cInput Reference Manual

by cMonkeys

Note

cInput 2 is a commercial release and is not free.

We tried to keep the price low so everyone can still afford it. If for some reason you can not afford the price you can use Custom InputManager 1.x which still works fine. It can be found on the UnifyCommunity Wiki.

If you really need to use cInput 2 and can't afford the price then drop us an email with your motivations and we'll see if we can get you a freebie.

Get cInput 2 from our website or from the Asset Store!

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

How to Get Started With cInput

Initial Setup

1. Use the "Edit -> Project Settings -> cInput -> Replace InputManager.asset file" menu command in the Unity Editor to create an InputManager.asset file designed to work with cInput.
2. Place *cInput.cs* somewhere in your Assets/Plugins folder (or a subfolder of that directory) and it will automatically be accessible from your other scripts. This step should happen automatically if you import the unitypackage file.
3. cInput comes with some optional helper classes that can make it easier to do certain things such as the Keys class which gives you AutoComplete on key names. These should also go somewhere within the Assets/Plugins folder. Again, this should happen automatically if you import the unitypackage file.

Setting Up the Default Keys

The first thing to do is to create a setup script that will run run once before any of your other scripts try to get input from cInput. Define your keys first (using SetKey) and then any axes (using SetAxis) in the Awake() or Start() function of that setup script. This script only needs to run once, so if you put it in the first scene the player will see (e.g., splash screen or loading screen) then you don't need to include it in any other scenes. cInput will persist when loading new scenes.

```
// this creates a new key called "Left" and binds A and the Left Arrow to the key
cInput.SetKey("Left", "A", "LeftArrow");
// this uses Keys.D and Keys.RightArrow to accomplish something similar
cInput.SetKey("Right", Keys.D, Keys.RightArrow);
// uses "Left" and "Right" inputs previously defined with SetKey to create a new axis
cInput.SetAxis("Horizontal Movement", "Left", "Right");
```

For a full list of acceptable inputs see the Valid Inputs section. Additionally, you can use the included Keys class (cKeys.cs) to let autocomplete/Intellisense help you type in the correct keys. Many of the examples in this documentation take advantage of the Keys class that comes in the cKeys.cs file.

A Note About Xbox 360 Controllers

Xbox 360 controllers have the unfortunate "feature" of both triggers being mapped to the same axis on Windows, which means that if you push both triggers at the same time, they cancel each other out and Unity can't tell that you're pushing either trigger. Thankfully, the triggers are also mapped separately on different axes if you have the driver installed. But there are two further complications:

1. If more than one Xbox 360 controller is connected, the triggers may or may not work properly, or may or may not be mapped to these separate axes as previously described. Recent versions of Unity have taken steps to correct this issue, though some problems may still exist. More details about this problem can be found here: http://goo.gl/LQFt9z

Table of Contents

2. Another complication with using Xbox 360 controllers is that the buttons and axes are mapped differently depending on the OS. The Keys class attempts to assist with this, by providing special keys specifically for the Xbox 360 buttons and joysticks.

The Keys class has entries for up to 4 Xbox 360 controllers, following a similar format as follows:

Xbox1A	Xbox1BumperLeft	Xbox1LStickLeft
Xbox1B	Xbox1BumperRight	Xbox1LStickRight
Xbox1X	Xbox1TriggerLeft	Xbox1LStickUp
Xbox1Y	Xbox1TriggerRight	Xbox1LStickDown
Xbox1Back	Xbox1DPadLeft	Xbox1RStickLeft
Xbox1Start	Xbox1DPadRight	Xbox1RStickRight
Xbox1LStickButton	Xbox1DPadUp	Xbox1RStickUp
Xbox1RStickButton	Xbox1DPadDown	Xbox1RStickDown

For the 2nd, 3rd, or 4th Xbox 360 controllers, you would use Xbox2, Xbox3, or Xbox4 at the beginning respectively.

Using Modifier Keys

cInput supports the use of modifier keys. You can designate a key to be used as a modifier with AddModifier. Note that once a key has been designated as a modifier, it is forbidden from being used as an input all by itself. In other words, if you designate LeftShift as a modifier, then it can be used in multiple key combinations such as LeftShift+T and LeftShift+1, but it cannot be used alone as just LeftShift. To define a key to use a modifier by default, set it with SetKey, passing in the modifiers you want to use.

Modifier keys do not work with axes. If you use SetKey to define an action that uses a modifier, then use that action with SetAxis to create an axis, the axis won't care if the modifier is pressed or not.

If you have added a key as a modifier and would like to remove it from being used as a modifier, use RemoveModifier.

