Input Prompt Display

Version 1.0.0

Input Prompt Display
Version 1.0.0

Contents

1.0 Overview

1.1 Introduction

Input Prompt Display is a Unity Plug-in designed to easily display key bindings in the UI using Unity's input system.

This project uses example sprite textures taken from Free Prompts Pack created by and made available to public domain by Nicolae Berbece as well as sprite textures taken from Free Controller Prompts created by and made available to public domain by Jeremy Statz. Textures are licensed under CC0.

1.2 Features

Input Prompt Display includes several features to allow for flexibility.

- Displays key bindings in the UI
- Integrated to use the input system
- Supports dynamic input device switching
- Supports different prompt styles, including platform specific styles
- Supports displaying a prompt for unassigned or unknown key bindings
- Supports displaying key bindings for multiple players
- Supports displaying alternative key bindings
- Supports dynamic display based on control scheme
- Supports both direct asset reference loading and addressable reference loading

1.3 How It Works

The Input Prompt Display system works in tandem with Unity's input system. Follow this documentation for more details on the input system. The input system abstracts the key binding process into several layers. At the top is the input action asset, which defines the available control schemes and action maps. A control scheme defines a set of input device requirements that will give access to bindings. For example, an Xbox control scheme could be defined for Xbox controllers and a Playstation control scheme could be defined for Playstation controllers. An action map defines a set of actions, which represent different player commands and interactions such as move, jump, crouch, etc. An action contains a set of key bindings with each binding being associated with a control scheme.

Input Action Asset							
Action Map				Action Map			
Action		Action		Action		Action	
Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme	Binding + Control Scheme

The Input Prompt Display system has several layers of abstraction as well. At the top is the master mapping, which defines the list of control scheme mappings. A control scheme mapping pairs a control scheme with a set of input prompts. An input prompt pairs an input from a device from the control scheme with a set of prompts. A prompt is a pairing of a sprite texture and an associated visual style.

Master Mappings							
Control Scheme Mapping				Control Scheme Mapping			
Input Prompt		Input Prompt		Input Prompt		Input Prompt	
Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style	Sprite Texture + Style

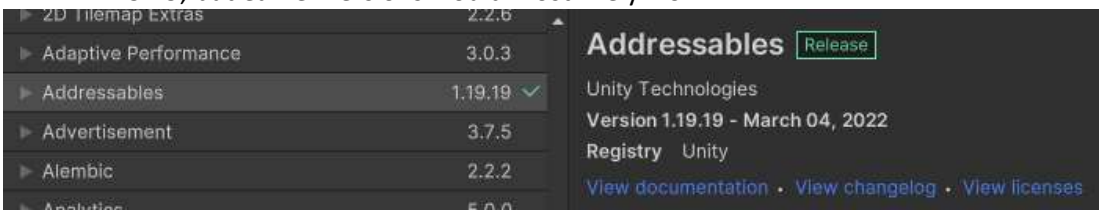
Within the input system, the developer will set the current action map, and the player will set the current control scheme based on the input device being used. The Input Prompt Display system will look up a given action from the current action map to find a binding for the current control scheme. The system will then look up the matching control scheme mapping from the current control scheme. From there, it will look up the input prompt based on the binding returned by the input system for the given action. Lastly, it will look up a sprite texture that matches a given style.

1.4 Setup

Input Prompt Display relies on a number of different features in order to function. The primary dependencies are Unity's input system and addressables. This requires quite a bit of setup in order for the system to work. You may already have some setup requirements in your own project, but this documentation will assume you are starting from scratch.

