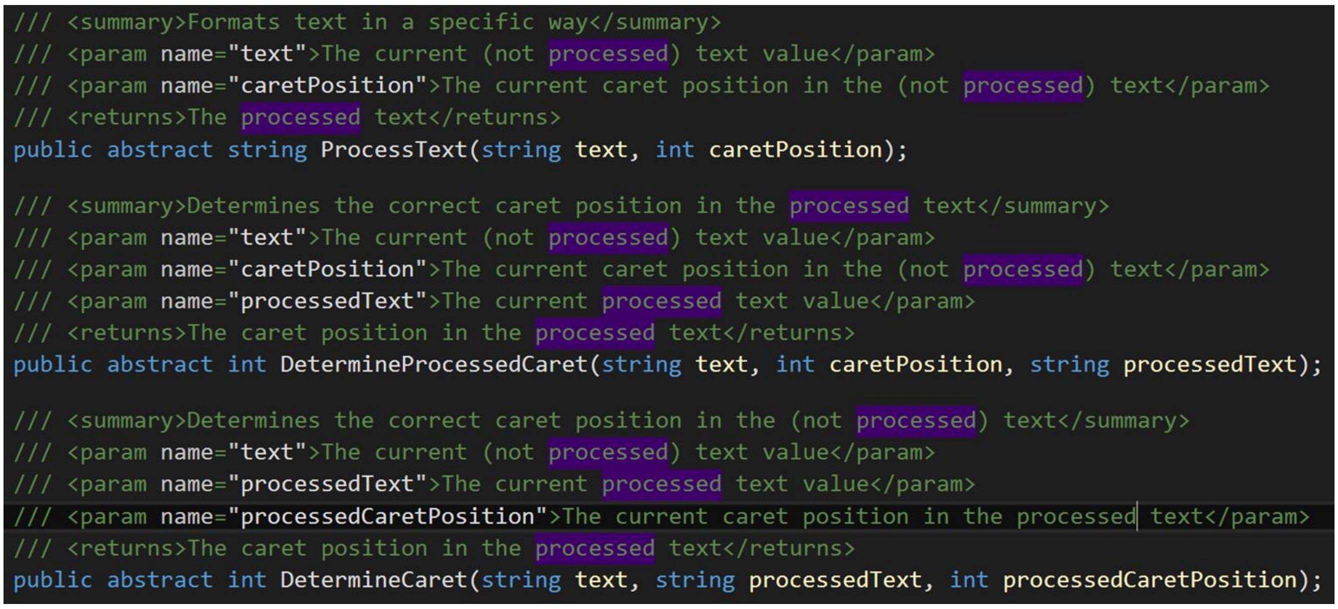


```
/// <summary>Class to format text as dollar amount</summary>
public class DollarAmountFilter: PostProcessingFilter
{
    public DollarAmountFilter()
    {
    }

    /// <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;
        }
    }
}
```

DollarAmountFilter.cs).
To create your own filter create a new code file to inherits from PostDecorationFilter 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 Decoration Filter” field of a Advanced Input Field.

Using Live Decoration Filters

Live Decoration filters will format the text in the InputField while an InputField is selected/in edit mode). Just as Post Decoration 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 Decoration 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 LiveDecorationFilter 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 Decoration Filter” field of a Advanced Input Field.

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

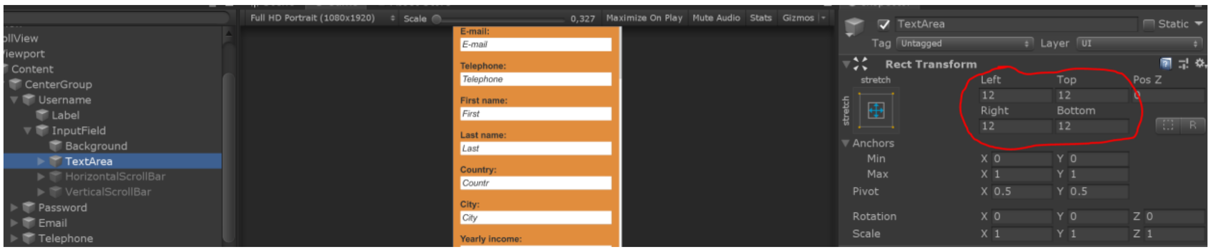
Frequently asked questions

- **How to get the keyboard height on mobile platforms?**
  You can register to the OnKeyboardHeightChanged event using the method:
  NativeKeyboardManager.AddKeyboardHeightChangedListener();
  The signature for the event callback is: OnKeyboardHeightChanged(int keyboardHeight).

  <u>For example:</u>

  ```
  NativeKeyboardManager.AddKeyboardHeightChangedListener(OnKeyboardHeightChanged);

  public void OnKeyboardHeightChanged(int keyboardHeight)
  {
          float keyboardHeightInCanvas = keyboardHeight / Canvas.scaleFactor; //keyboard
  height is in Screen pixels, so divide it by the Canvas ScaleFactor

  }
  ```

- **Can it get the keyboard height without showing the keyboard on mobile platforms?**
  No, the native SDK for Android & iOS don't provide an easy way to get the keyboard height. The way the OnKeyboardHeightChanged event works is by monitoring the difference in native view sizes when the keyboard appears/disappears.

