viperOSK

Easy to use On-Screen Keyboard for 2D & 3D games

Guide Document v3.6
[2025-10-29]

The Package

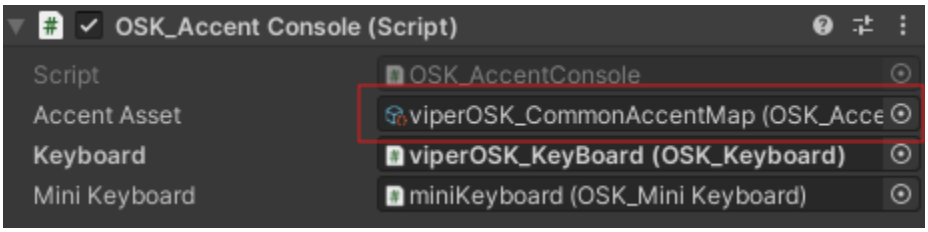
V3.6 Package includes the following:

- Ability to use glyphs in the keyboard layout which allows for non-Roman alphabet languages (ex: Greek, Arabic). You can simply type your keyboard layout using the glyphs directly. Such as: "ς ε ρ τ υ θ".
- To use glyph or script languages, a **OSK_LanguagePackage scriptable object** is needed, assign the scriptableObject to LanguageProfile in the OSK inspector (see Example 11).
- The **OSK_LanguagePackage** can handle multiple languages (up to 1000 glyphs) that are introduced using the standard 2 letter cultures under **Culture** property in the object, such as: "el" for Greek, "ar" for Arabic..etc.
- Ability to map your on-screen keyboard to physical keyboard keys under OSK_Settings component of the on-screen keyboard. Note: the developer will need to account for foreign keyboard layouts (ex, keyboards that are not QWERTY). The developer also must account for key placement in the OSK vs physical keyboard.
- Resizeable OSK to fit any arbitrary RectTransform using **OSK_Keyboard.ResizeKeyToFit(Vector2)**
- See **Example11** scene for a live example of both the glyph layout and resize. The example also demonstrates how OSK can dynamically adjust to fit portrait or landscape screen dimension.
- Fix: physical keyboards can now support long-presses. The long-press UnityEvent action can be set in OSK_Settings.
- Fix: Topleft child object under OSK_Keyboard now refers to the top-left corner of that button (used to be the top-middle).

V3.5 Package (or accents galore package!) includes the following:

- Capacity to include accented characters (like "é") in your layout and they will automatically generate the corresponding keys.
- Capacity to set actions for long presses. The action has to be set in script (see below).
- Added a singleton accessible under **OSK_Keyboard.Settings.instance**, this singleton allows access to set actions to long press, delay before a long press action is invoked.
- Switched from using Unity's KeyCodes as to a custom **OSK_KeyCode** enum that allows viperOSK to scale up with added symbols, accented characters and such

    - Presently, the enum has 18 derivations for every Latin alphabet letter (A to Z), such as A_01, A_02..etc
    - Accented characters are mapped to these enum entries in order of their appearance in the layout. For example, the following layout string: “à é ê” would be mapped to A_01, E_01 and E_02 respectively.
    - There are additional keycodes for symbols such as not equal, greater than or equal..etc. However, ensure your text asset can handle those entries where needed.
    - For developer created symbols, please enter them into OSK_Keymap.chartoKeycode to map them to your own KeyCodes.
    - NOTE: OSK_KeyCodes from 1000 to 4000 are reserved for accented characters.
    - OSK_KeyCode can be casted to Unity’s KeyCode for entries that are available in KeyCode (with value less than 504, note, accented characters would not map to KeyCode.).
    - Note, this change was necessary to increase viperOSK’s out-of-the-box use and capacity to scale this OSK to other languages in the future.
- Accented characters console:
    - Added **viperOSK_AccentedCharacters** Prefab, this prefab generates a mini-keyboard with accented variations of a key when long-pressed for the required delay (1 second is the default).
    - Adding this Prefab automatically sets the longPressAction in **OSK_Keyboard.Settings** to trigger a mini-Keyboard with the accented derivations.
    - The prefab works in 3D viperOSK and the UI version (must be placed as child of Canvas for UI scenes).
    - The prefab comes with a base list of accented characters based on the most common ones (at least based on our research). You can set your own easily though the **viperOSK Accented Characters Tool** (see below)
    - The prefab uses a ScriptableObject **OSK_AccentAssetObj** to load the accented derivations for base characters.
- Added **viperOSK Accented Characters Tool** accessible from the **Tools** menu where you can create your own dictionary of accented characters and save is as an **OSK_AccentAssetObj** scriptable object.
    - To load a saved accent object simply set it in the inspector (see image below) or through script: **OSK_AccentConsole.LoadAccentMap(OSK_AccentAssetObj accents)**



- Added capacity for loading **Language Packages**. **The language packages are currently limited to Latin languages**. Loading language packages requires the ScripableObject **OSK_LanguagePackage**
    - An **OSK_LanguagePackage** comprises of:

        - a keyboard layout,
        - an alternate keyboard layout (for switching to numbers and symbols keyboard), and
        - an **OSK_AccentAssetObj** is you require that capability
    - See **Example8UI** scene for an example of how this is done. Language loading is done through **Example8UI.cs** script. The same functionalities are available to the 3D version of viperOSK
- Added **OSK_MiniKeyboard**, this is the mini-keyboard used when a long-press is detected for the accent console. However, it is also very versatile and can be used on its own, see Example10.
    - Note however, that the Mini-Keyboard does not have a layout string. It organizes keys based on a table of rows and columns defined by **Dimensions** Vector2Int where x are rows and y are the keys per row.
    - The dimensions of the keyboard depend on how many keys it is given and will prioritize fitting keys along columns before rows
    - Mini-Keyboard organizes keys with the first being the bottom-right to the top-left (made so it can service the accent console). This can be changed by re-ordering the List<string> used as input.
    - Mini-Keyboard works in both 2D/3D and UI, the key prefab must match it's intended use.
    - The keys are created so that the pivot of the OSK_MiniKeyboard game object is in the middle
- Added **OSK_Background** component to the background of keyboard. This component resizes the sprite for the background to fit the parent keyboard.
- Known Issues and Limitations:
    - For now, physical keyboards cannot be used for long-presses.
    - OSK_GamepadHelper does not support long-presses.
    - If you have more than one viperOSK keyboard in a scene, each keyboard would require a viperOSK_AccentedCharacters prefab (remember to set the keyboard for the prefab to the respective OSK_Keyboard).

V3.4 Package includes the following:

- Keyboard and key prefabs
- Scripts: primarily **OSK_Keyboard** and **OSK_Key** to manipulate the keyboard layout and keys, several other keyboard helper scripts are included as well
- **OSK_Receiver** for the text object receiving keyboard strokes.
- **OSK_UI_Keyboard** and **OSK_UI_Key** to manipulate the keyboard layout and keys for a Unity UI implementation.
- **OSK_UI_InputField** to manage OSK input into Unity's UI InputField (TMP or TextMesh versions)
- **OSK_UI_CustomReceiver** to manage OSK input into a custom made UI field that works in Unity UI. ***This is the preferred input field to use in Unity UI***
- **[NEW 3.0]** a component **OSK_GamepadHelper** to better streamline gamepad support, including support for multiple controllers for on the same keyboard.
- Texture assets

- Examples: two example scenes with scripts that show full functioning of the on-screen keyboard
- **[NEW 3.4]** added backwards compatibility to issues being experienced in **Unity 2019.4**. NOTE: you may need to re-Generate keyboards in examples scenes when you import the viperOSK into Unity 2019.4 and recompile.
    - **Known Issues with Unity 2019.4 backward compatibility:**
    - **Example 3UI** scene require the following steps to work: 1) Generate the keyboard by clicking Generate in the Inspector; 2) Click **on Add Default Input Modules** under the EventSystem gameobject inspector.
    - **Example 5UI** regenerate the keyboard by clicking on Generate in the inspector (in the viperOSK_UI gameobject inspector).
- **[NEW 3.4]** added **Auto-Correct** features to the layout. You can use the Auto-Correct Layout button in the inspector to correct issues with the string layout of your OSK keyboard.

Compatibility

viperOSK is a powerful on-screen keyboard that can be used in any platform supported by Unity:

- PC/Mac [tested √]
- mobile (iOS & Android) [tested √]
- TVOS (AppleTV) [tested √]
- Consoles (PS4 & Xbox) [tested]

viperOSK can be fully configured from the Inspector without any need to adjust coding.

viperOSK uses Unity's inhouse TextMeshPro for all on screen text. Assets could easily be converted to plain TextMesh if users need [requires coding]

Requires **TextMeshPro**, **TMP Essential Resources** and **TMP Examples & Extras** packages. The packages can be installed through the Unity Editor Package Manager. They are accessible through Windows > Package Manager and Windows > TextMeshPro.

All source code (in C#) included!

Setup of viperOSK

Drag and drop **viperOSK_Keyboard** (or **viperOSK_UI_Keyboard** for a Unity UI implementation) from the prefab folder anywhere you would like the On-Screen Keyboard to be.

Where needed add a **OSK_Receiver** (or **OSK_UI_CustomReceiver**) to a TextMeshPro text object and set the keyboard pointer to it under "output".

OSK_Receiver requires a collider object (box collider would work).

For a cursor, add an **OSK_Cursor** to the **OSK_Receiver** object as child (see viperOSK_Keyboard prefab or Example scenes for implementation)

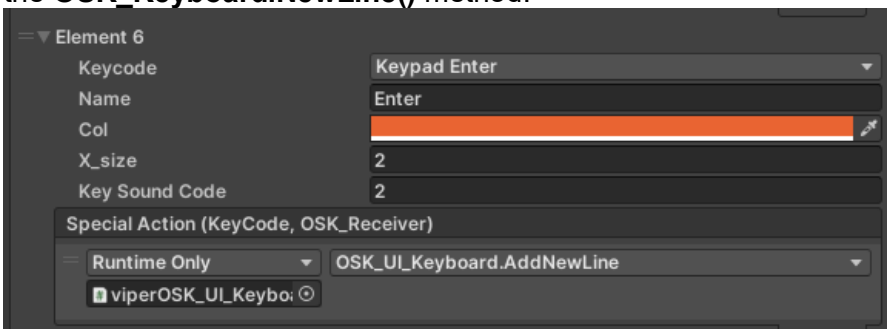
The backgrounds for keys and the keyboard can be changed at will by simply using your own texture or sprites.

