InteractionKit - For Unity.

www.unitygamesdevelopment.co.uk

Created By Melli Georgiou

Table of Contents

Version History

ALWAYS BACKUP YOUR PROJECT BEFORE UPDATING!

v3.0.1
- Bugfix for Unity 2021 (automatic setting up of GameObject Icons would throw console errors).

v3.0
- Fully supports Unity's new Input System or the default system.
- Included demos automatically switch between default and new input systems.

v2.0.3
- Updated for Unity 2020.1
- Changed Local Variable Ints to use letter names to make them easier to understand.
- The "Activate GameObjects By Name" action now works as expected.
- The Message Canvas script in the demos now exposes its "ShowMessage" method to make it compatible with UnityEvents as well as custom SendMessage actions.

v2.0.2
- Updated for Unity 2019.3

v2.0.1
- Updated GUIDs of InteractionKit to work with other HellTap tools without conflicts.
- Unity 2017.4 (LTS) is now the minimum supported version.

v2.0
- New "Hold to Trigger" options on every interact screen. Allows you to setup interactions where the player has to hold the input down for a specific duration before triggering it.
- New Multi-cam / Player switcher demo which showcases switching between different character and camera rigs.

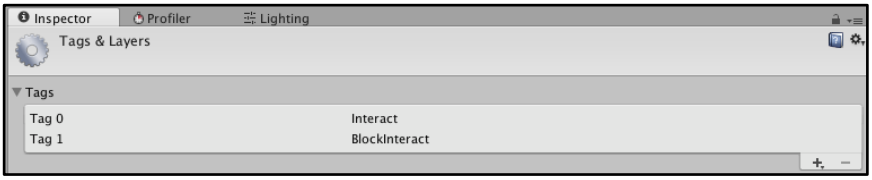
v1.5
- New Local Integer Variables! Each Interact component can use up to 10 number variables to simplify conditional events without using PlayerPrefs!
- New Local Integer Variable Actions! Set, add, subtract and randomize your variables using built-in actions!
- New Unity Event actions! You can now use Unity's built-in event system in every conditional event!

v1.1
- New detection modes for "Mouse Raycast 2D" and "Mouse Raycast 3D". Great for point and click based input!
- Audio Improvements: Slight fade-ins to reduce audio popping as well as a dedicated AudioSource for transitions.
- Fixed a bug where a custom icon would sometimes not appear.

v1.0
- First Commercial version of plugin.

Installing and Setting Up InteractionKit

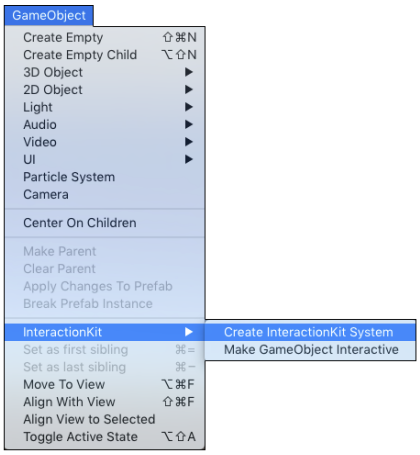
1) Install the package file into your project.
2) Create two new game tags called “Interact” and “BlockInteract”.



NOTE: InteractionKit requires the ‘Interact’ tag to work correctly.

3) Next, we need to create the InteractionKit System in every scene of our project. This handles the core system and UI.

   Create the InteractionKit object by clicking on “**GameObject > InteractionKit > Create InteractionKit System**” in the Unity menu. The “InteractionKit” object should appear in your scene already setup for you.



   NOTE: Typically, the InteractionKit GameObject should be saved as a prefab so that it is the same object in every scene!

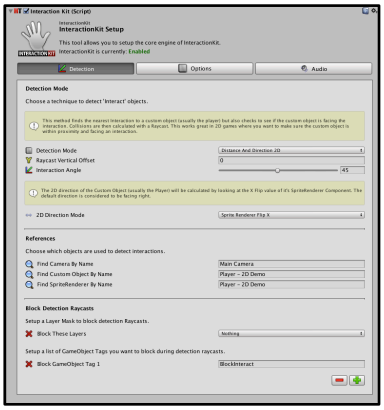
An Overview Of InteractionKit

What Is InteractionKit?

InteractionKit is an advanced solution for interacting with your game environment. No matter if you're working in 2D or 3D, the system handles everything from the UI, object detection, ray casting, input handling and even conditional events and actions through visual scripting to create rich, RPG-style interactions.

Any object can easily be made interactive by adding the **Interact** component to it. At runtime, these **Interact** components are pooled by the **InteractionKit** script, which handles all of the complicated detection, input and trigger code. Finally, the **InteractionKit GUI** is what handles the UI and allows for a variety of customizations in its appearance.

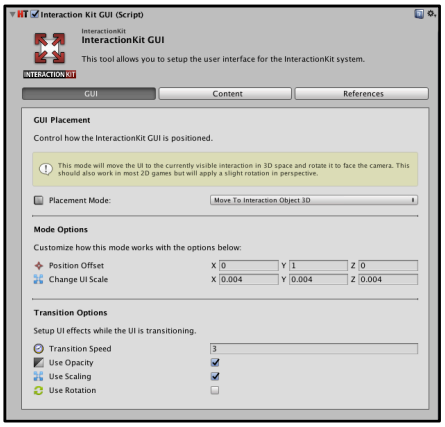
About The InteractionKit Script



The "InteractionKit" script is the engine behind the entire system. From this component, you can setup all of the technical aspects of how InteractionKit will work.

This includes the technique used for detecting interactions, as well as input and audio options.

About The InteractionKit GUI Script

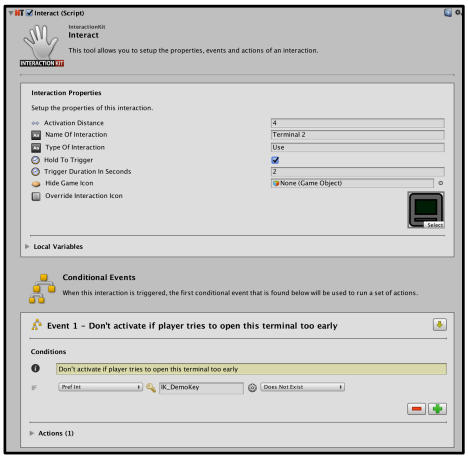


The InteractionKit GUI component is usually found on the same GameObject as the InteractionKit script.

