```
/// <summary>
/// This will re-calculate the highlight renderer scale based on the size of the Ring Highlight Positio                lightPositio
/// Note this does call PositionChildObjects once complete.
/// </summary>
public void RefreshHighlightRendererScale()
{
    ringHighlightPosition.localPosition = new Vector3(0, ringHighlightPosition.localPosition.y, 0);

    if (scaleMode == MinAndMax.Min)
        hightlightRendererSize = Mathf.Min(ringHighlightPosition.rect.width, ringHighlightPosition.rect.height);
    else
        hightlightRendererSize = Mathf.Max(ringHighlightPosition.rect.width, ringHighlightPosition.rect.height);

    highlightOffset = ringCenterPivot.localPosition.y - ringHighlightPosition.localPosition.y;

    PositionChildObjects();
}

/// <summary>
/// Calculates the ring and element size
/// This will fully rebuild the element list based on the child elements of the Ring Center Pivot (ringCenterPivot) RectTransfo
/// </summary>
public void PositionChildObjects()
{
    elementList.Clear();
    foreach(Transform childElement in ringCenterPivot)
    {
        if (!childElement.gameObject.activeInHierarchy)
            continue;

        indexChildObject(childElement);
    }

    if (elementList.Count > 0)
    {
        //Get the required circumfrence
        float circ = ((hightlightRendererSize * relativeElementSize) * (relativeGapSize + 1)) * elementList.Count;

        //Get the resulting raidius
        ceterPointOffset.z = ((circ / Mathf.PI) / 2f);
        ringCenterPivot.localPosition = new Vector3(ringCenterPivot.localPosition.x, ringCenterPivot.localPosition.y, ceterPoin
        //plot a point on the ciricle offset by the index of each element
        RadianGap = (Mathf.Deg2Rad * 360f) / (elementList.Count); //this will add 1 slot for our gap

        rotateToIndex(selectedIndex, true);

        //We start at 1 since 0 is the selected index
        for (int i = 0; i < elementList.Count; i++)
        {
            elementList[i].targetReference.transform.localPosition = new Vector3((ceterPointOffset.z * Mathf.Sin(RadianGap * i)
```

/// <summary>
/// This will re-calculate the highlight renderer scale based on the size of the Ring Highlight Position (ringHighlightPositi
this does call PositionChildObjects once complete.

Contents

Overview 2
Demo Scenes 2
Manifest 3
Common 3
RoutedEvent<T> 3
Helpers 3
KeyboardKey 3
Ligature Helper 3
Controls 4
Keyboard 4

Overview

Heathen Engineering's UIX is a collection of custom controls, helper behaviours and example scenes meant to accelerate development of UI elements and behaviours. The collection is built with the mind to grow and to this effort Heathen Engineering is taking requests for new and expanded scripts, controls and behaviours on the Heathen Support Center.

E-mail: Support@HeathenEngineering.com

All code is heavily commented and includes XML based comments and Unity Inspector Tool Tips where appropriate. Prefabs are available along with an editor script which will add GameObject and Component menu entries. Note the GameObject menu entries are sensitive to the file location of the prefabs and will break if the folder structure is changed.

Demo Scenes

Demo Scenes are available along with prefabs, menu entries and component entries for all elements of the collection. The demo scenes demonstrate as follows

1. Filtered Input Demo
    a. Filtered Input Field
    b. Password Mask
    c. Combo Box Button & Combo Box Item
2. Keep Size on Screen
    a. Keep Size on Screen
    b. Clamp to Screen
    c. Rotate 2D
3. Keyboard Demo
    a. Keyboard
    b. Keyboard Template
    c. Ligature Helper
4. Render Queue and 3D Masking
    a. Scroll Rect Helper
    b. Render Queue
5. Ring Demo
    a. Snap to Transform
    b. Ring Collection
    c. Rotate 3D
6. Scroll Helper Demo
    a. Scroll Rect Helper
    b. 3D Mask
7. UI Manager Demo
    a. UI Manager
    b. Pointer Manager
8. [UIX] Simple tooltip example
    a. Tooltip Manager
    b. Tooltip Message
    c. Tooltip

Manifest

Common

The common scripts are available from several Heathen assets and are located in a ‘Common’ folder which should be shared among all assets.