Using cInput to Control Your Game

Once your inputs are defined in your setup script, you'll probably want to use those inputs to control your game. This is very simple to do using cInput's GetKey, GetKeyDown, GetKeyUp, or GetAxis functions, which all work in the same way that Unity's Input functions of the same name do. For example:

```
// this will only trigger once each time the "Jump" button is pressed
if (cInput.GetKeyDown("Jump")) {
        // jumping code here
}
```

Table of Contents

```
// this will trigger repeatedly while the "Shoot" button is held down
if (cInput.GetKey("Shoot")) {
        // shooting code here
}

// this will trigger once each time the "Grenade" button is released
if (cInput.GetKeyUp("Grenade")) {
        // throw grenade code here
}

// movement based on analog or virtual analog axis
float dx = cInput.GetAxis("Horizontal");
```

Changing the Keys from the Defaults

Changing Keys From Script

For information on how to change keybindings from script, see the ChangeKey documentation.

Using the Built-In GUI

cInput comes with a built-in GUI to make it simple to change which inputs are bound to which actions. This is great for rapid prototyping since it allows you to adjust the controls without having to invest a lot of time into a custom GUI. Simply call cGUI.ToggleGUI() to show or hide the built-in GUI menu, and assign a GUISkin to cGUI.cSkin to change the way it looks. cInput includes a GUISkin for your convenience. You can also optionally change the color/alpha of the GUI using cGUI.bgColor. For more specific information on how to use cGUI, please see the included cGUI reference manual.

Making a Custom GUI Menu

Admittedly, even though you can customize the look of the built-in GUI with cSkin, it still won't fit the design or theme for every game. With that in mind, there are functions included in cInput which will assist in the process of making your own custom menu for changing your controls.

A quick note: explaining how to make a GUI in Unity is beyond the scope of this document. To learn more about how to make GUIs in Unity, see the GUI Scripting Guide section in the online Unity Reference Manual. This section will only explain the methods that cInput provides which will assist you in making your own GUI menu for displaying and changing the keybindings in your game. Also, note that if you make your own GUI, cInput's included GUI component should not degrade performance when it is unused. If you have access to the source code (in cInput Pro) you can prevent cInput from adding the GUI component at all by commenting out the first line which should look something like this:

```
#define Use_cInputGUI // Comment out this line to use your own GUI instead of cInput's built-in GUI.
```

There a few methods and properties you will likely want to use when creating a GUI menu to display the current keybindings and allow players to change them or reset them back to defaults. They are the

Table of Contents

GetText, ChangeKey, ResetInputs functions and the length property. You may also be interested in the OnKeyChanged event.

Basically, the idea is to use a *for* loop to iterate through all of the inputs and create buttons the player can click on in order to change the keybindings while in game. For example:

```
private Vector2 _scrollPosition = new Vector2();

void OnGUI() {
	_scrollPosition = GUILayout.BeginScrollView(_scrollPosition);
	GUILayout.BeginHorizontal();

	GUILayout.BeginVertical();
	GUILayout.Label("Action");
	for (int n = 0; n < cInput.length; n++) {
		GUILayout.Label(cInput.GetText(n, 0));
	}
	GUILayout.EndVertical();

	GUILayout.BeginVertical();
	GUILayout.Label("Primary");
	for (int n = 0; n < cInput.length; n++) {
		if (GUILayout.Button(cInput.GetText(n, 1)) && Input.GetMouseButtonUp(0)) {
			cInput.ChangeKey(n, 1);
		}
	}
	GUILayout.EndVertical();

	GUILayout.BeginVertical();
	GUILayout.Label("Secondary");
	for (int n = 0; n < cInput.length; n++) {
		if (GUILayout.Button(cInput.GetText(n, 2)) && Input.GetMouseButtonUp(0)) {
			cInput.ChangeKey(n, 2);
		}
	}
	GUILayout.EndVertical();

	GUILayout.EndHorizontal();
	GUILayout.EndScrollView();

	GUILayout.BeginHorizontal();
	if (GUILayout.Button("Reset to Defaults") && Input.GetMouseButtonUp(0)) {
		cInput.ResetInputs();
	}

	if (GUILayout.Button("Close") && Input.GetMouseButtonUp(0)) {
```

Table of Contents

```
                cInput.ShowMenu(false);
        }
        GUILayout.EndHorizontal();
}
```

OnKeyChanged Event

cInput fires an event whenever the ChangeKey function which waits for player input, or the ResetInputs functions are called. Note that it will be called even if the keys stay the same (such as from cancelling a ChangeKey action). An example for why you might want to use the OnKeyChanged event would be to run a function to update the text labels on a custom made GUI. Here's some code showing one such example:

```
// subscribe to the OnKeyChanged event
void OnEnable() {
        cInput.OnKeyChanged += UpdateGUITexts;
}

// unsubscribe to the OnKeyChanged event so we don't cause errors
void OnDisable() {
        cInput.OnKeyChanged -= UpdateGUITexts;
}

void UpdateGUITexts() {
        // put your code here to update the GUI texts when the OnKeyChanged event is fired
}
```

General Tips and Tricks

If you encounter odd behavior while developing with cInput, it may be helpful to make a call to Clear once to clear out any misconfigured cInput settings.

cInput should execute before other scripts to make sure that all inputs are updated for the current frame before your other scripts try to access them. By default, this should automatically be handled for you when you import the cInput UnityPackage. But it's something to be aware of if the script execution order is somehow lost or modified.