As the name suggests, the InteractionKit GUI script allows you to setup a customized UI that represents interactions to the player.

UI Effects, placement, and hotkeys are all customizable in this screen.

About The Interact Script



The Interact script represents the individual interactions in the scene. They are setup in the Unity Hierarchy as children of valid objects (usually GameObjects with colliders).

The Interact screen allows you to describe the type of interaction as well as allowing you to create an unlimited number of conditional events related to it.

As you work with InteractionKit, you'll find this to be the component you use most. Its possibilities are vast and are explained in more detail in the 'Setting Up The Interact Script' chapter.

What Is The InteractionKit Workflow?

InteractionKit has a very streamlined approach. After setting up the core system and UI, you can make any object in the scene interactive by selecting "Make GameObject Interactive" in the menu. These scripts do more than simply describe the interaction in the UI; they act as the basis of an abstracted visual programming layer, which empowers you to easily create conditional events!

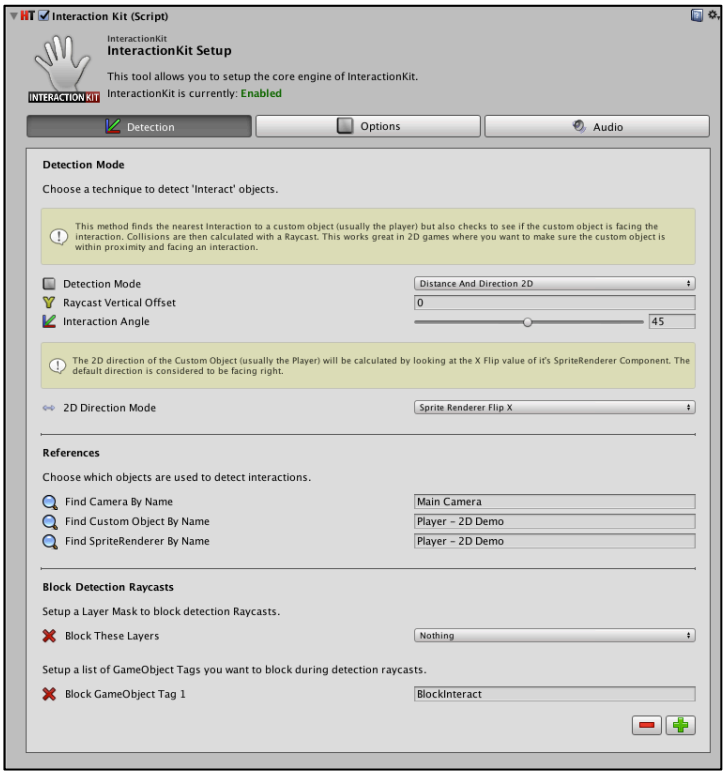
For example, you could add the Interact script to a door and create 2 conditional events. The first event checks to see if the player has the key to the door, and if they do have it, the door opens normally. If they don't, the second conditional event shows a message telling the user to go find the key.

What about RPG-like conversations? No problem! You add the Interact script to an NPC in the game. Then, create several conditional events to have them speak certain lines but only when something in the game has been completed. If the player hasn't completed level 1, the NPC says "Go do mission 1!" If the player has completed level 1, the NPC says "Go do mission 2!" and so on.

The awesome thing about InteractionKit is this happens all within the same Interact script, and the logic can be created visually in the easy to use editors by comparing local variables or PlayerPrefs (which is likely the way you are saving data anyway!). Both conditional logic and actions take place within the Interact object. If there is an action that InteractionKit doesn't have out of the box, it can call other scripts with its powerful SendMessage actions to have it run there instead!

It is worth mentioning that you can create an unlimited number of conditional events on each Interact script, making it possible to create powerful RPG-like sequences. The only limit is your imagination!

Setting Up The InteractionKit Component



The InteractionKit script allows you to link the system to your game and set it up in a way that will make sense. It is made of the Detection, Options and Audio tabs.

The Detection Tab

The first thing to do is choose a detection mode for InteractionKit to work with. Detection modes are essentially the technique that will be used to figure out if the player is interacting with an object.

The following detection modes are available:

Camera Raycast 3D
This technique uses a Raycast from the Main Camera. This works great with most first person and VR games.

Custom Raycast 3D
This technique uses a Raycast that originates from a custom object that you choose.

Distance 3D
This method finds the nearest Interaction closest to a custom object (usually the player). Collisions are then calculated with a Raycast. This works well for many 3rd person 3D games where only the proximity to an interaction is needed.

Distance And Direction 3D
This method finds the nearest Interaction to a custom object (usually the player) but also checks to see if the custom object is facing the interaction. Collisions are then calculated with a Raycast. This works well for many 3D games where you want to make sure the custom object is within proximity and facing an interaction.

Distance 2D
This method finds the nearest Interaction closest to a custom object (usually the player). Collisions are then calculated with a Raycast. This works great in 2D games where only the proximity to an interaction is needed.

Distance And Direction 2D
This method finds the nearest Interaction to a custom object (usually the player) but also checks to see if the custom object is facing the interaction. Collisions are then calculated with a Raycast. This works great in 2D games where you want to make sure the custom object is within proximity and facing an interaction.

Mouse Raycast 3D
This technique uses a Raycast that originates from the mouse. This works great for point and click games in a 3D environment.

Mouse Raycast 2D
This technique uses a Raycast that originates from the mouse. This works great for point and click games in a 2D environment.

These detection modes cover most types of games but you will need to choose the one that makes sense for your project. The great thing is it is easy to experiment with different detection modes to see what works best!

Depending on your chosen Detection mode, these additional options may appear:

Raycast Vertical Offset
When a Raycast is performed, a vertical offset can be applied to the starting point (usually the player) so that you can get more accurate simulations based on your game character.