The keyboard supports your own Fonts, although you must convert to a TextMeshPro Font Asset (*TMP_FontAsset*). This can be done by going through Windows > TextMeshPro > Font Asset Creator. The Creator allows you to import almost any font and produce a TMP_FontAsset

Releases	
Version	Release notes
3.5	• -Accents support: Automatically generates keys for accented characters (e.g., "é") from layout input. • -Added viperOSK_AccentedCharacters Prefab to display mini-keyboards with accented variations on long press. • -Supports customization through the viperOSK Accented Characters Tool. • Long Press Actions • -Set actions for long presses via OSK_Keyboard.Settings.instance. • -Configurable delay for long press actions (default: 1 second). • -Custom KeyCode System: Replaced Unity's KeyCodes with OSK_KeyCode enum for better scalability with symbols and accents. Reserved KeyCodes (1000–4000) for accented characters. • -Accented Characters Console: Includes a prefab for easy integration of mini-keyboards for accents in both 3D and UI scenes. • -Comes with a default dictionary of common accents, customizable via the viperOSK Accented Characters Tool (under Tools menu). • -Uses ScriptableObjects for accent data management. • - Ability to load language packages (for Latin alphabets). They are scriptable objects with a keyboard layout, alternate layout (to switch to symbols..etc), and an accent package for long-presses.
3.41	• minor fixes to OSK Cursor sometimes now showing • minor fixes to OSK_UI_InputReceiver to permit better use with OSK_UI_Cursor and greater control for OnFocus and OnLostFocus (previous version events were not triggered, this has been fixed) • Experimental OSK_Background, to auto-resize the background of the keyboard. This functionality is still in testing (happy to receive feedback).

3.4	• Added **Auto-Correct** features to the layout. You can use the **Auto-Correct Layout** button in the inspector to correct issues with the string layout of your OSK keyboard.
	• Added backward compatibility for developers using Unity 2019. See script labelled v3.4 for notes. No change to the functionality of viperOSK overall.
	• You may need to **Generate** the keyboard again for some Example scenes so that they compile with Unity 2019 specific scripts.
	  ○ **Known Issues with Unity 2019.4 backward compatibility:**
	  ○ **Example 3UI** scene require the following steps to work: 1) Generate the keyboard by clicking Generate in the Inspector; 2) Click **on Add Default Input Modules** under the EventSystem gameobject inspector.
	  ○ **Example 5UI** regenerate the keyboard by clicking on Generate in the inspector (in the viperOSK_UI gameobject inspector).
	• Minor bug fixes.

3.3	• Solved known issue [KI001] below, gamepad navigation in and out of UI

	• Solved issues with the UI input field based on Unity UI's TMP_InputField. The field can be edited correctly, InputField does not create visual artifacts anymore
	• Solved known issues with interacting with Unity UI's TMP_InputField or the TextMesh Input Field (through OSK_UI_InputField component). You can now select text, replace it, delete in the middle..etc.
	• Added support to multi-line for TMP_InputField and viperOSK custom UI InputField (prefab: **viperOSK_UI_InputField**)
	• Added **multi-line support** for 3D OSK_Keyboard & OSK_Receiver. To use multi-line support create a **special key** and ensure you attach a Unity event to the **OSK_Keyboard.NewLine()** method.
	 
	• Removed Invoke calls that halt if TimeScale=0 (what some developers use to pause a game). Invoke calls were replaced with Coroutines.
	• Known issue: caret (cursor) doesn't always show when using Unity's UI input field. There are two workaround:
	  ○ Recommended: use viperOSK custom UI input field, the prefab is **viperOSK_UI_InputField**
	  ○ Attach **UI_Cursor** prefab to your Unity UI InputField (TMP) as a child to the main object. Remember to hide/turn off the input field's built-in caret by checking the Custom Caret Color checkbox and putting the Caret Color alpha to zero

3.2	• Fixed compiler errors for developers who didn't have Rewired installed. Rewired is not necessary for viperOSK but it is supported.
3.1	• Added ToggleCharMask([bool]) to enable the revealing password fields and hiding them. The Char Mask must be set to a masking character to hide characters being typed. • Updated Example 4 to demonstrate this functionality • Fixed character masks when using OSK_UI_InputReceiver. Use OSK_UI_InputReceiver.Text() function to retrieve the text in the input field. The Input Field's text would have the masking characters (ex:"****") whereas the Text function returns the real string (ex:"pass123") • [KI001] Known issues: navigating with a gamepad to an OSK_UI_InputReceiver may create graphical artifacts in the OSK. We're looking to solve this issue in future releases. For now, we highly recommend using our custom OSK_UI_CustomReceiver class or the UI_InputField prefab (they have almost all of the functionality of Unity's UI Input Field).
3.0	• Addition of Example 5 scene that shows a full example of a UI system • Example 5 also includes example of configuring accented characters (such as è, á). • Addition of Examples 6.A to illustrate the use of multiple controllers that alternate in their use of the OSK