Table of Contents

Script Reference

Variables and Properties

allowDuplicates

bool **allowDuplicates**

Description

If set to true, cInput will allow the same inputs to be used for multiple actions.

anyKey

bool **anyKey**

Description

Is any key, mouse button, or key defined with SetKey currently held down? (Read Only)

anyKeyDown

bool **anyKeyDown**

Description

Returns true the first frame the user hits any key, mouse button, or key defined with SetKey. (Read Only)

deadzone

float **deadzone**

Description

Values less than this will register as 0 on the virtual axis. Default value is 0.001f.

externalInputs

string externalInputs

Description

A string containing all the cInput settings. **Read-only**. Use this with LoadExternal to save and load

Table of Contents

cInput settings somewhere other than PlayerPrefs.

gravity

float **gravity**

Description

How fast the virtual axis value will return to 0. Default value is 3.0f.

length

int **length**

Description

How many keys have been defined using SetKey. **Read-only**. Useful for making a custom GUI menu.

scanning

bool **scanning**

Description

Whether or not cInput is currently scanning for a keypress/input to bind to an action. Useful for making a custom GUI menu.

sensitivity

float **sensitivity**

Description

How fast the virtual axis value will reach 1. Default value is 3.0f.

usePlayerPrefs

bool **usePlayerPrefs**

Description

Should cInput automatically save and load to/from PlayerPrefs? Default value is true.

Table of Contents

Script Reference

Methods

AddModifier

AddModifier(modifierKey: *KeyCode***) :** *void*
AddModifier(modifier: *string***) :** *void*

Parameters

modifierKey	They keycode of the key to be used as a modifier.
modifier	They string name of the key to be used as a modifier. Allows you to use the Keys class.

Description

Designates **modifier** or **modifierKey** to be used as a modifier. Note that a modifier key cannot be used as a standalone input key.

Example

```
// allows LeftShift to be used as a modifier key
cInput.AddModifier(KeyCode.LeftShift);
```

AxisInverted

AxisInverted(description: *string*, **[inversionStatus:** *bool*]**) :** *bool*
AxisInverted(descriptionHash: *int*, **[inversionStatus:** *bool*]**) :** *bool*

Parameters

description	The name/description of the axis you want to invert.
descriptionHash	The hashcode of the name/description of the axis you want to invert.
inversionStatus	If true, axis will be inverted. If false, axis will not be inverted.

Returns

bool	Whether or not **axisName** is inverted.

Description

Inverts the **axisName** axis. If **inversionStatus** is not passed in, AxisInverted will simply return the inversion status of **axisName**. If **inversionStatus** is passed in, then this axis's inversion will be set to the boolean value and return that value. For more information, see Making a Custom GUI Menu.

Table of Contents

Example

```
// this toggles the inversion status of "Horizontal"
cInput.AxisInverted("Horizontal", !cInput.AxisInverted("Horizontal"));
// this makes a toggle button in the GUI
cInput.AxisInverted("Horizontal", GUILayout.Toggle(cInput.AxisInverted("Horizontal"), "Invert Axis"));
```

Calibrate

Calibrate() : *void*

Description

This calibrates analog inputs to their default/neutral position.

ChangeKey

ChangeKey(name: *string*, **[input:** *int*], **[mouseAx:** *bool*], **[mouseBut:** *bool*],
[joyAx: *bool*], **[joyBut:** *bool*], **[keyb:** *bool*]) **:** *void*
ChangeKey(index: *int*, **[input:** *int*], **[mouseAx:** *bool*], **[mouseBut:** *bool*],
[joyAx: *bool*], **[joyBut:** *bool*], **[keyb:** *bool*]) **:** *void*
ChangeKey(name: *string*, **primary:** *string*, **[secondary:** *string*], **[primaryModifier:** *string*],
[secondaryModifier: *string*]) **:** *void*

Parameters

name	The name/description of the key you wish to change.
index	The index number of the key you wish to change. Useful for custom GUIs.
input	Which input to change. 1 for primary (default). 2 for secondary.
mouseAx	Use mouse axes for input? Defaults to *false*.
mouseBut	Use mouse buttons for input? Defaults to *true*.
joyAx	Use gamepad axes (analog sticks/buttons) for input? Defaults to *true*.
joyBut	Use gamepad buttons for input? Defaults to *true*.
keyb	Use keyboard buttons for input? Defaults to *true*.