Interaction Angle
When using 'Distance and Direction' detection modes, a dot product needs to be calculated to determine if the player is looking at the object. You can modify this value to make the calculation narrower or wider.

2D Direction Mode
When using the 'Distance and Direction 2D' mode, we need to tell InteractionKit how to check your player's direction. You can use 'Local Scale X', which checks to see if you've inverted the local X scale of the player to make the character appear flipped, or 'Sprite Renderer Flip X', which checks to see if the player's Sprite Renderer has the X flip value set.

It is worth mentioning that both techniques assume that the player is facing right by default.

References

After setting up how you want the detection mode to work, you will need to provide the following references:

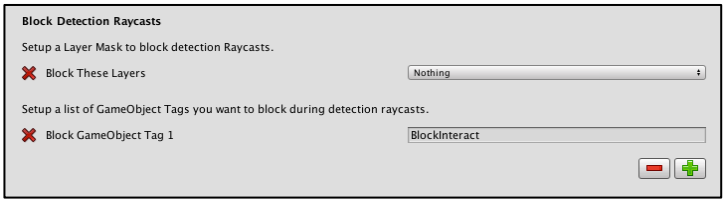
Find Camera By Name
The name of the Camera to use. This should usually be "Main Camera" unless you are using a custom camera setup.

Find Custom Object By Name
The Custom Object is usually looking for the name of the player. Unless you have a custom setup where Raycasts are preferred from a different object.

Find SpriteRenderer By Name
This option only appears when using the 'Distance and Direction 2D' detection mode with a 'Sprite Renderer Flip X' 2D Direction mode. It is asking you to provide the name of the Sprite Renderer to use when calculating if the player is flipped from left to right.

NOTE: These InteractionKit references are found by name so that you have the option to dynamically load objects at runtime and not necessarily have all items in the scene at once! It also makes the prefabs more resistant to missing references, etc.

Block Detection Raycasts



A powerful feature of InteractionKit is the ability to block interactions based on both layers and GameObject tags. If possible, using layers without tags will be the most efficient way of handling Raycasts.

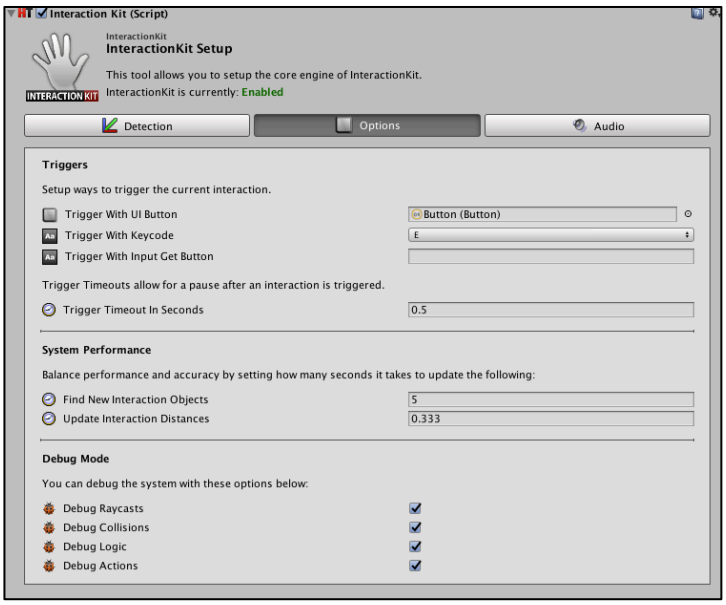
However, in complicated games it is possible to use up all Physics Layers on other things and InteractionKit allows you to workaround this limitation by using a mix of both if needed.

Block These Layers
This gives you a LayerMask for you to choose layers that will act as an obstacle when attempting to interact with objects.

Block GameObject Tags
This is a list of tag names that will act as an obstacle when Raycasts are being calculated. You can use the "+" and "-" buttons to add or delete tags.

NOTE: You can visually test the results of these settings in the 'Debug Mode' section of the Options tab!

The Options Tab



The Options tab allows you to setup how the player will trigger interactions. Settings to tweak system performance and debugging modes are also available.

Triggers

The Triggers section allows you to setup ways to "trigger" the current interaction via a range of input options.

Trigger With UI Button
You can set a reference to use a UI Button to trigger the active interaction. This is usually set as the Button in the InteractionKit GUI.

Trigger With Keycode
You can set a Unity Keycode to trigger an interaction with a specific key.

Trigger With Input Get Button
You can set a string to reference a Button from Unity's Input Manager.

Trigger Timeout In Seconds
It's a good idea to have a pause between interaction triggers to avoid actions executing too quickly. 0.5 seconds is recommended.

New Unity Input Triggers

If you are using Unity's new Input System, these options will be available:

Trigger PlayerInput Action
The name of the action used to trigger an Interaction. By default, this would be named 'Interact'.

PlayerInput Mouse Position
The name of the action that provides the position of the mouse.

System Performance

Internally, InteractionKit is frequently updating a list of possible interactions in the scene as well as the distances of each interaction in relation to the player. These updates can be tweaked to balance performance and accuracy. You should use a number to represent how many seconds between each update.

Find New Interaction Objects
This update attempts to recreate a list of the interactions that are currently in the scene. Interactions will automatically update the list when they are created or destroyed but this acts as a failsafe to detect or remove references in certain conditions. Every 5 seconds is generally a good balance for most applications. If your scene never deletes or creates Interactions dynamically, you could try using a much larger number (e.g. 999999) to not have to update the list during gameplay.

Update Interaction Distance
This update attempts to determine which interaction is closest to the player and whether or not the player is facing the interaction without being blocked by other objects. This is where Raycasts are performed and block Layers and Tags are factored in to the calculation. Different types of games will have different requirements. Generally speaking, FPS games will require a much higher frequency rate (smaller values like 0.2 – 0.333 is a good range). A Platform game, which only accounts for distance and not direction, could get away with a lower frequency (higher values like 0.5 - 1 second). It is worth clarifying that the smaller the number, the larger the performance cost as Raycasts can be expensive if over-used. It is best to play the game and tweak at the same time to find the best possible balance.