RoutedEvent<T>

The routed event delegate simplifies event driven design key to a flexible and powerful UI design approach. Unlike Unity’s stock Event Manager; delegates afford much greater flexibility from a developer’s point of view where in custom events can be defined and registered in code with the standard syntax of.

```
public event RoutedEvent<MyDataTypeOfChoice> MyCustomEvent;
```

Registering to such an event is just as simple via

```
MyCustomEvent += MyCustomEventHandlerFunction;
```

Where MyCustomEventHandlerFunction is a function taking an ‘Object sender’ and ‘MyDataTypeOfChoice data’ paramiters.

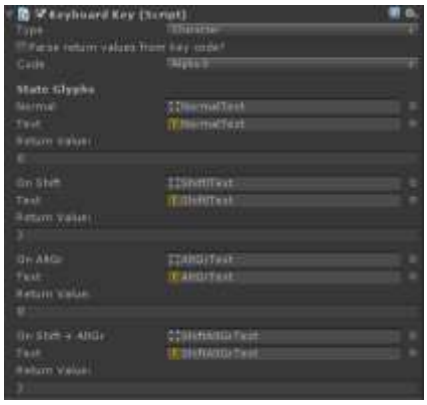
The use of events, event delegates and related topics is well defined on MSDN and other standard .NET code resources. Most compilers with a form of intellisense provide shortcuts and autocomplete options for events and delegate registrations.

Heathen’s UIX and other Heathen Engineering assets leverage such events extensively however where applicable more Unity centric approaches are also used e.g. Boolean tests, EventManager, etc. as a matter of conveyance for developers.

Helpers

KeyboardKey

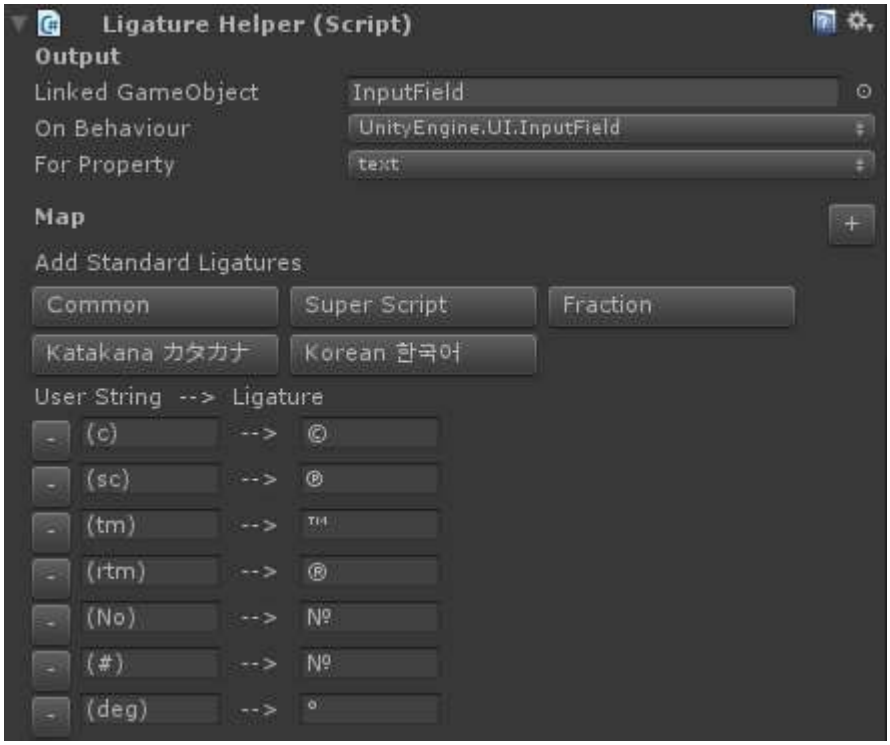
Keyboard key is a sub component of the Keyboard object and replaces Heathen’s On Screen Keyboard Key. This behaviour carries the required information and events to define a functional virtual keyboard key. See the keyboard control notes for details.



Ligature Helper

This helper can be placed on any gameobject and multiple may exist in a scene. Its purpose is to consume a string and convert character combinations into new symbols for example (c) becomes © or ka becomes 力.

Ligature helper is required in order to support some languages with the UIX Keyboard such as Korean or Japanese as these languages depend on ligatures for typing. Like the keyboard ligature helper can reference any text attribute of any component as its output and it has ReplaceAll and ReplaceEnd methods suitable for OnValueChanged unity events. Note ReplaceEnd will only test the end of the string regardless of the active insert point while replace all will always test the full string for viable replacements. Utility functions are available making it easy to develop your own replacement rules.