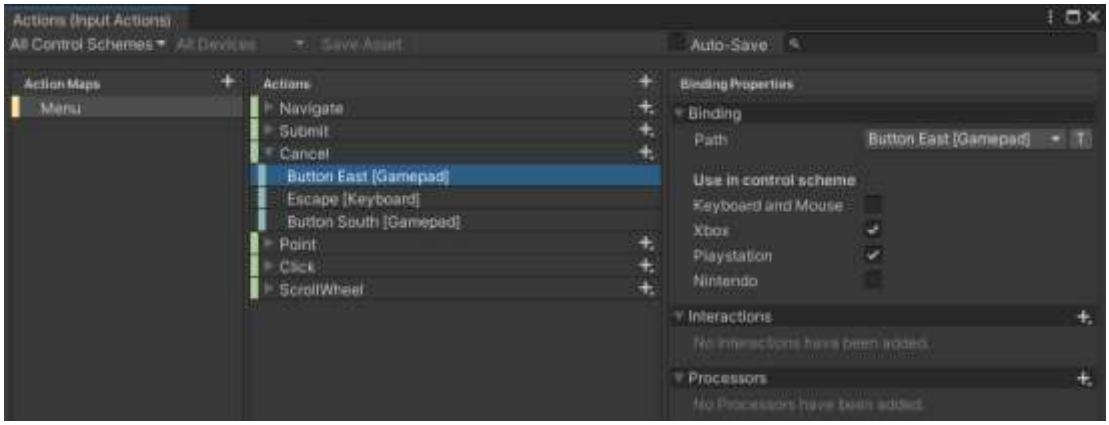
1. Install and setup the Addressables package from the package manager. Follow this documentation for more details. Input Prompt Display was built using version

1.19.19, but earlier versions would most likely work.

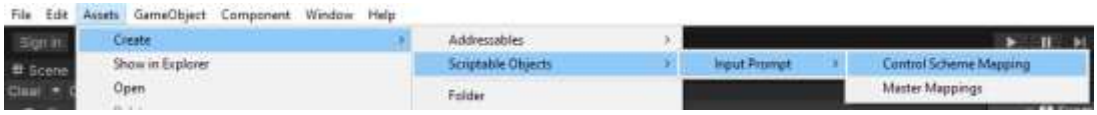


2. Install and setup the Input System package from the package manager. Follow this documentation for more details. Input Prompt Display was built using version 1.3.0.

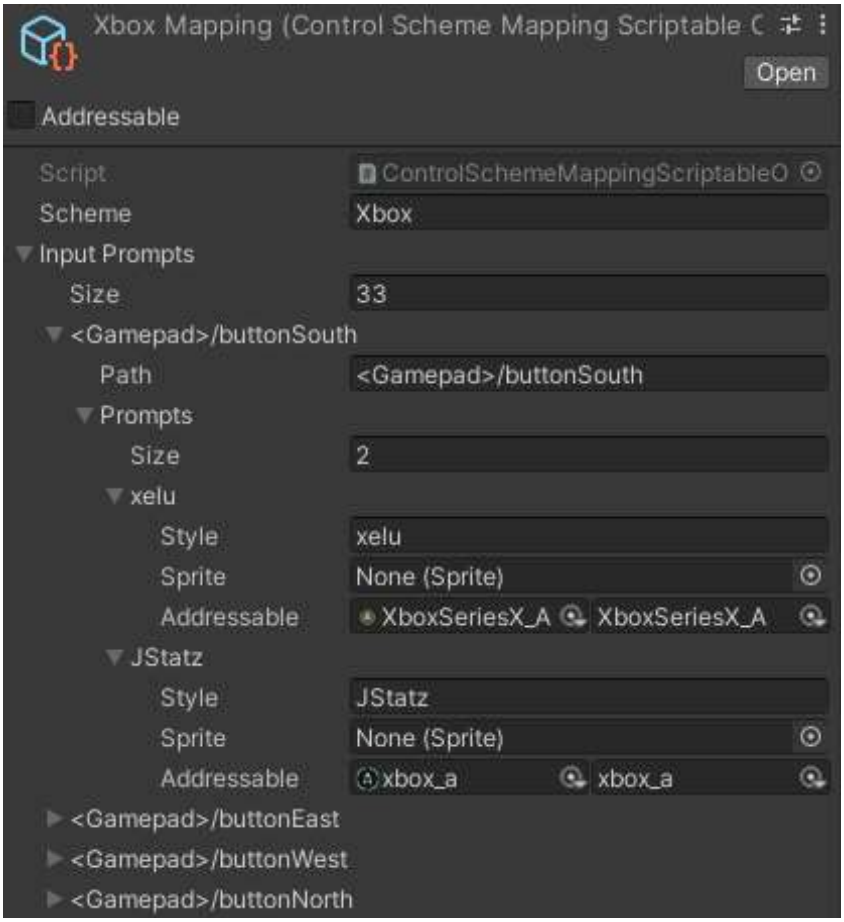
3. Create an Input Action Asset complete with the appropriate action maps, actions, control schemes, and bindings. Follow this documentation for more details. Do note the names used for the control schemes.