Debug Mode

InteractionKit offers simple checkboxes to enable debugging for specific areas:

Debug Raycasts
This shows Raycasts in the Scene view as green or red lines. A green line represents an active Raycast between the player and an interaction, and a red line shows an interaction being blocked by a Layer or GameObject Tag.

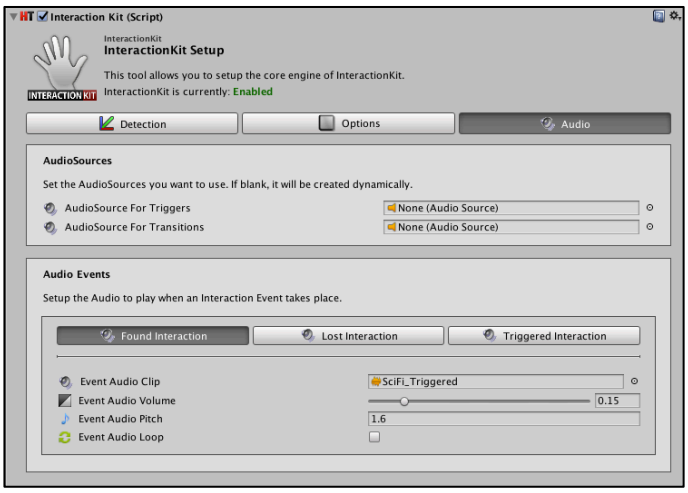
Debug Collisions
This shows information in the console about any blocked collisions. If an interaction is not becoming active, this will help you figure out what the problem is (usually an unexpected layer or tag).

Debug Logic
This shows information in the console about the conditional events in

'Interact' objects. This will help you figure out which events are firing and what the cause might be if a different event was expected.

Debug Actions
This shows information in the console about the actions run from a conditional event.

The Audio Tab



The Audio tab allows you to setup sound effects based on when an interaction is found, lost or triggered.

AudioSource

The AudioSource section allows you to specify a custom AudioSource to use when playing sound effects for triggers and transitions. If it is empty, one will be created automatically.

Audio Events

The Audio Events section offers new tabs for when an interaction is found, lost or triggered. Clicking on each tab allows you to set the following options:

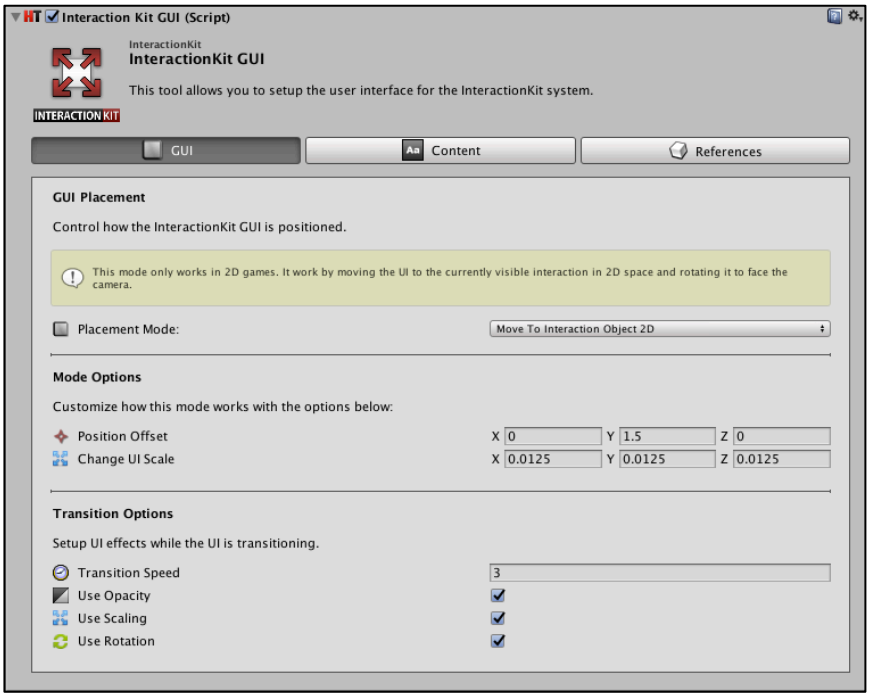
Event Audio Clip
The AudioClip to play when this event happens.

Event Audio Volume
The volume of the audio clip.

Event Audio Pitch
The pitch of the audio clip.

Event Audio Loop
Should this AudioClip be looped? This should usually be off at all times.

Setting Up The InteractionKit GUI Component



The InteractionKit GUI script allows you to setup and customize the UI of the system. It consists of the GUI, Content and References tabs.

The GUI Tab

The GUI Tab handles how to position the UI and setup transitions.

GUI Placement

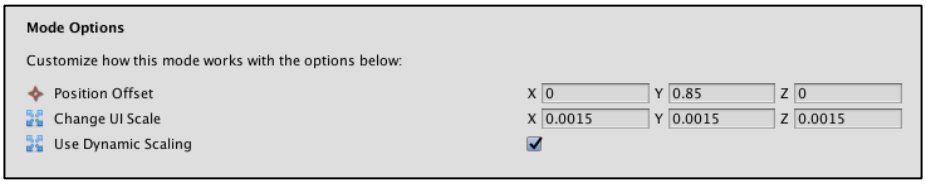
The first section of the GUI tab is the Placement mode, which allows you to choose if the UI should be positioned statically or over the interaction object. The following options are available:

Static
Static mode never moves the UI from the default position.

Move To Interaction Object 3D
This mode will move the UI to the currently visible interaction in 3D space and rotate it to face the camera. This should also work in most 2D games but will apply a slight rotation in perspective.

Move To Interaction Object 2D
This mode only works in 2D games. It works by moving the UI to the currently visible interaction in 2D space and rotating it to face the camera.