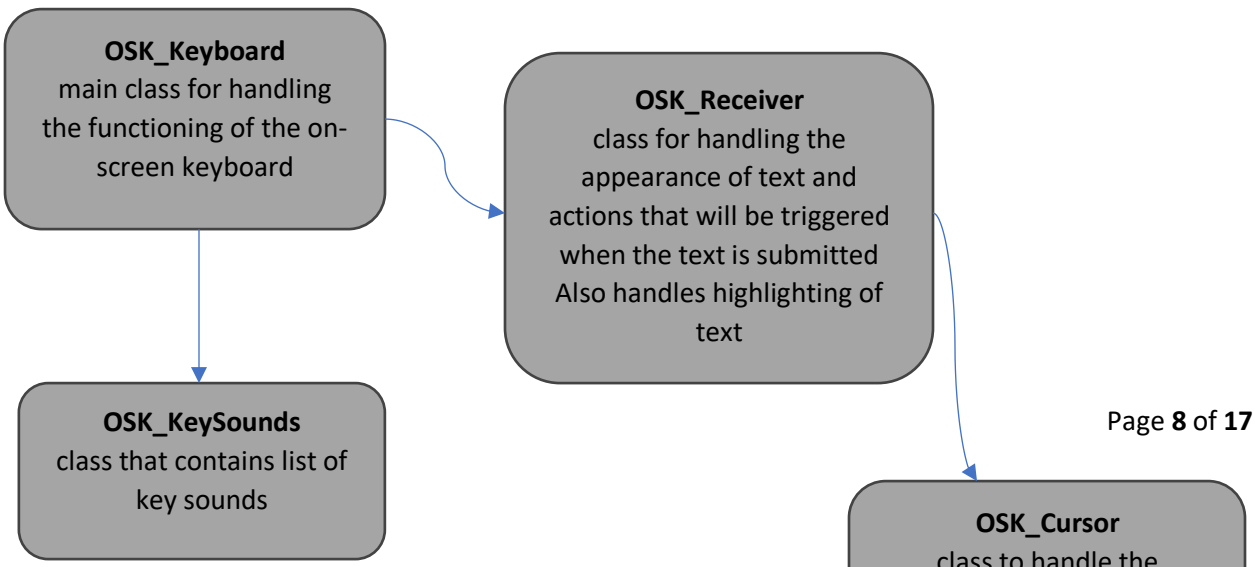
	• Addition of Example 6.B to illustrate the **simultaneous** use of 2 controllers as well as the use of animated objects to show which key is selected by a controller.
	• Addition of **`bypassDefaultInput`** [Boolean] to bypass viperOSK's input management. This helps developers who prefer using their own (ex: Rewired), or the UI implementation.
	• Addition of GamepadInput_Horizontal, GamepadInput_Vertical, GamepadInput_Submit, GamepadInput_Cancel to provide developers tailored access to gamepad input through custom event handlers.
	• Passive support for Rewired built into OSK_GamepadHelper component (requires Rewired asset and to define REWIRED in the project settings under scripting define symbols).
	• Addition of methods to set or get SelectedKey when using a gamepad controller. This allows developers to default the key selection to any key on the OSK.
	• Addition of keyboard wrapping (for controllers) when your selection marker goes off one end it wraps to the other (X, Y, XY and cascade wrapping available). Note: Y wrapping not available in UI mode.
	• Addition of sound when moving the selection marker using a controller, the choice of audio can be changed in OSK_KeySounds inspector.
	• Addition of support for various text alignment (Left, Center, Right)
	• Addition of ability to add listeners to events such as the when an input field:
	  ○ OnSelect: when an input field is highlighted
	  ○ OnSelectClick: when an input field is highlighted and the user presses the A button on their gamepad (or whatever the controller mapping is for A button). *Note: this event does not work with Unity's UI Input Field as it is hardwired for OS keyboard support. Instead use the OSK_Receiver or OSK_UI_CustomReceiver that are designed for viperOSK and provide almost all of the functionality that Unity's input field does*.
	  ○ OnValueChanged: when the text in the input field is changed (the string that is sent is the entire new text after changes)
	  ○ OnFocus and OnLostFocus: for when the receiver is "clicked on" or deselected.
	• Scripting Note: the callback when a key is pressed KeyCall parameters have changed, the 2nd parameter is no longer the key type but rather the OSK_Receiver for when the keyboard is outputting to more than one receiver (saves you from creating 2 keyboards)

2.1	Minor bug fixes and addition of `SetText_ShftEnabled(string)` which allows developers to program in special characters (for example: `é`) that are capitalized when shift/caps is selected.

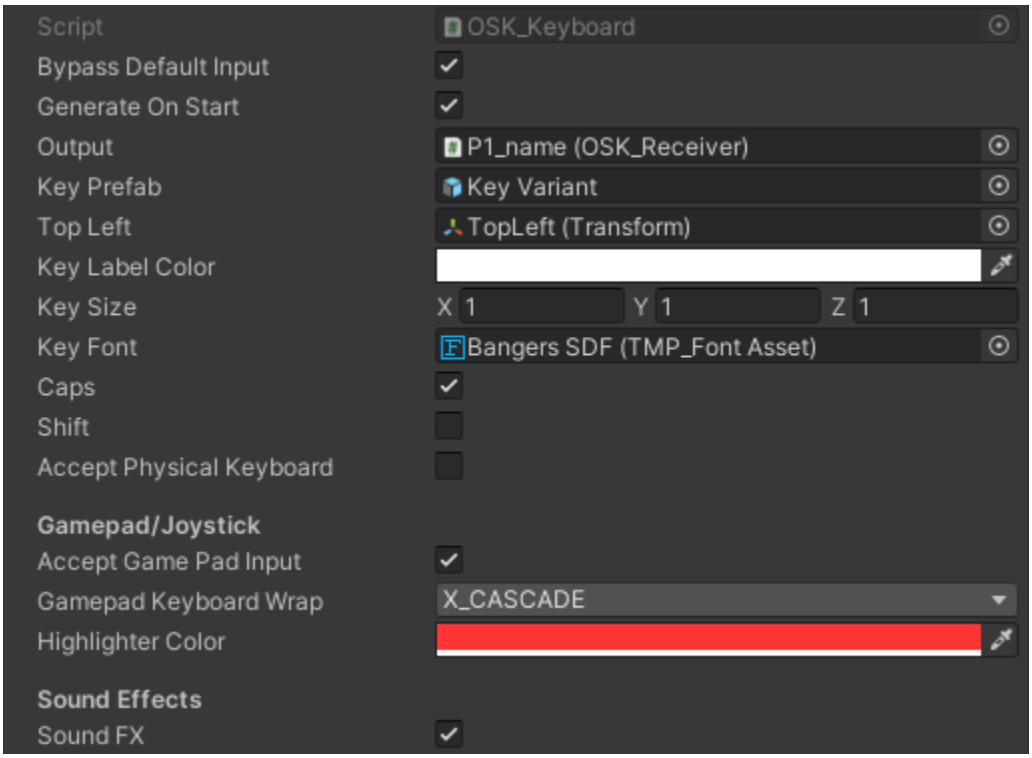
2.0	Considerable changes were made to **viperOSK**:
	1- There is now two implementations of the on-screen keyboard:
	  a. **OSK_Keyboard** is largely the same implementation as before and can work in most scenes (2D or 3D)
	  b. **OSK_UI_Keyboard** is a Unity UI implementation where the Keys are modified UI Buttons
	2- The 'text', cursor and 'OnSubmit' fields in OSK_Keyboard have been removed. They are now found in the new "receiver" object where keyboard strokes are sent. This allows developers to have multiple keyboard entry fields in the scene and more control over the functioning and look of the text.