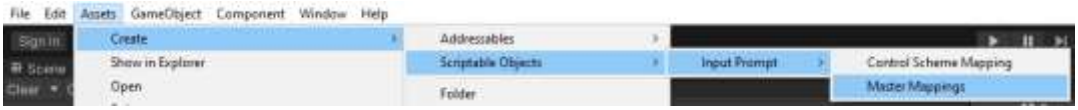
4. Create a Control Scheme Mapping Scriptable Object for each supported control scheme. To create one, go to **Assets > Create > Scriptable Objects > Input Prompt > Control Scheme Mapping** from Unity's main menu.



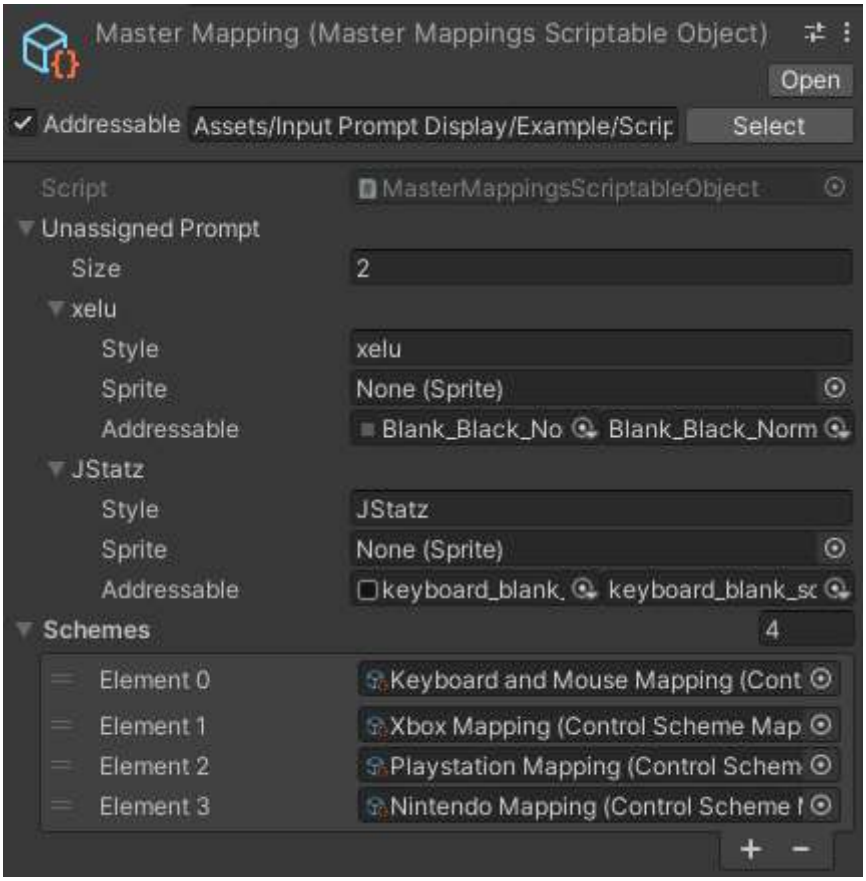
5. For each Control Scheme Mapping Scriptable Object, provide the name of the control scheme as well as the input prompts for each input. For more information on input prompts, see section **3.0 Data Mapping**.



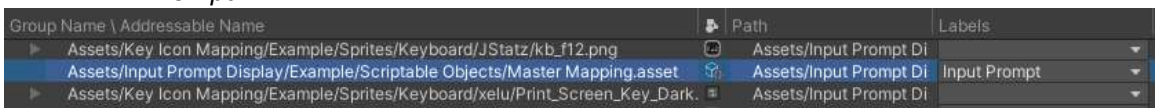
6. Create a Master Mappings Scriptable Object. To create one, go to **Assets > Create > Scriptable Objects > Input Prompt > Master Mappings** from Unity's main menu.