Mode Options



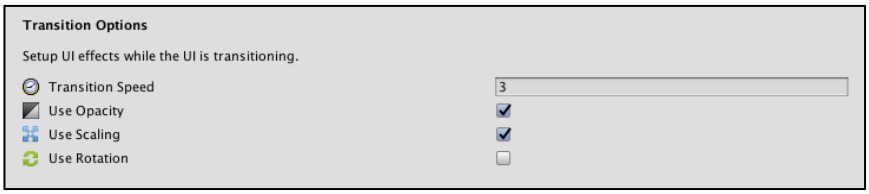
If you are moving the UI to the interaction, the following options are available:

Position Offset
You can set a custom positional offset when the UI is placed over the interaction in the scene.

Change UI Scale
You can override the scale of the interface by adding values here.

Use Dynamic Scaling
If you're using "Move To Interaction Object 3D" mode, you can enable dynamic scaling to make the UI appear to be the same size while placed in 3D space. It does this by comparing the distance between the Camera and the interaction.

Transition Options



In the transition Options section, you can setup UI effects while the UI is transitioning:

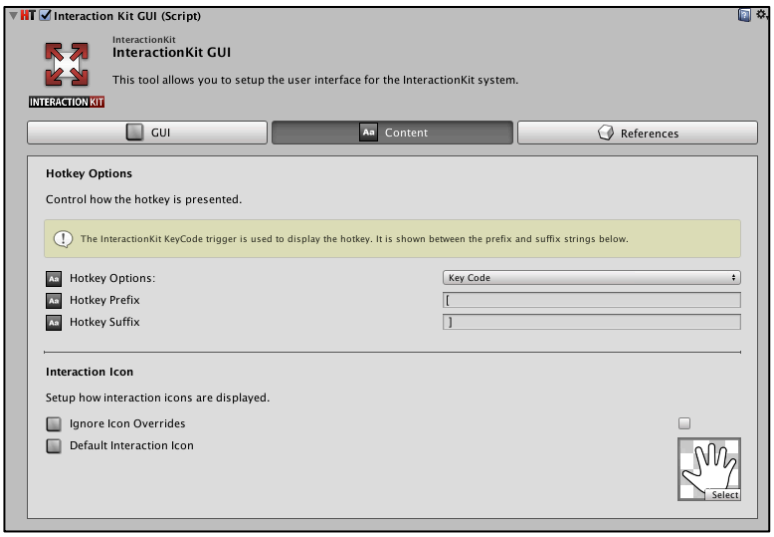
Transition Speed
The speed it takes to show the UI (bigger values are faster).

Use Opacity
Should transparency be faded in and out with the UI?

Use Scaling
Should the UI be scaled in and out?

Use Rotation
Should the UI be rotated when transitioning in and out?

The Content Tab



The Content Tab allows you to setup how hotkeys are displayed and the default options for handling interaction icons.

HotKey Options

You can setup exactly how InteractionKit displays hotkeys in the UI. The following options are available:

HotKey Options
You will need to choose a primary input method for displaying hot keys. The 'Key Code' option uses the Input KeyCode used in the InteractionKit script, the 'Get Button' option uses the Input Get Button used in the InteractionKit script, and 'Custom String' allows you to enter a custom string to be displayed instead.

HotKey Prefix
You can enter some text to be displayed before the HotKey.

HotKey Suffix
You can enter some text to be displayed after the HotKey.

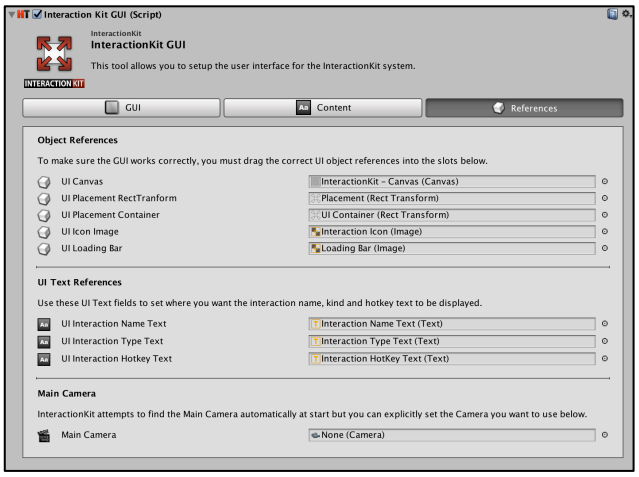
Interaction Icon

In this section you can setup how InteractionKit displays icons in the UI:

Ignore Icon Overrides
If checked, individual 'Interact' scripts cannot override the default icon.

Default Interaction Icon
This is the default interaction icon displayed in the UI if not overridden by 'Interact' scripts.

The References Tab



The References Tab is where you connect the InteractionKit GUI script with all the references of the UI (this is setup by default when creating the InteractionKit System from the menu).

Object References

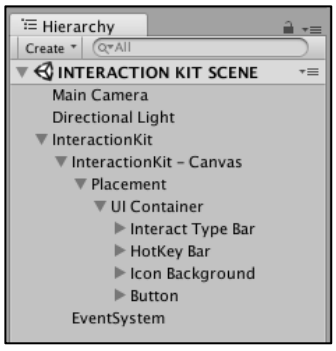
These references point to objects needed by the UI. The list is as follows:

UI Canvas: The Canvas component of the InteractionKit UI.
UI Placement RectTransform: The RectTransform used to position the UI.
UI Placement Container: The RectTransform used to apply transitional effects to the UI.
UI Icon Image: The Image component to use for interaction icons.
UI Loading Bar: The Image component used for hold-to-trigger durations.

UI Interaction Name Text: The Text component to use for displaying interaction names.
UI Interaction Type Text: The Text component to use for displaying interaction types.
UI Interaction Hotkey Text: The Text component to use for displaying the hotkey.

Main Camera: If you would like to use a custom camera for the UI, you can set it in this field. If blank, Camera.main will be used automatically.