Description

Waits for player input, then assigns that input to trigger this key. All inputs are monitored by default except mouse axes. All arguments except **name** or **index** are optional and will use their default values if not explicitly passed in. Note that you can also use ChangeKey in the same way as SetKey.

Example

```
// the next input pressed will be assigned as the primary input for "Pause"
cInput.ChangeKey("Pause");
cInput.ChangeKey("Pause", 1); // this does the same thing as the previous line
// the next input pressed will be assigned as the secondary input for "Pause"
cInput.ChangeKey("Pause", 2);
// only gamepad axes and buttons will be accepted for the primary "Accelerate" input
cInput.ChangeKey("Accelerate", false, false, true, true, false);
```

Table of Contents

```
// only the keyboard can be used for the secondary "Jump" input
cInput.ChangeKey("Jump", 2, false, false, false, false, true);
// using ChangeKey like SetKey to change the primary and secondary inputs for "Up"
cInput.ChangeKey("Up", "W", Keys.UpArrow);
```

Clear

Clear() : *void*

Description

Clears all data stored by cInput from PlayerPrefs. This can fix problems that may occur when changing the number or order of inputs used by cInput.

ForbidKey

ForbidKey(key: *KeyCode***) :** *void*
ForbidKey(keyString: *string***) :** *void*

Parameters

key	The KeyCode of the key to forbid.
keyString	A string representing the KeyCode of the key to forbid.

Description

Forbids the key from being used as input for cInput.

Example

```
// forbid the use of the number 1 as an input
cInput.ForbidKey(KeyCode.Alpha1);
// forbid the use of "tab" and "space" as an input
cInput.ForbidKey(Keys.Tab);
cInput.ForbidKey("Space");
```

GetAxis

GetAxis(description: *string***) :** *float*
GetAxis(descriptionHash: *int***) :** *float*

Parameters

description	The name of the axis, as defined in SetAxis.
descriptionHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

float	A value between -1 and 1 inclusive.

Table of Contents

Description

Returns the value of the axis or virtual axis.

Example

```
// move the transform horizontally with the "Horizontal Movement" axis
float horizMovement = cInput.GetAxis("Horizontal Movement");
float h = 60 * horizMovement * Time.deltaTime;
transform.Translate(h, 0, 0);
```

GetAxisDeadzone

GetAxis(description: *string***) :** *float*
GetAxis(descriptionHash: *int***) :** *float*

Parameters

description	The name of the axis you want to retrieve the deadzone value for.
descriptionHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

float	The deadzone value for **axisName**.

Description

Returns the deadzone value for **axisName**.

Example

```
// gets the deadzone value for a previously defined axis
float hDeadzone = cInput.GetAxisDeadzone("Horizontal Movement");
```

GetAxisGravity

GetAxisGravity(description: *string***) :** *float*
GetAxisGravity(descriptionHash: *int***) :** *float*

Parameters

description	The name of the axis you want to retrieve the gravity value for.
descriptionHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

float	The gravity value for **axisName**.

Description

Returns the gravity value for **axisName**.

Table of Contents

Example

```
// gets the gravity value of a previously defined axis
float hGravity = cInput.GetAxisGravity("Horizontal Movement");
```

GetAxisRaw

GetAxisRaw(description: *string***) :** *float*
GetAxisRaw(descriptionHash: *int***) :** *float*

Parameters

description	The name of the axis, as defined in SetAxis.
descriptionHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

float	A value between -1 and 1 inclusive.

Description

Returns the value of the virtual axis identified by **description** with no smoothing filtering applied. The value will be in the range -1...1 for keyboard and joystick input. Since input is not smoothed, keyboard input will always be either -1, 0 or 1. This is useful if you want to do all smoothing of keyboard input processing yourself.

Example

```
// move the transform horizontally with the "Horizontal Movement" axis
transform.Translate(cInput.GetAxisRaw("Horizontal Movement"), 0, 0);
```

GetAxisSensitivity

GetAxis(description: *string***) :** *float*
GetAxis(descriptionHash: *int***) :** *float*

Parameters

description	The name of the axis you want to retrieve the sensitivity value for.
descriptionHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

float	The sensitivity value for **axisName**.

Description

Returns the gravity value for **axisName**.

Example

```
// gets the sensitivity value of a previously defined axis
```

Table of Contents

```
float hSensitivity = cInput.GetAxisSensitivity("Horizontal Movement");
```

GetButton

Description

This works in exactly the same way as cInput.GetKey().

GetButtonDown

Description

This works in exactly the same way as cInput.GetKeyDown().

GetButtonUp

Description

This works in exactly the same way as cInput.GetKeyUp().