	3- Added new component **OSK_Receiver** to receive OSK_Keyboard input. New component handles text display, text selection and cursor displacement, character limit, color and highlight color as well as the Submit events (allows text fields to have different submit behaviours, see Example 4 scene) 4- Added two inherited implementations of **OSK_Receiver**: **OSK_UI_CustomReceiver** and **OSK_UI_InputField** that can function in Unity UI (regardless of Canvas Render settings). See Example 3 for demo. 5- Added **OSK_Cursor** and **OSK_UI_Cursor** for a more robust cursor implementation, the look and color can be changed by modifying the sprite object. Both component have prefab implementations. 6- **TMP Essentials** is a required prerequisite of using **viperOSK** (you can bypass this depending on your level of programming/Unity knowledge). 7- Added ability to bypass programmed actions by allowing developers to set special actions for OSK_SpecialKeys (see Inspector instructions below for more). See Example 4, Tab key special action for illustration.
1.0	Original implementation. Detailed instructions are still available in this Guide Doc, see Annex A.

General Class Structure



Setup in Inspector part-1



1- **Bypass Default Input** bypasses the built in controller support in viperOSK, this is useful when using Unity UI version, or when using the OSK_GamepadHelper.
2- **Generare On Start** indicates whether the keys for the keyboard will be generated when the gameobject is created (through Start() script). If set to false, the keys must be generated in script through a call to Generate().
3- **Output** is the pointer to the **OSK_Receiver** game object where the text will be typed out by the on-screen keyboard. You can use the existing gameobject in the prefab or drag-n-drop your own under Output.
4- **Key Prefab** is a pointer to the Key prefab. The key prefab should be changed if you want to use your own key texture.
5- **Top Left** is the Transform where the first key (top left most key) will be displayed, all other keys will be drawn based on that (see more under Layout)
6- **Key Label Color** is the color of the key labels.
7- **Key Size** affects the scale of the keys and how they will look. For example, if x=1.5, then keys will be 1.5 times their width.
8- **Key Font** is the pointer to a TMP_FontAsset (see Setup for more info), leaving it empty will use the in-house TMP font, otherwise all keys will be displayed in the font of your choosing.
9- **Caps** whether caps lock is on or not
10- **Shift** shows whether the Shift or Caps Lock key is pressed or depressed
11- **Accept Physical Keyboard** allows users with a hardware keyboard (ex, PC keyboard) use their keyboard to type. Note that there are currently no limits to which characters will be allowed by the keyboard. This is a planned enhancements in future versions.

12- **Accept GamePad Input** allows viperOSK to accept gamepad/joystick controls to highlight keys and select them (where touch controls are not available such as in consoles).
13- **Gamepad Keyboard Wrap** will allow the selection marker to go to the other side (for example from the right most key to the adjacent leftmost key). X, XY, and Y wrapping are available, CASCADE wrapping will let the key go down one level when it's off to the right side and vice versa.
14- **Highlighter Color** color of highlighted key when using gamepad or other controls that are not touch or mouse. If using a SelectionMarker in OSK_GamepadHelper, you can set the alpha to 0 here.
15- **Sound FX** allows you to turn on or off whether key sounds will be used.

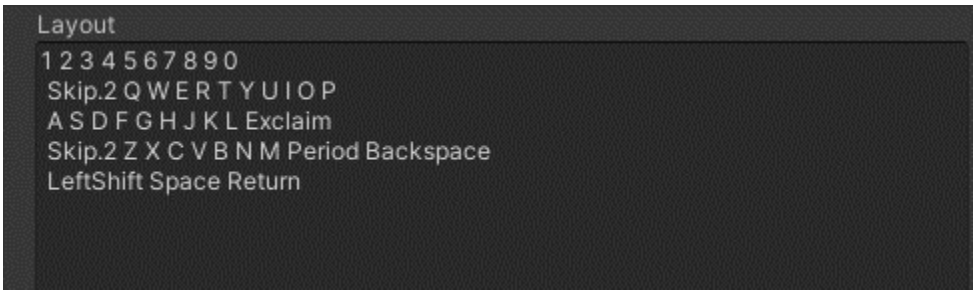
NOTES:

Changes from v1.0 to v2.0+:

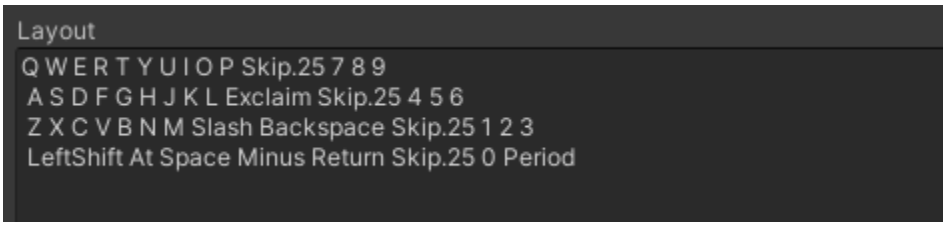
In v1.0 the cursor was a text appended to the end, this has been changed to a far more robust implementation through OSK_Cursor.