Customizing The UI

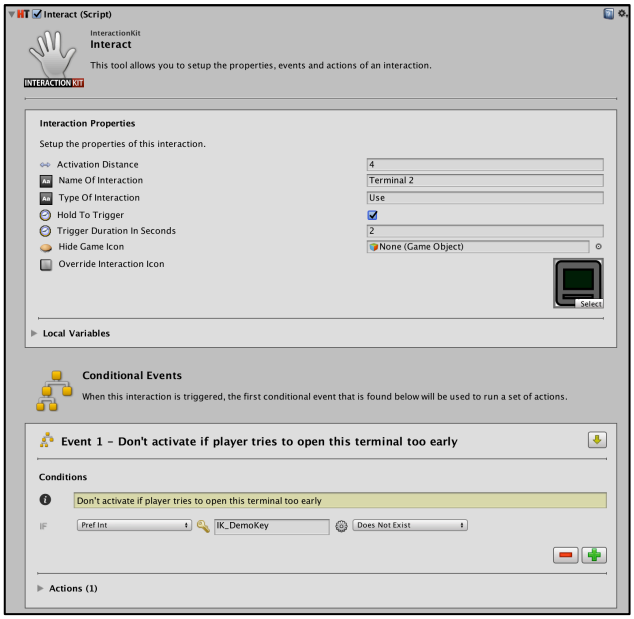


For users familiar with Unity's UI system, the UI can be customized relatively easily. Just be sure you keep to the same general structure that exists.

In other words, you should make sure that the Canvas, Placement and UI Container objects are setup in the same kind of layout. Pay attention to the pivot points of the RectTransforms and feel free to experiment.

It's far easier to change the colors of the UI objects, replace textures and hide existing sections. For example, simply disabling the "Icon Background" object will totally hide the icons in the UI with little effort.

Setting Up The Interact Component



In InteractionKit, the Interact script is what makes objects interactive. It is divided into the "Interaction Properties" section, and the "Conditional Events" section.

Interaction Properties

Interaction Properties can be thought of as metadata for the interactive object. The following fields are available:

Activation Distance
How close the player needs to be to the interaction to make it active.

Name Of Interaction
The name of the Interactive object.

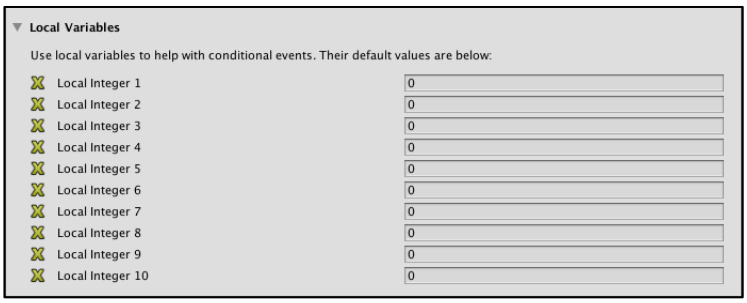
Type Of Interaction
The type of the Interactive object. Example: a verb such as "Use", "Talk", etc.

Hold To Trigger / Trigger Duration In Seconds
Allows you to force the player to hold down the input for a specified number of seconds in order to trigger the interaction.

Hide Game Icon
You can set another GameObject to act as an indicator for this interaction. When this interaction becomes active, it will hide the GameObject until it is no longer active. An example of this in use can be found in the 2D and 3rd person 3D InteractionKit Demos.

Override Interaction Icon
Use a specific icon in the UI when this interaction becomes active.

Local Variables

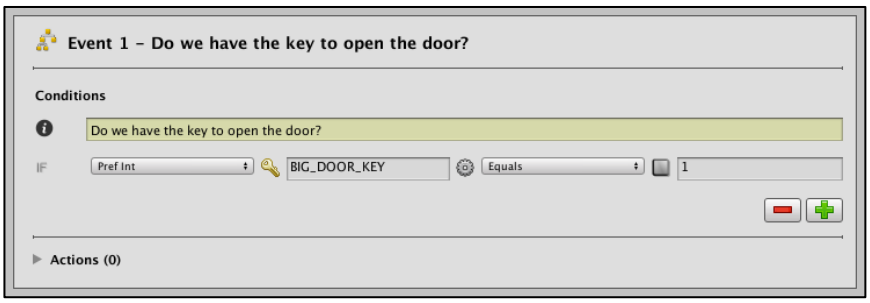


The Local Variables section opens up 10 integers (numbers) that are local only to the current Interact screen. You can setup default values for each one and even track / edit them while the application is running so you can visually see what's going on as you develop your games.

These are often easier to use for tracking sequences such as an NPC's dialogue or to track if items were found, etc. The benefit of using Local Integer Variables is that it's easier than using PlayerPrefs to track simple sequences and game states.

However, it's worth noting that these changes won't be saved across scenes. To have global values tracked you should use PlayerPrefs instead.

Conditional Events



Conditional Events consist of a list of conditions and actions. Firstly, the list of conditions are tested when the interaction is triggered, and if found to be true, it runs a series of actions. A custom-built visual scripting tool is included right in the editor to make it extremely easy to create Conditional Events!

In the example above, we have put a Conditional Event on a door. It contains only one condition. We are checking an Integer (number) saved in PlayerPrefs to see if we have the "BIG_DOOR_KEY" to open the door. We represent this in our game by saving the PlayerPrefs integer "BIG_DOOR_KEY" to 1.

If the BIG_DOOR_KEY is set to 1, it means the door can be opened and this Event will run its actions. Otherwise, this event will be ignored and any other events remaining will be tested instead.

Conditional Events have a comments field that can be renamed to make them easier to understand. You can add as many conditions to an event as you wish and alternatively, you can remove every condition to make an event always trigger without testing anything.

Conditions

Conditions are built up of the following:

Type Of Condition
We first choose a condition type to test. The current types available are PlayerPrefs String (Pref String), PlayerPrefs Float (Pref Float), PlayerPrefs Integer (Pref Int) and Local Integer Variable (Local Int).

Condition Key
We need to tell InteractionKit the name of the PlayerPrefs Key we want to test. When using Local Ints a dropdown list is displayed instead.

Condition Operator
We must then choose the kind of test we want to perform. We can check if something 'Exists', 'Does Not Exist', 'Equals', 'Is Not', 'Greater Than' or is 'Less Than' a value we define.