- **How to change the padding between the AdvancedInputField RectTransform and the viewport of the actual rendered text?**
  

  ```
  TextArea
  Tag Untagged    Layer UI
  Left 12    Top 12    Pos Z 0
  Right 12   Bottom 12
  Anchors
    Min    X 0     Y 0
    Max    X 1     Y 1
  Pivot    X 0.5   Y 0.5
  Rotation X 0     Y 0    Z 0
  Scale    X 1     Y 1    Z 1
  ```

  You can modify the padding by changing the values in the anchor settings of the TextArea child object.

- **Is the plugin going to support native UI controls, such as the native ActionBar and native Selection Handles?**
  No, not on short term at least. Because the whole input field and text rendering is done in Unity, it's a lot more difficult to sync the position and sizes of native UI controls (especially when using ScrollRects). There also a small delay between the Unity main thread and the native Android/iOS threads. Also, the actual native input fields that are used internally are rendered offscreen and the native UI controls might want to stick to them.
  A positive point of rendering the controls in Unity is that you can customize the appearance a bit by modifying the ActionBar prefabs. (Accessible from the TopBar: Tools => Advanced Input Field => Global Settings). For example make the ActionBar use the same color scheme as the rest of your app.

- **How to keep the AdvancedInputField focused when clicking on other UI Elements?**
  There are few way to counteract the default deselect/unfocus behaviour of the Unity EventSystem when using AdvancedInputFields. Basically the plugin delays the deselect call of the Unity Event System by one frame and then checks if one of following conditions is met and reselects itself in those cases:

  1. Child object is selected: Make the Button a child object of the AdvancedInputField transform. And set the "Navigation" property of the Button to "Automatic". See the "Chat" sample scene for an example.

  2. InputFieldButton is selected: Use the InputFieldButton component instead of the Button component. This button doesn't need to be a child of the AdvancedInputField transform. This component is still included with the plugin, because projects with AdvancedInputField 1 might still use it, but it can be replaced by a gameobject with a Button component and a InputFieldRefocusable component. See next option.

  3. InputFieldRefocusable is selected: Instead of using the custom Button script “InputFieldButton”, you can also add a InputFieldRefocusable component to a gameobject with one of the default Selectable components on it. For example on a gameobject that uses the default Unity Button component.

  4. .ShouldBlockDeselect property is set to true: Set the .ShouldBlockDeselect property of an AdvancedInputField instance to true. When this flag is set, the AdvancedInputField instance will automatically reselect itself when another UI element got focused/selected. Don't forget to set this property back to false when you are done editing the text. For example when moving to a next screen.

- **How can I hold press an empty AdvancedInputField to show paste action without clicking on it first?**
  You’ll need to set the SelectionMode property to “SELECT_ON_PRESS” for this to work without clicking on it first. (Also enable the ActionBar as you already normally would do)

- **How can I input and render Asian glyphs (for example: Japanese, Chinese, Korean) on Android & iOS?**
  NOTE: This is currently only supported on Android & iOS (the plugin uses the IME that the mobile OS already provides). Asian IME on Standalone are not supported/tested.

  To configure them for Android & iOS:
  1. Download an font file that supports the characters you need and import it in Unity
  2.a When using Unity Text Renderers: Font asset should work as is
  2.b When using TextMeshPro Text Renderers: Create a dynamic font file. This can be done by right clicking on the font file in the Unity Editor and selecting “Create” => “TextMeshPro” => “Font Asset”
  3. Assign the font asset on the TextRenderers of the AdvancedInputField. These can be found on the child objects “Text”, “Processed” & “Placeholder”.
  4. Make a build and run it on your Android or iOS device

- **How can I change the appearance of the ActionBar?**
  The ActionBar prefabs that will be used on the different platforms can be found and changed in the Global Settings (TopBar => Tools => AdvancedInputField => Global Settings).
  You could modify the existing ActionBar prefabs, but it’s better to duplicate them and then assign your custom ActionBar prefabs in the Global Settings. This way your changes are not overwritten when applying a plugin update.
  The default prefab settings are configured to render properly on a Canvas with a target resolution of 1080p. If you are using different Canvas settings and the sizes of the ActionBar items don’t look nice, you can change the preferred font size, min item width, padding,… on the ActionBar prefabs.