7. For the Master Mappings Scriptable Object, set the prompts for the default unassigned prompt as well as the Control Scheme Mapping Scriptable Objects for each supported control scheme. For more information on input prompts and the unassigned prompt, see sections **3.0 Data Mapping** and **4.3 Unassigned Prompt**.



8. Set the Master Mappings Scriptable Object as addressable and set its label as *Input Prompt*.



9. Optional: It is recommended to load the input prompt data during startup or whenever loading would be most optimal to avoid any visual hitches. If the input prompt data is not loaded manually, it will be automatically loaded on the Input Prompt Display system's first use. To do so, follow these two steps:

    a. Include the IPD namespace.

```
using IPD;
```

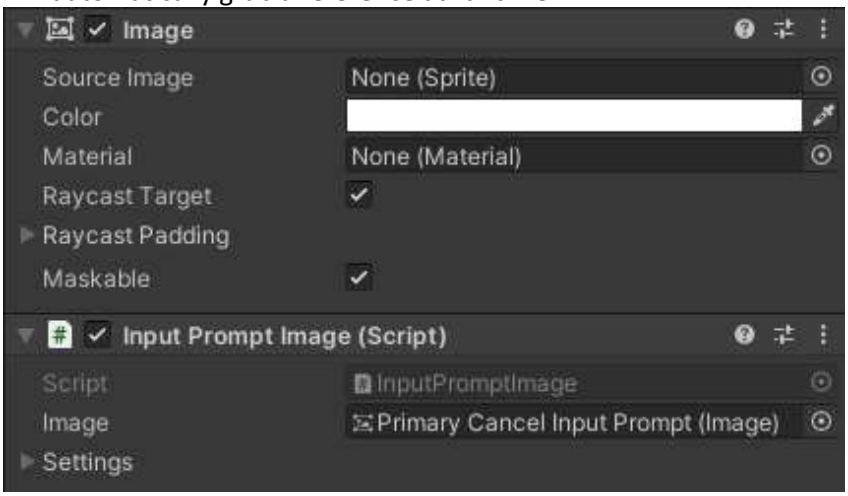
    b. Asynchronously load the input prompt data.

```
bool isLoaded = await InputPromptUtility.Load ( );
```

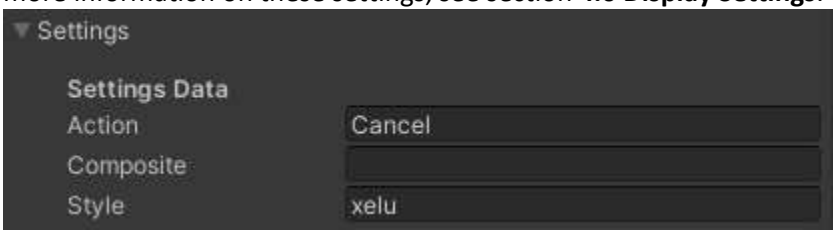
1.5 Basic Usage

Once the Input Prompt Display system has been fully set up in your project, displaying a key binding in the UI can be done in two simple steps:

1. Add an Input Prompt Image component to a game object with an Image component and set a reference to the Image component. If a reference to the Image component is not set in the inspector manually, the Input Prompt Image component will automatically grab a reference at runtime.



2. Provide the desired action, composite part (if applicable), and style to display. For more information on these settings, see section **4.0 Display Settings**.



2.0 Classes

2.1 IPD.DisplaySettingsModel

Description: This class stores the settings for displaying an input prompt.

Variables		
Name	**Type**	**Description**

Action	string	The action being displayed.
Composite	string	The name of the composite part binding for the action. Leave empty for non-composite bindings for an action.
Style	string	The style for the prompt.
IsUnassignedDisplayed	bool	Whether or not the unassigned prompt should be displayed if a binding is not found. Set to true as default. Set to false to hide the element instead.
PlayerIndex	int	The index of the player bindings to display. Set to 0 as default. Increase to display bindings for an action of an additional player.
AltBindingIndex	int	The index of the binding to display. Set to 0 as default. Increase to display alternative bindings for an action.
HideForControlSchemes	string [ ]	The list of control schemes that this input prompt should be hidden for. This is useful for hiding input prompts for mouse or touch inputs.