Condition Value
The value we are testing against.

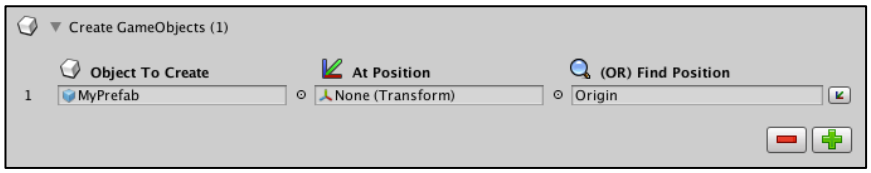
Actions

All Conditional Events have an Actions foldout in the bottom left which allows us to bring interactions to life.

We have access to plenty of powerful built-in actions allowing us to create, activate, de-activate and destroy GameObjects. We can also send information to the console, set PlayerPrefs directly, change interaction properties, trigger code in different scripts and navigate to other Unity scenes.

The following actions are available:

Create GameObjects



A GameObject / prefab can be created from the Project pane and placed at a position either at a direct reference, or to be found by searching for its name. Objects can be created at the origin (Vector3.zero) by clicking the small button on the right.

Activate GameObjects

Drag a GameObject from the Hierarchy into the list to activate it.

Activate GameObjects By Name

Enter the name of a GameObject to find and activate at runtime.

De-Activate GameObjects

Drag a GameObject from the Hierarchy into the list to de-activate it.

De-Activate GameObjects By Name

Enter the name of a GameObject to find and de-activate at runtime.

Destroy GameObjects

Drag a GameObject from the Hierarchy into the list to destroy it.

Destroy GameObjects By Name

Enter the name of a GameObject to find and destroy at runtime.

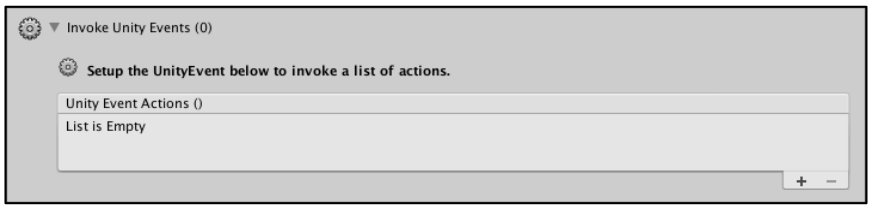
Show Messages In Console

Enter the some text to display in the console.

Send Message To Other GameObjects

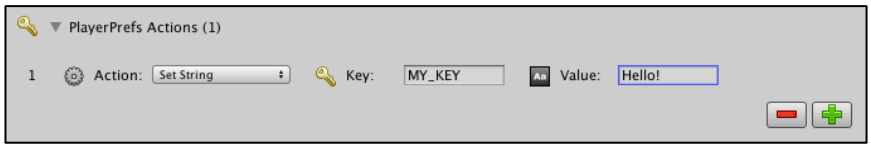
You can trigger functions in other scripts by using this action group. You can drag a GameObject to send the message to directly, or find it by name. You must then enter its method (function) name and setup any arguments to send. Supported arguments are None (void), String, Integer, Float, Boolean, GameObject and Transform.

Invoke UnityEvent Action



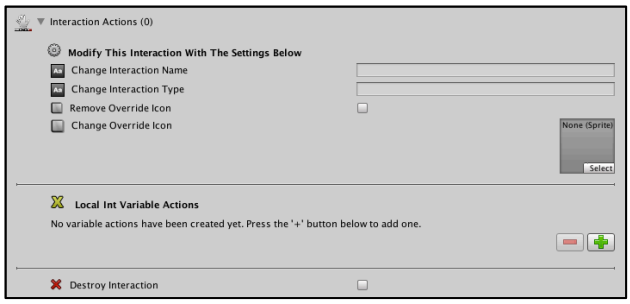
You can setup UnityEvent actions using Unity's own Event system.

PlayerPrefs Actions



Powerful PlayerPrefs actions can be used with this action group. You can Set a PlayerPrefs String, Float or Int; add or subtract to a PlayerPrefs float or int; delete a specific PlayerPrefs Key; or delete all PlayerPrefs keys. The layout is similar to the visual scripting in conditional events.

Interaction Actions



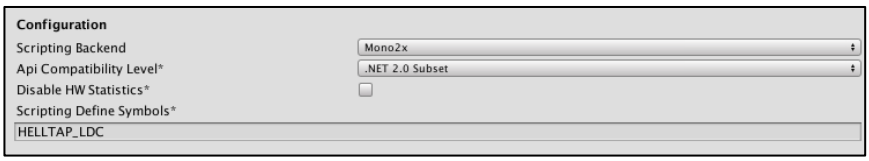
This action group allows you to change the properties of the current Interaction. You can change the name, type, and icon or choose to destroy the interaction completely.

Additionally, you can create Local Int Variable Actions. These allow you to set, add, subtract or randomize the component's local integer variables.

Navigation Actions

This action group allows you to easily restart the current level or load another Unity scene by name.

3rd Party Actions



InteractionKit also has built-in support for our Localized Dialogs & Cutscenes (LDC) plugin!

If you want access to these extra features you must first make sure that you have already installed the LDC package and added "HELLTAP_LDC" as a scripting define symbol in the Unity Player settings.