GetKey

GetKey(description: *string***) :** *boolean*
GetKey(descriptionHash: *int***) :** *boolean*

Parameters

description	The name of the key, as defined in SetKey.
descriptionHash	The hashcode of the name of the key, as defined in SetKey.

Returns

boolean	True if the key is being held down.

Description

Use this to determine if a key is being held down. GetKey returns true *repeatedly* while the user holds down the key, and returns false if the key is not being pressed. The use of GetKeyDown or GetKeyUp is recommended if you want to trigger an event only once per keypress, e.g., for jumping.

Example

```
// prints the message repeatedly, as long the player keeps pressing the "Shoot" input
if (cInput.GetKey("Shoot")) {
        Debug.Log("The player is shooting.");
}
```

Table of Contents

GetKeyDown

GetKeyDown(description: *string***) :** *boolean*
GetKeyDown(descriptionHash: *int***) :** *boolean*

Parameters

description	The name of the key, as defined in SetKey.
descriptionHash	The hashcode of the name of the key, as defined in SetKey.

Returns

boolean	True only once each time the key is first pressed down.

Description

Use this to determine if a key has been pressed. GetKeyDown returns true *only once* when the key is first pressed down. The use of GetKey is recommended if you want to trigger an event repeatedly while the key is being held down, e.g., for continuous movement.

Example

```
// prints the message just once when the player starts pressing the key
if (cInput.GetKeyDown("Jump") {
        Debug.Log("You pressed the jump button!");
}
```

GetKeyUp

GetKeyUp(description: *string***) :** *boolean*
GetKeyUp(descriptionHash: *int***) :** *boolean*

Parameters

description	The name of the key, as defined in SetKey.
descriptionHash	The hashcode of the name of the key, as defined in SetKey.

Returns

boolean	True only once each time the key is released.

Description

Use this to determine if a key has been released. GetKeyUp returns true *only once* when the key is first released. The use of GetKey is recommended if you want to trigger an event repeatedly while the key is being held down, e.g., for continuous movement.

Example

```
// prints the message just once when the player releases the key.
if (cInput.GetKeyUp("Jump") {
        Debug.Log("You released the jump button!");
```

Table of Contents

```
}
```

GetText

GetText(description: *string*, **[input:** *int*], **[returnBlank:** *bool*) **:** *string*
GetText(index: *int*, **[input:** *int*], **[returnBlank:** *bool*]) **:** *string*

Parameters

description	The name/description of the key you want to get the text for.
index	The index number of the key you want to get the text for.
input	Which input you want to get the text for.
returnBlank	Whether or not "None" should return an empty string. Defaults to **false**.

Description

Returns the text of the input used for the key. Note that **input** is optional and if omitted will default to 0 if you pass in the index or 1 if you pass in a string. Pass in 0 for **input** to get the name of the action. Pass in 1 or 2 for **input** to get the name of the primary or secondary inputs assigned to the key/axis. This is useful for displaying to the player which input is assigned to what key in a GUI. For more information on how to use this function in the creation of a GUI, see Making a Custom GUI. If no input is assigned to the action (as may often be the case with secondary inputs), GetText will return "None" by default, or it will return an empty string if **returnBlank** is true.

Example

```
cInput.SetKey("Shoot", Keys.LeftControl, Keys.RightControl); // index for this key is 0 in this example
Debug.Log(cInput.GetText("Shoot")); // displays "LeftControl"
Debug.Log(cInput.GetText("Shoot", 1); // also displays "LeftControl"
Debug.Log(cInput.GetText("Shoot", 2); // displays "RightControl"
Debug.Log(cInput.GetText("Shoot", 0)); // displays "Shoot"
Debug.Log(cInput.GetText(0)); // displays "Shoot"
Debug.Log(cInput.GetText(0, 0)); // also displays "Shoot"
Debug.Log(cInput.GetText(0, 1); // displays "LeftControl"
Debug.Log(cInput.GetText(0, 2); // displays "RightControl"
cInput.SetKey("Jump", Keys.Space); // Note that no secondary input is assigned to this key
Debug.Log(cInput.GetText("Jump", 1); // displays "Space"
Debug.Log(cInput.GetText("Jump", 2); // displays "None"
Debug.Log(cInput.GetText("Jump", 2, true); // displays "" (an empty string)
```

Init

Description

Use this to manually initialize the cInput object. Normally you won't need to use this since cInput will create the cInput object automatically, but there are some situations which might require you to manually call this method.

Table of Contents

Example

```
// manually create the cInput object
cInput.Init();
```

IsAxisDefined

IsAxisDefined(axisName: *string*) **:** *boolean*
IsAxisDefined(axisHash: *int*) **:** *boolean*

Parameters

axisName	The name of the axis, as defined in SetAxis.
axisHash	The hashcode of the name of the axis, as defined in SetAxis.