Controls

Keyboard

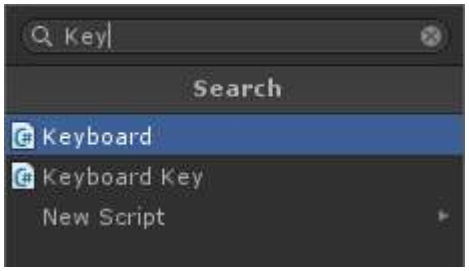
The keyboard is a complex generic control which allows the user to simulate a keyboard e.g. a virtual or 'On Screen' keyboard. UIX Keyboard supports templates and UIX includes ready to use templates for basic QWERTY basic AZERTY and Korean QWERTY mapping. Prefabs of each are available in the package or the templates can be used to generate keyboards from new empty GameObjects. Templates may be subsequently saved as to produce a new template file on disk which will include key placement, settings and behaviour information.

Create a Key Template

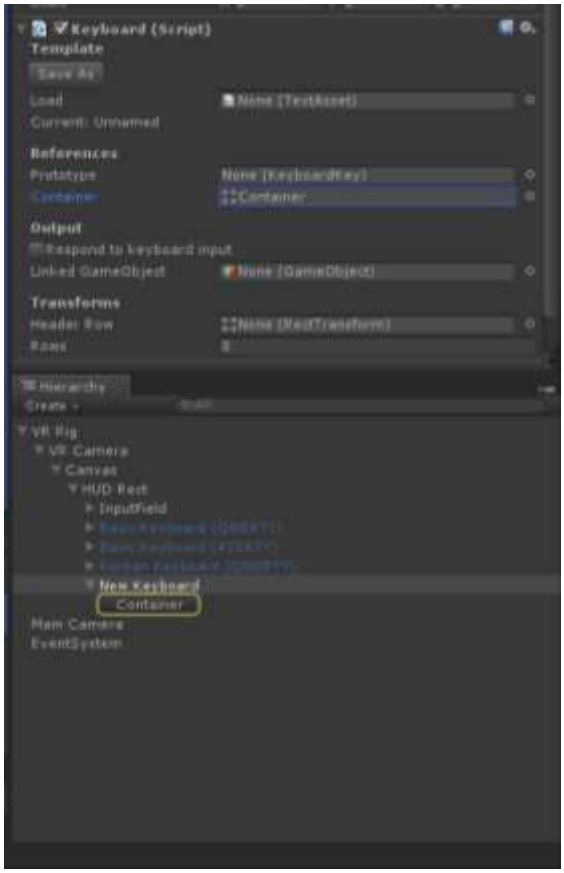
A key template is simply a generic key, this is a GameObject which has the Keyboard Key component attached. You may add any additional elements you like including 2D and 3D objects to give it the visual style you want. An example is available in the prototype folder named 'Basic Key Template'

Create Keyboard from Template

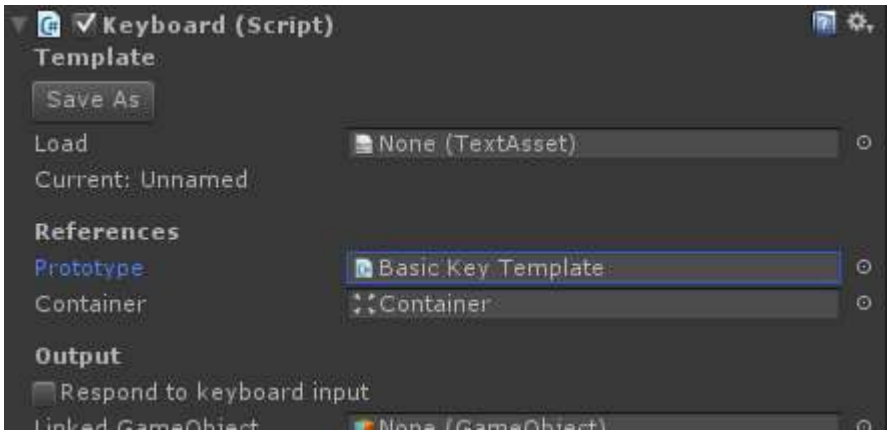
1) Create an empty GameObject and add the Keyboard component to it from Heathen's UIX



2) Add a new empty below your new keyboard and add it to the Container transform reference



3) Add a reference to your key template; see Create a Key Template for more details



4) Select the template you want and drop it on the Load field, the key board will automatically build once you add the template.

5) Adjust any keys as desired and add an output reference if desired