

ContextActionService basics

Used for binding to inputs gracefully. If an input is bound to one action and another action is to use the **same** button, the **BindAction** and **UnbindAction** functions will handle the collision properly. Example: **(A)** could make the player jump OR open a door but only when close enough.

Each input type bound to a function using BindAction works like a **stack**: whichever function was the **most recent** to be bound will be called when that input type is activated by the player.

ContextActionService binding

`:BindAction(name, func, touchButton, inputTypes...)`

Calls `func` with `name`, `InputState`, and `InputObject` when `inputTypes` are used.

`:UnbindAction(name)`

Unbinds a function from action `name`.

Input types for `:BindAction()`:

`UserInputType`, `KeyCode`, and/or `PlayerActions`

As soon as a player can use an input (like A/B, triggers, or thumbstick), use `:BindAction(...)`. This will override the given input types current actions with the new one. When the player can no longer use the input for the action, use `:UnbindAction(name)`.

ContextActionService example

```
local cas = game:GetService("ContextActionService")

function handleAction(actionName, inputState, inputObject)
    if actionName == "Swing sword" and inputState == Enum.UserInputState.Button then
        print("Swinging sword")
    end
end

-- When the sword is equipped:
cas:BindAction("Swing sword", handleAction, false, Enum.KeyCode.ButtonA)
-- When the sword is unequipped:
cas:UnbindAction("Swing sword")
```

When the given input type is activated/changed, the function passed to `:BindAction(...)` is called with the action name, the **input state** (**Begin**, **Change**, **End** or **Cancel**) and the **InputObject**. It's good practice to have just one action-handling function per script.

Wiki References

Wiki References (cont)

[UserInputState](#)

[PlayerActions](#)

Gamepad KeyCodes and UserInputTypes



Buttons use the `UserInputStates` `Began/End`. Triggers and thumbsticks use `Change`.

UserInputService querying

Property (boolean): `GamepadEnabled`

Returns **true** if at least one gamepad is connected.

Event: `GamepadConnected`

Fired when a gamepad is available.

Event: `GamepadDisconnected`

Fired when a gamepad is no longer available.

Avoid using `UserInputService.InputBegan` for actions involving button presses! Use `ContextActionService.BindAction(...)` because this "overrides" existing actions using the given input types, and properly returns control to existing actions when yours are unbound.

Tips and Good Practices

A Button - `Enum.KeyCode.ButtonA`

Bound to **jump** by default. Should be used as **accept** button for prompts.

B Button - `Enum.KeyCode.ButtonB`

Should be used as **back** or **cancel** button for menus/prompts.

Right Trigger - `Enum.KeyCode.ButtonR2`

Use for **primary** character actions.

Left Trigger - `Enum.KeyCode.ButtonL2`

Use for **secondary** character actions.

Right Thumbstick - `Enum.KeyCode.Thumbstick1`

Use for **camera** movement.

Left Thumbstick - `Enum.KeyCode.Thumbstick2`

Use for **character** movement.

Right/Left Bumpers - `Enum.KeyCode.ButtonR1/ButtonL1`

Bound to **switch tools** by default.

A good way to know what kinds of control schemes work is by

Guides and Tutorials

[Gamepad input](#)

Game Services

[ContextActionService](#)

[UserInputService](#)

Enum Types

[KeyCode](#)

[UserInputType](#)

playing other gamepad/controller enabled games.



By **Ozzypig** (Ozzypig)
cheatography.com/ozzypig/
ozzypig.com

Published 28th January, 2016.
Last updated 24th January, 2017.
Page 1 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>