Returns

boolean	True if **axisName** or **axisHash** exists.

Description

Use this to determine if an axis exists by the name of **axisName** or the hashcode of **axisHash**. Note that you will probably never need to use this method unless you are a developer making a separate script/plugin and you want to make it compatible with cInput.

Example

```
// make sure user has defined this axis
if (cInput.IsAxisDefined("Horizontal")) {
        // do cInput stuff
} else {
        // fallback to non-cInput stuff?
}
```

IsKeyDefined

IsKeyDefined(keyName: *string*) **:** *boolean*
IsKeyDefined(keyHash: *int*) **:** *boolean*

Parameters

keyName	The name of the key, as defined in SetKey.
keyHash	The hashcode of the name of the key, as defined in SetKey.

Returns

boolean	True if **keyName** or **keyHash** exists.

Description

Use this to determine if a key exists by the name of **keyName** or the hashcode of **keyHash**. Note that

Table of Contents

you will probably never need to use this method unless you are a developer making a separate script/plugin and you want to make it compatible with cInput.

Example

```
// make sure user has defined this key
if (cInput.IsKeyDefined("Left")) {
        // do cInput stuff
} else {
        // fallback to non-cInput stuff?
}
```

LoadExternal

LoadExternal(externString: *string***) :** *void*

Parameters

externString A string containing all of the cInput settings.

Description

Use this to load cInput settings from some source other than PlayerPrefs. This is used in conjunction with externalInputs.

Example

```
// loads the cInput settings from an external text file
string external = System.IO.File.ReadAllText(Application.dataPath + "/settings.cInput");
cInput.LoadExternal(external);
```

RemoveModifier

RemoveModifier(modifierKey: *KeyCode***) :** *void*
RemoveModifier(modifier: *string***) :** *void*

Parameters

modifierKey They keycode of the key to stop using as a modifier.
modifier They string name of the key to stop using as a modifier.

Description

Removes **modifier** or **modifierKey** from being used as a modifier. Note that a modifier key cannot be used as a standalone input key. This function allows the key to be used again for normal inputs.

Example

```
// allows LeftShift to be used as a normal input key
cInput.RemoveModifier(KeyCode.LeftShift);
```

Table of Contents

ResetInputs

ResetInputs() : *void*

Description

Resets all controls back to the defaults as defined in SetKey.

Example

```
// reset the inputs back to default
cInput.ResetInputs();
```

SetAxis

SetAxis(name: *string*, **negative:** *string*, **positive:** *string*, **[sensitivity:** *float*], **[gravity:** *float*], **[deadzone:** *float*]) **:** *void*
SetAxis(name: *string*, **input:** *string*, **[sensitivity:** *float*], **[gravity:** *float*], **[deadzone:** *float*]) **:** *void*

Parameters

name	The description of what the axis is used for.
negative	The input which provides the negative value of the axis.
positive/**input**	The input which provides the positive value of the axis.
sensitivity	Optional. The sensitivity for this axis. Default is 3.
gravity	Optional. How fast the axis returns to 0 when input stops. Default is 3.
deadzone	Optional. How fast the axis returns to 0 when input stops. Default is 0.001.

Description

Creates an axis out of two inputs, which must be previously set with the SetKey function. The first input will be the negative axis, the second input the positive axis. Note that if only one input is passed in then the axis will only return positive values (unless inverted). You can optionally assign the sensitivity, gravity, or deadzone of the axis by passing in **sensitivity**, **gravity**, and **deadzone** respectively. Additionally, you can use SetAxisSensitivity, SetAxisGravity, and SetAxisDeadzone if you want to change these optional values for an axis after it has already been created.

Also note that setting up axes is not required and you should only do this if you require analog-like controls instead of digital controls. An axis input can be analog (e.g., gamepad axis) or digital (e.g., keyboard button) or even a combination of the two. A virtual analog axis will be created if necessary.

Example

```
// uses "Left" and "Right" inputs previously defined with SetKey to create a new axis
cInput.SetAxis("Horizontal Movement", "Left", "Right");
// or the same as above but with increased sensitivity
cInput.SetAxis("Horizontal Movement", "Left", "Right", 4.5f);
// uses only a single input as an axis (e.g., for a gas pedal in a driving game)
cInput.SetAxis("Acceleration", "Gas");
// or the same as above but with decreased sensitivity
```

Table of Contents

```
cInput.SetAxis("Acceleration", "Gas", 1.5f);
```

SetAxisDeadzone

SetAxisDeadzone(axisName: *string*, **deadzone:** *float*) **:** *void*
SetAxisDeadzone(axisHash: *int*, **deadzone:** *float*) **:** *void*

Parameters

axisName	The name of the axis you want to change deadzone for.
axisHash	The hashcode of the name of the axis you want to change deadzone for.
deadzone	The value to use for deadzone.