The OnSubmit callback is now handled in the OSK_Receiver. Developers can have multiple Unity Events triggered the user hits the Return key

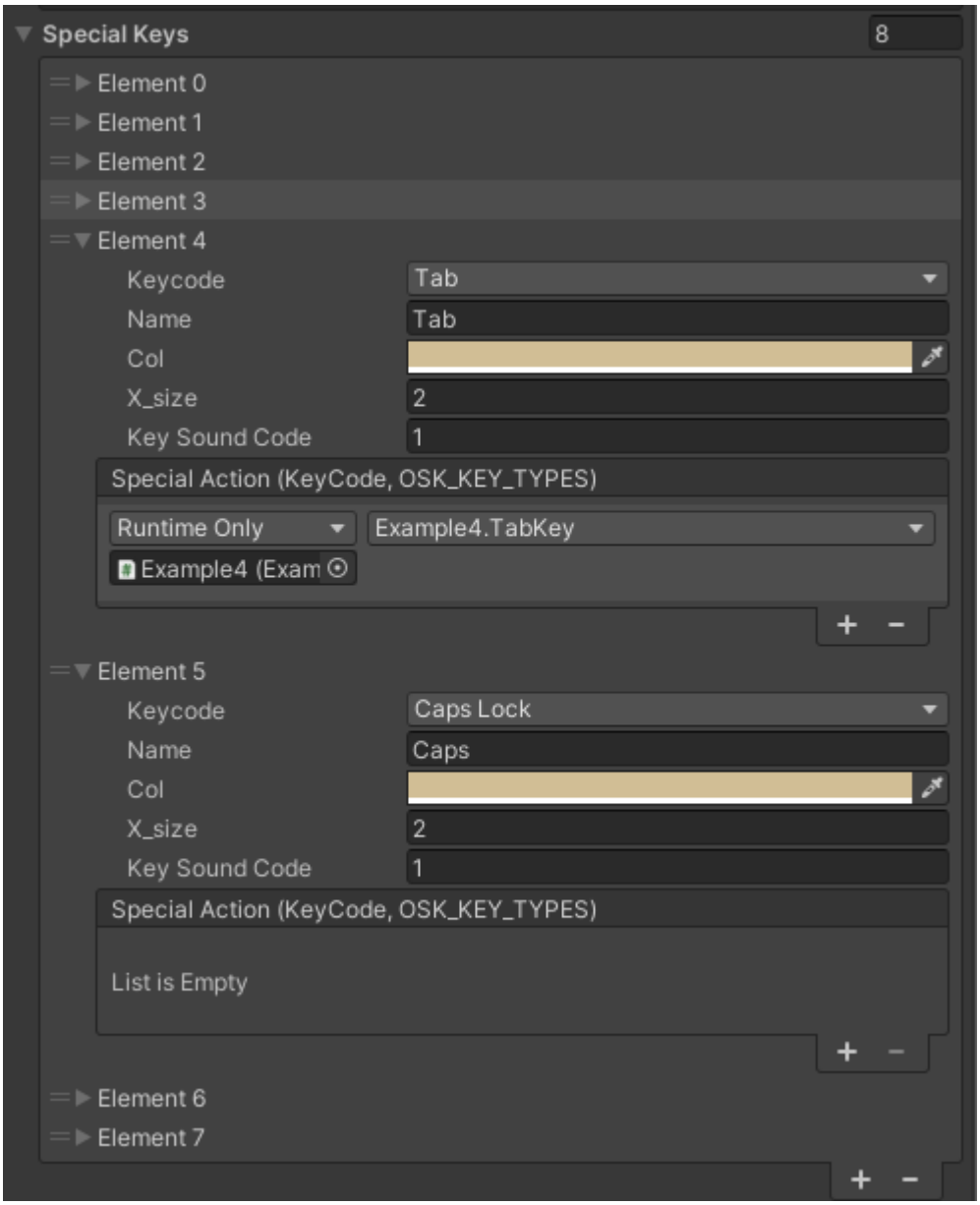
Setup in Inspector part-2: Layout



16- **Layout** is a simple string that guides the layout of the keys on the keyboard. There are several important notes to keep in mind as you setup the keyboard layout to your liking:
    a. You **must** separate all characters (including 'new line' or '\n') with a space (" "). The **OSK_Keyboard** script splits up the string using a single space as a separator. If you run into an error it is highly likely you forgot to separate with spaces correctly. Using the **Auto-Correct Layout** button in inspector would normally resolve most issues
    b. Punctuations and other keys must be spelled out using Unity **KeyCode** names. Note that all digits will be automatically converted to Alpha (ex, Alpha1, not KeyPad).
    c. The keyword **Skip** is a special keyword to create spacing in your keyboard. The format is **Skip#.#** for example Skip1.5 (no spaces!) will setup the next key at 1.5x the key width. In the example above, the keys are offset using **Skip.2** which offsets the keys by 0.2x key width.
    d. The Layout and using Skip is a powerful tool to setup the layout of the keyboard exactly how you want. Like the following example where the digits are in a keypad on the right:



Setup in Inspector part-3: Special Keys

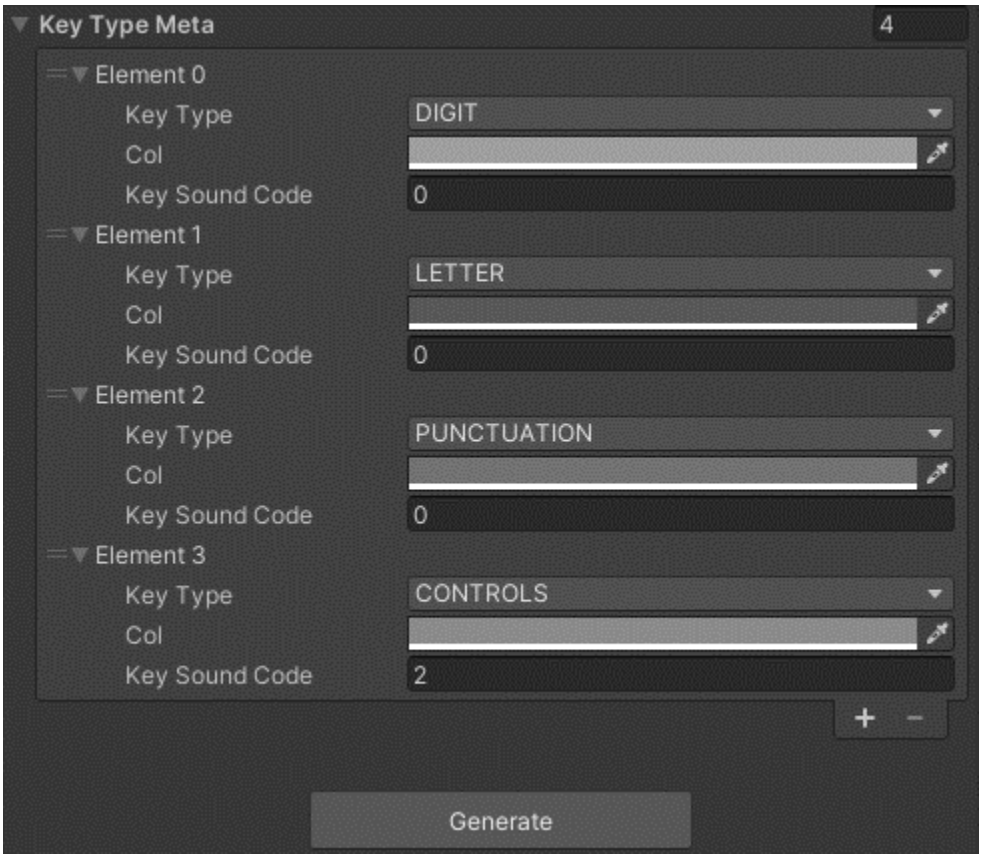


17- Special Keys is a list of keys you denote as being special. For example, the SPACE key is a special key as it is much wider than a regular key.
    a. viperOSK allows you to select any number of keys to be special. For each key you first select the corresponding ***KeyCode***, then set the **display name** (what will show on the on-screen key); the **Col** is the color of the key; **x_size** is the width (in float) of the key in multiples of Key Size (for example, Space is the width of 6 keys); and, **Key Sound Code** denotes the index of the sound to be used for this key (see Sounds section further below)
    b. [v2.0 change] the behaviour of special keys like Tab can now be programmed individually in the Inspector (see example above). Backspace, Shift, CapsLock, Return, Delete are still handled in the script but can be modified as well. **If a**

“specialAction” event is added then it would no longer go through the pre-programmed KeyCall callback.

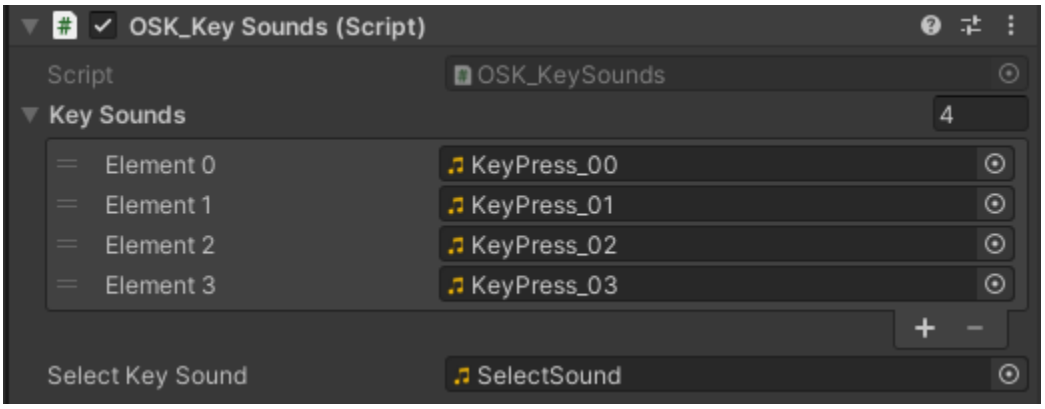
c. Any key can be designated as a Special Key, in example 2, you’ll see how we used this feature to color the WASD keys differently similarly to what you see in gaming keyboards.

Setup in Inspector part-4: Key Types and Generating keyboard



18- Key Type Meta allows further meta specifications by key type. You can specify key color and key sounds for each key type (digit, letter, punctuation or controls). Note the following:
    a. you should not have more than one specification per type
    b. the color and behaviour of Special Keys will trump the specifications in KeyTypeMeta
    c. punctuations include mathematical and other symbols
19- The **Generate** button will remove previous keys and generate the keyboard in Editor. The script can also be called during runtime for on-the-fly keyboard generation.