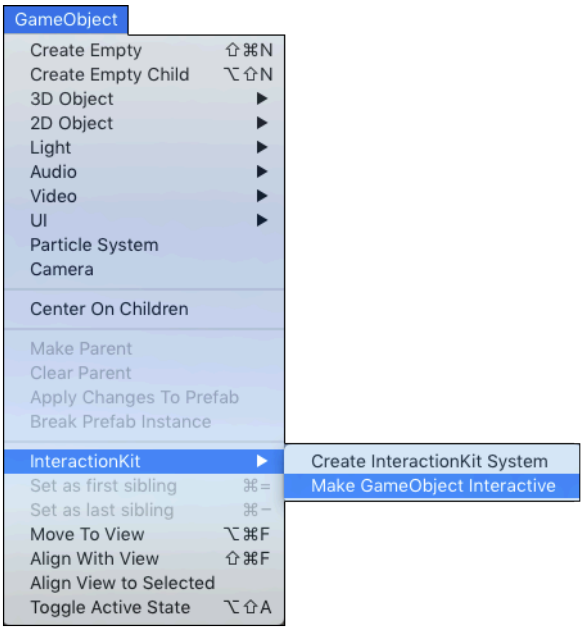
This will enable the following actions in the Interact screen:

LDC Actions

This action group allows you to play an LDC prefab in a variety of ways:

> *PLAY DIALOG CONTROLLER*
> *This allows you to drag in a DialogController to play when the interaction is triggered.*
>
> *INSTANTIATE DIALOG FROM PREFAB*
> *This option allows you to drag in a prefab from the Project pane to play when the interaction is triggered. You should make sure that it is set to Auto-Play and that it will destroy itself when it finishes.*
>
> *You also have the handy option to disable this interact component while the dialog exists.*
>
> *PLAY A RANDOM DIALOG CONTROLLER*
> *This option allows you to setup a list of DialogControllers, which will be chosen randomly when the interaction is triggered. Specifically, the first dialog is chosen at random, and then it cycles through them in order so you won't hear the same dialogs in a row.*

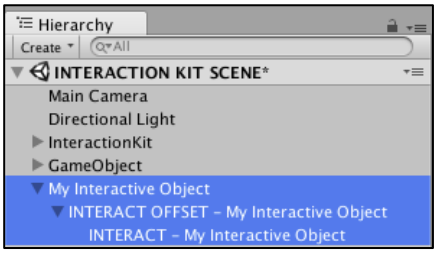
Important Notes On Making GameObjects Interactive



To make a GameObject interactive, you should use the menu to set it up correctly.

Select a GameObject in the Hierarchy (it should usually have a collider) and in the menu, select:

GameObject > InteractionKit > Make GameObject Interactive.



You will notice that the command not only creates a new INTERACT child object that represents the interaction, but an INTERACT PIVOT object which acts as another parent in between.

The purpose for this convention is to support some of the detection mode techniques which allow you to modify the exact point of the interaction separately from the actual 3D model / collider you are using.

The benefit of this is it allows you to switch between different detection techniques at any point without breaking anything.

It is highly recommended to not change this manually, even if things appear to work.

InteractionKit API

InteractionKit has methods and events that are available via the API. Remember to add "**using HellTap.InteractionKit;**" in C# scripts or "**import HellTap.InteractionKit;**" in Unityscripts.

InteractionKit Events

```
// Triggered when a new interaction becomes active (or an active interaction is changed)
InteractionKit.OnInteractionChanged

// Triggered when an active interaction is lost
InteractionKit.OnInteractionLost

// Triggered when an active interaction is triggered
InteractionKit.OnInteractionTriggered
```

Accessing Instances & Interactions

```
// Advanced users can access the instances of the InteractionKit scripts like this:
InteractionKit.com
InteractionKitGUI.com

// Access the currently active interaction by using this static Interact variable:
InteractionKit.current

// Access the currently active interaction's parent with this static Transform variable:
InteractionKit.currentParent
```

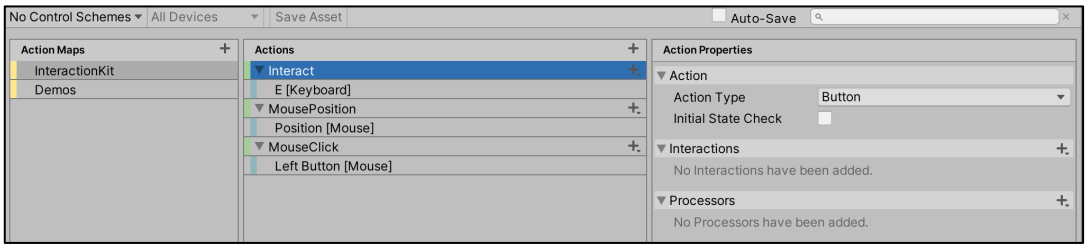
InteractionKit Sample Code

```
// Manually enable / disable interactions using this static Boolean.
InteractionKit.canInteract = true;

// Force an update of all interaction objects on the next frame (This shouldn't be needed).
InteractionKit.UpdateInteractionObjects();

// Example of changing blocked GameObject Tags at runtime (via the instance '.com'):
InteractionKit.com.blockTags = new string[]{ "Tag 1", "Tag 2" };
```

InteractionKit Demos

Some cool tricks and workflow approaches are included in the InteractionKit package demos. Feel free to look through them and some of the sample code to get an idea of what is possible. It's often the best way to learn!

Using Unity's New Input System



As of InteractionKit v3.0, users have the option of using Unity's new Input System.

Default Setup

If you are happy to use the default setup for InteractionKit ('E' button for interactions), no extra work is needed as InteractionKit will automatically build a default Input Action setup for you.

New Input System

To use your own InputActions asset, follow these steps:

1) On the InteractionKit GameObject, add a PlayerInput component.
2) Drag your InputActions asset into the Actions slot.

In your InputActions asset, it is recommended you setup an Action Map named 'InteractionKit' to keep things nicely organized.

It is recommended to setup the following actions:

Interact
In order to trigger an interaction, you should setup an 'Interact' action as a Button action type. This can be renamed in the InteractionKit options tab.

MousePosition
In order to help InteractionKit with mouse-based interactions, the 'MousePosition' action is setup as a Vector2 value. Bind this to the mouse position. This can be renamed in the InteractionKit options tab.

MouseClick (optional)
If using the demo scenes or if your input changes from scene to scene, it is sometimes useful to also provide a 'MouseClick' action as a Button action type. Bind this to the left mouse button.

Support

If you need any assistance or have suggestions for this plugin, feel free to visit our website at:

www.unitygamesdevelopment.co.uk

I hope you find this system useful, as I have in my own personal projects! =)

All the best!

- Mel