2.2 IPD.InputPromptImage

Description: This class controls displaying an input prompt.

Function		
Name	**Return Type**	**Description**
Initialize	void	Sets and displays the input prompt.
Parameters		
Name	**Type**	**Description**
newSettings	DisplaySettingsModel	The settings data for displaying the input prompt.

2.3 IPD.PromptModel

Description: This class stores the data of a single prompt for an input.

Variables		
Name	**Type**	**Description**
Style	string	The visual style for the prompt. Can include platform specific styling (e.g. WINDOWS/filled/small).
Sprite	Sprite	The directly loaded sprite for the prompt. Use this when not using the addressable.
Addressable	AssetReference	The indirectly loaded addressable for the prompt. Use this when not using the directly loaded sprite.

Function		
Name	**Return Type**	**Description**
IsValid	bool	Gets whether or not this model contains valid data.

Function		
Name	**Return Type**	**Description**
LoadSprite	Sprite	Gets the sprite value for this prompt.

Function		
Name	**Return Type**	**Description**
LoadSpriteAsync	Task<Sprite>	Gets the sprite value for this prompt asynchronously.

2.4 IPD.InputModel

Description: This class stores the data of the prompts for a single input of a device.

Variables		
Name	**Type**	**Description**
Path	string	The control path of the input on the device.
Prompts	PromptModel [ ]	The prompts for the input.

2.5 IPD.ControlSchemeMappingScriptableObject

Description: This scriptable object stores the data for mapping the input prompts of a control scheme.

Properties		
Name	**Type**	**Description**
Scheme	string	The name of the control scheme.
InputPrompts	InputModel [ ]	The data for the input prompts of the control scheme.

2.6 IPD.MasterMappingsScriptableObject

Description: This scriptable object stores the master data for the input prompt mappings for each supported control scheme.

Properties		
Name	**Type**	**Description**
UnassignedPrompt	PromptModel [ ]	The prompt to display for unassigned or unknown key bindings.
Schemes	ControlSchemeMappingScriptableObject [ ]	The input prompt mapping data for each control scheme.

2.7 IPD.InputPromptUtility

Description: This class controls loading and storing the scriptable objects for the input prompt system.

Enums		
Name	**Description**	**Values**
LoadingState	The listed states for loading the input data in the utility.	UNLOADED
		LOADING
		LOADED

Properties		
Name	**Type**	**Description**
State	LoadingState	The state of loading the addressables for the utility.

Function		
Name	**Return Type**	**Description**
Load	Task<bool>	Asynchronously loads the master mappings scriptable object from addressables.

Function		
Name	**Return Type**	**Description**
AddOnLoadCompleteCallback	void	Adds a callback to the On Load Complete event. The event clears all callbacks after triggering.
Parameters		
Name	**Type**	**Description**
onComplete	System.Action	The callback to be added.

Function		
Name	**Return Type**	**Description**
RemoveOnLoadCompleteCallback	void	Removes a callback from the On Load Complete event. The event clears all callbacks after triggering.
Parameters		
Name	**Type**	**Description**
onComplete	System.Action	The callback to be removed.

Function		
Name	**Return Type**	**Description**
GetPrompt	PromptModel	Returns the prompt model for a given control scheme, control, and style.
Parameters		
Name	**Type**	**Description**
scheme	string	The current control scheme.
control	InputControl	The input control for the action.
style	string	The desired style for the prompt.