Description

Sets the deadzone for **axisName** to **deadzone**.

Example

```
// sets the deadzone for a previously defined axis
cInput.SetAxisDeadzone("Horizontal Movement", 0.1f);
```

SetAxisGravity

SetAxisGravity(axisName: *string*, **gravity:** *float*) **:** *void*
SetAxisGravity(axisHash: *int*, **gravity:** *float*) **:** *void*

Parameters

axisName	The name of the axis you want to change gravity for.
axisHash	The hashcode of the name of the axis you want to change gravity for.
gravity	The value to use for gravity.

Description

Sets the gravityof **axisName** to **gravity**.

Example

```
// sets the gravity of a previously defined axis
cInput.SetAxisGravity("Horizontal Movement", 0.3f);
```

SetAxisSensitivity

SetAxisSensitivity(axisName: *string*, **sensitivity:** *float*) **:** *void*
SetAxisSensitivity(axisHash: *int*, **sensitivity:** *float*) **:** *void*

Parameters

axisName	The name of the axis you want to change sensitivity for.
axisHash	The hashcode of the name of the axis you want to change sensitivity for.

Table of Contents

sensitivity	The value to use for sensitivity.

Description

Sets the sensitivity of **axisName** to **sensitivity**.

Example

```
// sets the sensitivity of a previously defined axis
cInput.SetAxisSensitivity("Horizontal Movement", 0.5f);
```

SetKey

SetKey(name: *string***, primary:** *string***, [secondary:** *string***], [primaryModifier:** *string***], [secondaryModifier:** *string***]) :** *void*

Parameters

name	The description of what the key is used for.
primary	The primary input to be used for this key.
secondary	Optional secondary input to be used for this key. Defaults to None if left blank.
primaryModifier	Optional modifier key to be used for the primary input.
secondaryModifier	Optional modifier key to be used for the secondary input.

Description

Defines the default primary input (keyboard button or gamepad/mouse axis) and optionally the secondary input to the keymap. Also optionally assigns modifier keys.

Example

```
// this creates a new key called "Left" and binds A and the Left Arrow to the key
cInput.SetKey("Left", "A", "LeftArrow");
// this uses Keys.D and Keys.RightArrow to accomplish something similar
cInput.SetKey("Right", Keys.D, Keys.RightArrow);
// creates a new key called "Pause" and binds Escape to the key
cInput.SetKey("Pause", Keys.Escape); // note that secondary input defaults to None
// creates a new key called "Target" with LeftControl as a modifier. This means Ctrl-T is the "Target" key.
cInput.SetKey("Target", Keys.T, Keys.None, Keys.LeftControl, Keys.None);
```

Table of Contents

Valid Inputs

Below is a list of all the acceptable strings you can use for SetKey and similar functions that accept a string. We recommend you use the Keys class, which has the benefit of intellisense/autocomplete to help you make sure you don't make any typos. For example, instead of using the string "KeypadPeriod" you would use Keys.KeypadPeriod.

Keyboard Inputs

Alpha0	F3	KeypadPeriod	Slash
Alpha1	F4	KeypadPlus	Space
Alpha2	F5	LeftAlt	SysReq
Alpha3	F6	LeftApple	Tab
Alpha4	F7	LeftArrow	Underscore
Alpha5	F8	LeftBracket	UpArrow
Alpha6	F9	LeftControl	A
Alpha7	F10	LeftParen	B
Alpha8	F11	LeftShift	C
Alpha9	F12	LeftWindows	D
AltGr	F13	Less	E
Ampersand	F14	Menu	F
Asterisk	F15	Minus	G
At	Greater	Numlock	H
BackQuote	Hash	PageDown	I
Backslash	Help	PageUp	J
Backspace	Home	Pause	K
Break	Insert	Period	L
CapsLock	Keypad0	Plus	M
Caret	Keypad1	Print	N
Clear	Keypad2	Question	O
Colon	Keypad3	Quote	P
Comma	Keypad4	Return	Q
Delete	Keypad5	RightAlt	R
Dollar	Keypad6	RightApple	S
DoubleQuote	Keypad7	RightArrow	T
DownArrow	Keypad8	RightBracket	U
End	Keypad9	RightControl	V
Equals	KeypadDivide	RightParen	W
Escape	KeypadEnter	RightShift	X
Exclaim	KeypadEquals	RightWindows	Y
F1	KeypadMinus	ScrollLock	Z
F2	KeypadMultiply	Semicolon	

Mouse Inputs

Table of Contents

Mouse0 Mouse1 Mouse2	Mouse3 Mouse4 Mouse5	Mouse6 Mouse Wheel Down Mouse Wheel Up	Mouse Down Mouse Left Mouse Right Mouse Up