Setup in Inspector part-5: Key Sounds

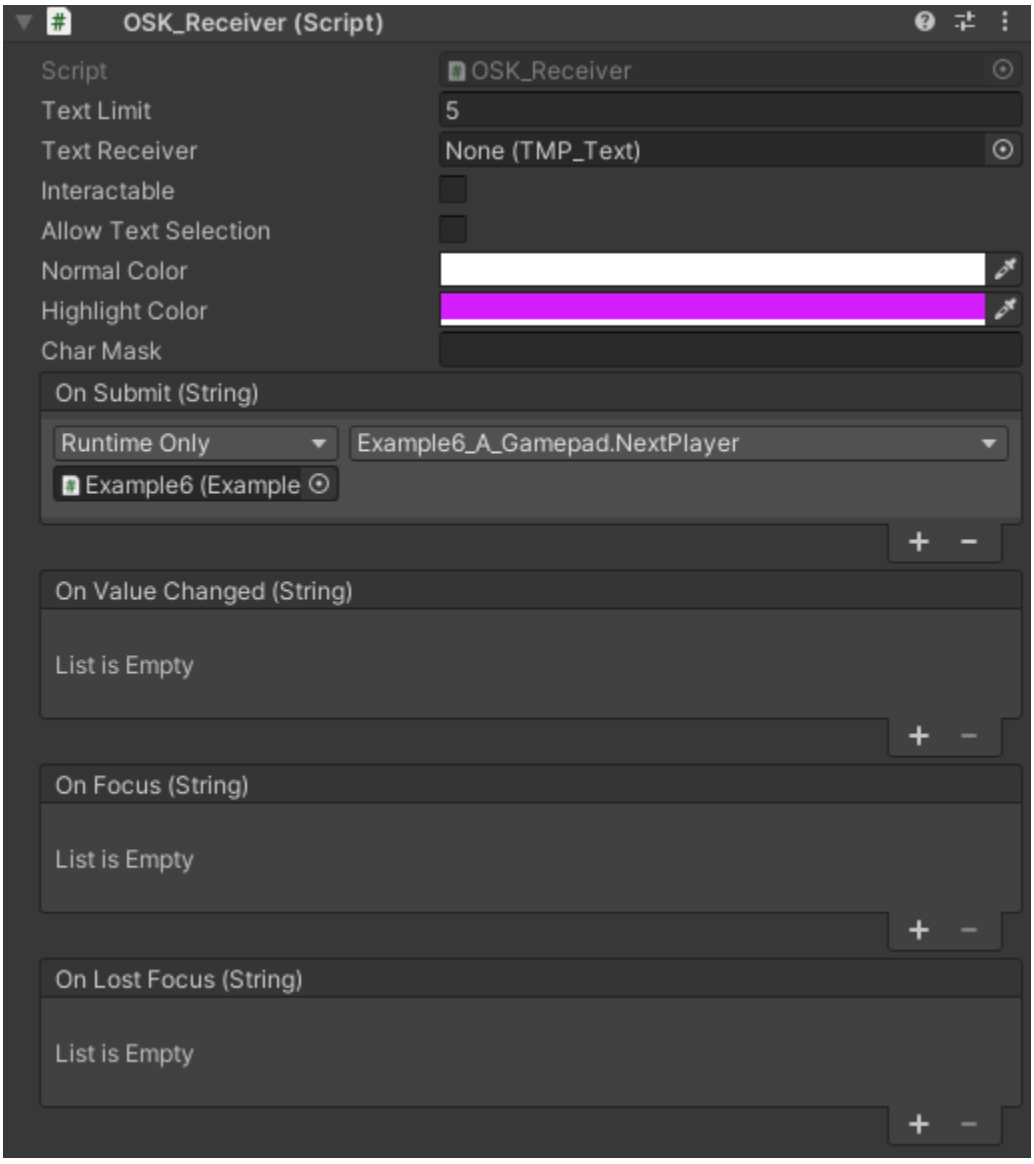


20- **Key Sounds** is a List of *Audioclips* that are reference from the **OSK_Keyboard** based on their index. The have been preloaded with open source sound but we recommend using your own if you have some handy. There are 4 pre-loaded sounds but you are not limited to only 4, you can load more or less.
21- **Select Key Sound** is the sound made when moving your key selection using a gamepad or controller

OSK_Receiver setup

The OSK_Receiver is component for the TMP object that will receiving the key strokes. There a few components that are important to the proper functioning.

1- The object must have a **Collider**, or, in the case of a Unity UI implementation it requires a **Selectable** so that events like moving the cursor or selecting the input field (in case of more than one) are triggered.
2- OSK_Receiver (or a component that inherits from it) should be in the same gameobject as the TextMeshPro or the InputField.



Inspector fields are the following:

1- **Text Limit**: limits the number of characters the text field can take.
2- **Text Receiver**: is the TMP component that receives the text, this is populated automatically if the TMP are on the same object, or can be set here.

3- **Interactable**: when checked the user can select the component by clicking/touch on it, and can move the cursor's position in the string (requires an OSK_Cursor child object).
4- **Allow Text Selection**: when checked the user can do text selection by dragging the pointer/touch.
5- **Normal and Highlight Color**: colors for normal text input and highlighted text (where allowTextSelection is true).
6- **Char Mask**: masks the input with the character/string entered in this box. For example "*" for a password field.
7- **On Submit**: Unity Events to send the submitted screen. Ensure you select the function under "dynamic string" to send the text.
8- **On Value Changed** sends to any added listeners the new string text of the receiver
9- **On Focus** is called when the receiver is selected or the OnFocus() function is called in script.
10- **On Lost Focus** is called when the receiver is de-selected or the OnLostFocus() function is called in script.

This Guide Doc will be updated with new features as they are added to **viperOSK**. I hope it makes your game dev easier and look forward to seeing your own creations with it.

For support, please email us at: support@vipercode.games