Function

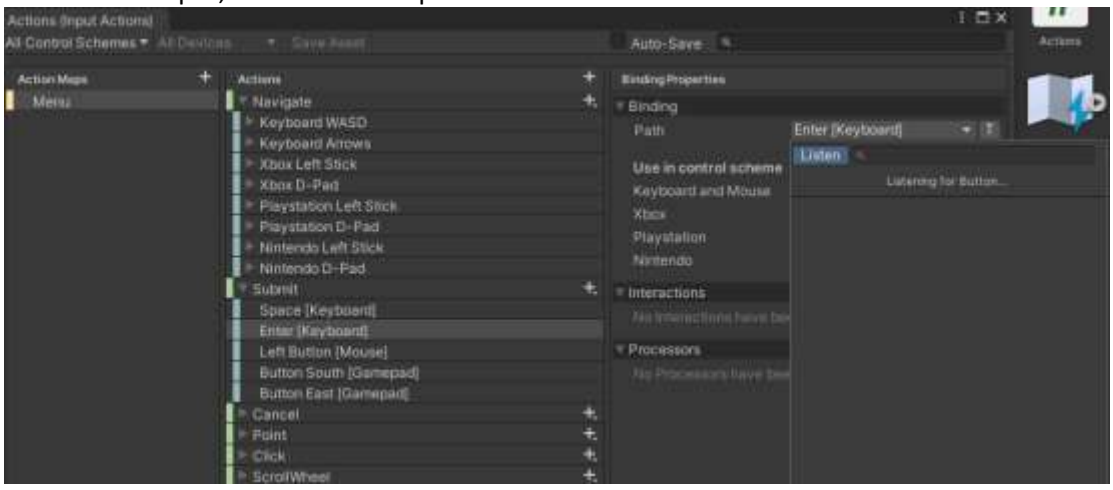
Name	Return Type	Description
GetUnassignedPrompt	PromptModel	Gets the data for the unassigned prompt.
Parameters		
Name	**Type**	**Description**
style	string	The desired style for the unassigned prompt.

Function		
Name	**Return Type**	**Description**
GetControlSchemeMappings	ControlSchemeMappingScriptableObject	Gets the data for a control scheme by its name.
Parameters		
Name	**Type**	**Description**
scheme	string	The name of the control scheme.

3.0 Data Mapping

3.1 Control Paths

When mapping prompts to an input, you will need to provide a control path to identify the specific input. Unity's input system uses control paths similar to regular expressions. Follow this documentation for more details. While Unity does not provide a list of control paths in their documentation, the Example scene provided in this project includes a debug controls setting that, when active, will log the control path of every input for the current device in use. You can use this setting as well as the Listen feature when binding inputs to actions. To access this feature, go to **Action Asset > Action Map > Action > Binding > Path > Listen** from Unity's action asset menu. It should be noted that the Listen feature will return the display name of the input, not the control path.



Controls path can be written to be generic or specific. For example, the control path for the B button on a standard Xbox Series X controller is */XInputControllerWindows/buttonEast*. However, it can be also written as *<Gamepad>/buttonEast* to cover any gamepad controller for that control scheme. It is generally recommended to use the generic control path to cover any potential edge cases, but the specific control path is useful if you'd like to provide unique prompts for a certain device. When providing both specific and generic control paths, set the specific control paths earlier in the array of input models since the system will return the first match it finds.

3.2 Styles

Styles allow you to map multiple prompts to a single input. This is useful for displaying key bindings with different visuals such as having both filled and unfilled icons or having sprites of varying resolutions. When mapping a prompt to an input, the prompt is given a string identifier for its style. It is recommended to have the same set of styles for every mapping across all control schemes. The reason for this is when attempting to retrieve an input prompt, the system will find the prompt matching a given action and style. If custom key bindings are allowed, then potentially any input could be assigned to an action, which necessitates consistency across all inputs within a control scheme. If the player switches control scheme by changing input devices, then the same style will need to be present for the input in the new control scheme, which necessitates consistency across all control schemes. If a matching style is not found, then the first prompt in the list for that input will be returned.

3.3 Platform Styles

Because Unity's input system runs off of control schemes, it is input device specific, not platform specific. This is useful because different input devices can be used on the same platform. However, this can be limiting if you need to display input prompts for a control scheme differently when on different platform. For example, the keyboard control scheme does not distinguish between Windows keyboards and Mac keyboards, which becomes a problem when trying to display the Windows key or the Command key respectively. This is where platform styles come in. When mapping a prompt to an input, the style can be prefixed with a platform keyword. This keyword will ensure that the prompt will only be displayed on that specific platform. To add a platform keyword, the style should be formatted like so: *<PLATFORM>/<Style>*

It should be noted that a platform keyword should be added to an existing style. While the platform style will take priority over the given style when on that specific platform, there should still be a version of that input prompt with the existing style to use when on other platforms. For example, the Windows key could be given the platform style *WINDOWS/minimal* for Windows, and the Command key could be given the platform style

MAC/minimal for Mac OS. However, there should be a prompt with the style *minimal* for the system key since other platforms can support keyboard inputs.