Advanced

Custom keyboard events handler

If you want to handle the keyboard events from the native keyboards yourself (without using the AdvancedInputField component) you can make your own KeyboardClient subclass. The KeyboardClient base class provides several methods for the callbacks of the different types of events, such as OnKeyboardShow, OnKeyboardHide, OnTextEditUpdate. The are marked as virtual and have not implementation by default. In the subclass you can override the ones you want to implement. See InputFieldKeyboardClient for an example implementation.
Because the native Android/iOS UI thread is not the same as the Unity main thread the native callbacks are put in a threadsafe Queue. In the Update() method of the KeyboardClient base class the entries in that queue are processed and the corresponding callback method is called, such as OnKeyboardShow(). The KeyboardClient also has an Activate() and Deactivate() method to start and stop processing the event queue. To make sure you are not processing old events you can call ClearEventQueue() method whenever starting/stopping the text editing.

Custom ActionBar actions

Aside from the default ActionBar actions (copy, cut, paste, select all) you can specify additional actions. For replace text (current selected word) actions you use the UpdateReplaceActions(List<ActionBarAction> actions) method of the ActionBar. (The ActionBar can be accessed with the .ActionBar property of an AdvancedInputField instance). For completely custom actions you can use the UpdateCustomActions(List<ActionBarAction> actions) method of the ActionBar. When creating an ActionBarAction (with ActionBarType.CUSTOM) you can specify a method to be called when the user clicks on that action. See the SpellChecker sample scene for an example implementation.

Changelog

2.0.9:

- Bugfix: Fixed issue that the text edit state was not getting synced properly with native code when reselecting an input field
- Change: Split Decimal character validation into DECIMAL and DECIMAL_FORCE_POINT. DECIMAL will allow both dot and comma character as the decimal separator and DECIMAL_FORCE_POINT will automatically convert comma characters to a decimal point/dot character
- Bugfix Android: Fixed issue on Android that keyboard was not closing when pausing app
- Bugfix Android: Fixed issue that keyboard restore didn't work correctly on some Android devices after resuming app
- Bugfix Editor: Fixed SimulatorKeyboard error when selecting InputField (from code) immediately in Awake()

2.0.8:

- Bugfix: Fixed several ActionBar sizing issues on Tablets
- Feature: Added IconButtonPadding and MinItemWidth properties on ActionBar
- Change: Changed default InputField prefab settings, so the advanced options are not enabled by default

2.0.7:

- Improvement: Added fallback if ActionBar text & size optimalization fails
- Bugfix: Fixed scalefactor issue of get preferred width/height methods (used for ActionBar text & size optimalization)
- Feature: Added SELECT_ALL option for CaretOnBeginEdit property
- Bugfix: Fixed issue when dragging text selection when using a LiveProcessingFilter
- Bugfix: Fixed race condition between Unity and native thread when rapidly typing or deleting characters

2.0.6:

- Bugfix: Fixed text not scrolling when dragging start touch cursor when using RichText or emoji support
- Bugfix: Small fix to get latest ShouldBlockDeselect value again after OnEndEdit event
- Bugfix: Fixed issue that putting app in background and back to foreground was autoselecting last inputfield when it was not focused when putting app in background
- Bugfix: When using the .BlockShouldDeselect property the keyboard type changes when tapping another inputfield
- Improvement: Improved autofill support for iOS & Android
- Feature: Implemented extra autofill types for iOS & Android
- Feature: Added Save Credentials method for iOS & Android (using autofill fields)
- Improvement: Improved algorithm to optimize ActionBar item and text sizes (preferred font size, text padding,… can be changed on the ActionBar prefabs)

2.0.5:

- Bugfix: Fixed text selection changes caused by arrow keys on Android soft keyboard are not detected
- Bugfix: Fixed line limit property not working properly
- Feature: Added ResizeMaxWidth and ResizeMaxHeight properties to be used with the resize modes. Will automatically start scrolling the text when max width/height has been reached

2.0.4:

- Bugfix: Fixed race condition in native iOS and Android code caused by long pressing backspace key
- Bugfix: Fixed error when calling RefreshOriginalContentSize() on KeyboardScroller when it's not initialized yet
- Change: Moved menu items under "Tools" in the TopBar
- Improvement: Added extra safety checks to ensure a new instance of the NativeKeyboardManager won't be created when some code is trying to access it when exiting play mode in the Editor

2.0.3:

- Feature: Added GetCaretTransform() method to get the RectTransform of the caret renderer
- Bugfix: Added extra null check in TMProTextRenderer in case there isn't a TMP_TextInfo object yet
- Improvement: Cleanup FrontRenderers from hierarchy view on deselect to avoid cluttering

2.0.2:

- Bugfix: Fixed ConversionTool not copying inherited properties
- Bugfix: Fixed arrow keys navigation on Standalone when not using emoji support
- Bugfix: Fixed selection handlers getting recreated on focus
- Bugfix: Fixed Readonly property still allowing keyboard input
- Bugfix: Fixed Editor asmdef file

2.0.1:

- Bugfix: Fixed skin variations emojis not getting included in emoji asset file
- Bugfix: Fixed PreferredSize when using autosize on TextRenderers
- Bugfix: Fixed copy action not hiding ActionBar
- Bugfix: Fixed hold to show ActionBar
- Change: Moved doubleTapThreshold & holdThreshold to the Global Settings of the plugin