Gamepad Inputs

JoystickButton0	Joystick2Button0	Joystick4Button0	Joy1 Axis 1-	Joy3 Axis 1-
JoystickButton1	Joystick2Button1	Joystick4Button1	Joy1 Axis 1+	Joy3 Axis 1+
JoystickButton2	Joystick2Button2	Joystick4Button2	Joy1 Axis 2-	Joy3 Axis 2-
JoystickButton3	Joystick2Button3	Joystick4Button3	Joy1 Axis 2+	Joy3 Axis 2+
JoystickButton4	Joystick2Button4	Joystick4Button4	Joy1 Axis 3-	Joy3 Axis 3-
JoystickButton5	Joystick2Button5	Joystick4Button5	Joy1 Axis 3+	Joy3 Axis 3+
JoystickButton6	Joystick2Button6	Joystick4Button6	Joy1 Axis 4-	Joy3 Axis 4-
JoystickButton7	Joystick2Button7	Joystick4Button7	Joy1 Axis 4+	Joy3 Axis 4+
JoystickButton8	Joystick2Button8	Joystick4Button8	Joy1 Axis 5-	Joy3 Axis 5-
JoystickButton9	Joystick2Button9	Joystick4Button9	Joy1 Axis 5+	Joy3 Axis 5+
JoystickButton10	Joystick2Button10	Joystick4Button10	Joy1 Axis 6-	Joy3 Axis 6-
JoystickButton11	Joystick2Button11	Joystick4Button11	Joy1 Axis 6+	Joy3 Axis 6+
JoystickButton12	Joystick2Button12	Joystick4Button12	Joy1 Axis 7-	Joy3 Axis 7-
JoystickButton13	Joystick2Button13	Joystick4Button13	Joy1 Axis 7+	Joy3 Axis 7+
JoystickButton14	Joystick2Button14	Joystick4Button14	Joy1 Axis 8-	Joy3 Axis 8-
JoystickButton15	Joystick2Button15	Joystick4Button15	Joy1 Axis 8+	Joy3 Axis 8+
JoystickButton16	Joystick2Button16	Joystick4Button16	Joy1 Axis 9-	Joy3 Axis 9-
JoystickButton17	Joystick2Button17	Joystick4Button17	Joy1 Axis 9+	Joy3 Axis 9+
JoystickButton18	Joystick2Button18	Joystick4Button18	Joy1 Axis 10-	Joy3 Axis 10-
JoystickButton19	Joystick2Button19	Joystick4Button19	Joy1 Axis 10+	Joy3 Axis 10+
Joystick1Button0	Joystick3Button0		Joy2 Axis 1-	Joy4 Axis 1-
Joystick1Button1	Joystick3Button1		Joy2 Axis 1+	Joy4 Axis 1+
Joystick1Button2	Joystick3Button2		Joy2 Axis 2-	Joy4 Axis 2-
Joystick1Button3	Joystick3Button3		Joy2 Axis 2+	Joy4 Axis 2+
Joystick1Button4	Joystick3Button4		Joy2 Axis 3-	Joy4 Axis 3-
Joystick1Button5	Joystick3Button5		Joy2 Axis 3+	Joy4 Axis 3+
Joystick1Button6	Joystick3Button6		Joy2 Axis 4-	Joy4 Axis 4-
Joystick1Button7	Joystick3Button7		Joy2 Axis 4+	Joy4 Axis 4+
Joystick1Button8	Joystick3Button8		Joy2 Axis 5-	Joy4 Axis 5-
Joystick1Button9	Joystick3Button9		Joy2 Axis 5+	Joy4 Axis 5+
Joystick1Button10	Joystick3Button10		Joy2 Axis 6-	Joy4 Axis 6-
Joystick1Button11	Joystick3Button11		Joy2 Axis 6+	Joy4 Axis 6+
Joystick1Button12	Joystick3Button12		Joy2 Axis 7-	Joy4 Axis 7-
Joystick1Button13	Joystick3Button13		Joy2 Axis 7+	Joy4 Axis 7+
Joystick1Button14	Joystick3Button14		Joy2 Axis 8-	Joy4 Axis 8-
Joystick1Button15	Joystick3Button15		Joy2 Axis 8+	Joy4 Axis 8+
Joystick1Button16	Joystick3Button16		Joy2 Axis 9-	Joy4 Axis 9-
Joystick1Button17	Joystick3Button17		Joy2 Axis 9+	Joy4 Axis 9+
Joystick1Button18	Joystick3Button18		Joy2 Axis 10-	Joy4 Axis 10-
Joystick1Button19	Joystick3Button19		Joy2 Axis 10+	Joy4 Axis 10+

Table of Contents