Here is the list of supported platform styles and their keywords:

Platform	Keyword
Windows	WINDOWS
Mac OS	MAC
Xbox One	XBOX ONE
Xbox Series S|X	XBOX SERIES
Playstation 4	PS4
Playstation 5	PS5
Nintendo Switch	SWITCH

3.4 Prompts

A prompt is the sprite texture that will be displayed for a key binding. When data mapping prompts, the sprite texture can be provided in one of two ways. The first method is to provide a direct reference to the asset. While this is the simplest approach, this has the result of the sprite being loaded into memory once the input prompt data is loaded, which can be an issue at scale. The second method is to provide an addressable reference. This approach has the benefit of loading assets into memory on demand and is recommended when working at scale. It should be noted that only one of the two methods should be used at a time for a prompt. If a prompt provides both a direct reference to the sprite and an addressable reference, the system will default to using the direct reference to save on performance.

4.0 Display Settings

4.1 Actions

When providing the settings data for an Input Prompt Image, you will need to provide the name of the action to be displayed. The system will look up the current key binding(s) for the action as well as the corresponding input prompt. The action provided needs to be present in the current action map in order to work. It should be noted that action names are not case sensitive.

4.2 Composite Bindings

When multiple key bindings need to work in unison for an action, the input system uses composite bindings. For example, movement is often programmed to use a 2D vector for

input from a gamepad analog stick or from the W, A, S, and D keys from a keyboard, so the movement action would require a composite binding. Follow this documentation for more details.

The Input Prompt Image can only display a single key binding at a time, so in order to display an action with a composite binding, the Input Prompt Image will only show the prompt for a single composite part at a time. When displaying an action with a composite binding, the action and the composite part will need to be provided in the settings data for the Input Prompt Image. For example, the composite part *up* would retrieve the W key on keyboard for the 2D axis movement action. If the action does not have a composite binding, the composite part should be left empty in the display settings data.

Here is the list of supported composite binding types and their composite parts:

Composite Type	Composite Part
1D Axis	positive
	negative
2D Axis	up
	down
	left
	right
3D Axis	up
	down
	left
	right
	forward
	backward
1 Modifier	modifier
	binding
2 Modifiers	modifier1
	modifier2
	binding

4.3 Unassigned Prompt

If an action is missing a key binding or has a key binding unknown to the input prompt data, a default unassigned prompt can be displayed instead. The unassigned prompt is set in the Master Mappings Scriptable Object and can have multiple styles (and platform specific styles) like any other input prompt.

The unassigned display setting is an optional display setting that controls displaying the unassigned prompt when encountering missing or unknown key bindings. If this setting is disabled, the Input Prompt Image will be hidden instead. It should be noted that even with the setting enabled, the Input Prompt Image may still be hidden due to certain errors or edge cases.

4.4 Multiple Players

Since the input system supports having multiple players, the Input Prompt Display system supports multiple players as well. This can be useful for local multiplayer when players are using different input devices or have custom key bindings.

The player index setting is an optional display setting that controls which player controller should be used when looking up key bindings. If a player controller cannot be found for the provided player index, the Input Prompt Image will be hidden.

4.5 Alternative Key Bindings

Since the input system supports having multiple key bindings for an action, the Input Prompt Display system supports it as well. This can be useful when needing to display alternative key bindings such as in a custom key binding menu.

The alt binding index setting is an optional display setting that controls which key binding for an action should be displayed. It should be noted that this feature coincides with composite bindings. For example, a 2D axis composite binding action could have a primary binding to the left analog stick for a gamepad controller as well as a secondary binding to the D-pad.

4.6 Control Scheme Dependent Display

The hide for control schemes setting is an optional display setting that controls hiding the Input Prompt Image when certain control schemes are in use. This can be useful for hiding input prompts in UI buttons for mouse or touch inputs.