

The Controller Manager, Input Component, and You

TLDR

These are the functions most important to you.

```
InputGetMoveUp()  
InputGetMoveRight()  
InputIsMoving()  
  
InputGetInteractPressed()  
InputGetInteractReleased()  
InputGetInteractTriggered()  
  
InputGetAttackPressed()  
InputGetAttackReleased()  
InputGetAttackTriggered()
```

TLDR	1
The Controller Manager	2
ControllerState	2
ControllerManagerRefresh	3
ControllerManagerStateUpdate	4
The Input Component	5
Input Get Functions	5
Keybinds	6

The Controller Manager

The controller manager surprisingly manages the xinput controllers connected to the game.

It consists of:

Structures

- ControllerState

Functions

```
ControllerManagerRefresh()  
ControllerManagerStateUpdate()  
  
ControllerManagerGetInput()  
ControllerManagerGetButtonsReleased()  
ControllerManagerGetButtonsTriggered()
```

We'll focus on the ControllerState structure, the Refresh function, and the StateUpdate function.

ControllerState

```
typedef struct ControllerState  
{  
    // Whether or not the controller port is connected  
    bool controllerConnected;  
  
    // The xinput state structure for this controller  
    XINPUT_STATE controllerState;  
  
    // The buttons that were released this frame  
    WORD buttonsReleased;  
  
    // The buttons that were triggered this frame  
    WORD buttonsTriggered;  
  
} ControllerState;
```

The Controller Manager, Input Component, and You

The `ControllerState` structure represents the variables that pertain to a controller connected (or not connected) to the game. Within the `ControllerManager.c` file there exists an array of 4 `ControllerState` structures, each belonging to the 4 controllers that could potentially be connected to the game. The array indices correspond to the ports the controllers are connected to. Index 0 represents the first controller connected to the computer.


```
static ControllerState controllerStates[MAX_CONTROLLERS] = { 0 };
```

The **`controllerConnected`** boolean variable is only set to true if the controller is connected to the game. This boolean is set in the `ControllerManagerRefresh()` function. This allows the `ControllerManagerStateUpdate()` function to only focus on querying the state of controllers that are connected to the system.

The **`controllerState`** variable holds the current `XINPUT_STATE` structure for the given controller.

From <https://docs.microsoft.com/en-us/windows/desktop/xinput/structures>

```
typedef struct _XINPUT_STATE {  
    DWORD      dwPacketNumber;  
    XINPUT_GAMEPAD Gamepad;  
} XINPUT_STATE, *PXINPUT_STATE;  
  
typedef struct _XINPUT_GAMEPAD {  
    WORD  wButtons;  
    BYTE  bLeftTrigger;  
    BYTE  bRightTrigger;  
    SHORT sThumbLX;  
    SHORT sThumbLY;  
    SHORT sThumbRX;  
    SHORT sThumbRY;  
} XINPUT_GAMEPAD, *PXINPUT_GAMEPAD;
```



The **`buttonsReleased`** and **`buttonsTriggered`** variables are of type `WORD`, which is an alias for an unsigned short. They contain a number comparable to the **`wButtons`** variable in the `XINPUT_GAMEPAD` structure. The bits that are set in these variables represent the buttons from the xinput controller that were released / triggered respectively during the current frame that the controller state was updated. These two variables can be accessed with the `ControllerManagerGetButtonsReleased()` and `ControllerManagerGetButtonsTriggered()` functions.

ControllerManagerRefresh

This function's only purpose is to check which controller ports are active and which are not. This function attempts to query the xinput state from all 4 controller ports. If there is no controller connected to a given port, the **`controllerConnected`** boolean variable is set to false. If it is, it will be set to true.

The Controller Manager, Input Component, and You

This function should **NOT** be called every frame. In fact at the time of writing this, it is only called when a new level loads. However, ideally this function should be called at an interval (maybe every 5 seconds?) to account for technical difficulties where a controller possibly gets disconnected and then reconnected.

ControllerManagerStateUpdate

This function updates the state of the controllers currently connected to the system. After this update, the current input values from each controller can be retrieved from game code.

Steps to updating the controller state:

1. Loop through the 4 controllers
 - a. If the controller port is inactive, skip to the next controller
2. Get the state of the controller, and if it is correctly connected continue
3. Reset the released and triggered button variables (to ensure the relevant bits are only active for one frame)
4. If the controller state packet numbers are different (indicating that the controller state has changed), then record the new state
 - a. Record the buttons state for the previous and current frame.
 - b. Bitwise-XOR the previous and current state to identify which buttons **changed** between the previous and current frame
 - c. Bitwise-AND the previous state and the buttons that changed this frame to identify which buttons were **released** this frame
 - d. Bitwise-AND the current state and the buttons that changed this frame to identify which buttons were **triggered** this frame
 - e. Finally, save the current controller state into the ControllerState structure for this controller.

The Input Component

The input component is another component that can be attached to a game object. When it is created and attached to an object, it is given an input source that indicates where its input should be coming from, whether it be from 1 of the 4 controller ports, or potentially an AI.

```
InputGetMoveUp()  
InputGetMoveRight()  
InputIsMoving()  
  
InputGetInteractPressed()  
InputGetInteractReleased()  
InputGetInteractTriggered()  
  
InputGetAttackPressed()  
InputGetAttackReleased()  
InputGetAttackTriggered()
```

Input Get Functions

These are the most important functions for the user (YOU!) to know about. These functions were modelled after the AEInputCheck functions.

InputGetMoveUp() and **InputGetMoveRight()** will give you the movement input in a range from -1 to 1.

InputIsMoving() will allow you to check if ANY movement is being input.

The **InputGetInteract** functions will return the status of the “Interact” input keys. For player 1, this might be the ‘E’ key on the keyboard or the ‘X’ button on the controller.

The **InputGetAttack** functions will return the status of the “Attack” input keys. For player 1, this might be the ‘Q’ key on the keyboard or the ‘A’ button on the controller.

The Controller Manager, Input Component, and You

Keybinds

All of the keys/buttons for these inputs can be modified using the defines at the top of the Input.c file.

```
// MoveUp keys
#define MOVE_UP_P0 'W'
#define MOVE_DOWN_P0 'S'
#define MOVE_UP_P1 'Y'
#define MOVE_DOWN_P1 'H'
#define MOVE_UP_P2 'P'
#define MOVE_DOWN_P2 ';'
#define MOVE_UP_P3 VK_UP
#define MOVE_DOWN_P3 VK_DOWN

// MoveRight keys
#define MOVE_RIGHT_P0 'D'
#define MOVE_LEFT_P0 'A'
#define MOVE_RIGHT_P1 'J'
#define MOVE_LEFT_P1 'G'
#define MOVE_RIGHT_P2 '\\'
#define MOVE_LEFT_P2 'L'
#define MOVE_RIGHT_P3 VK_RIGHT
#define MOVE_LEFT_P3 VK_LEFT

// Interact keys
#define INTERACT_P0 'E'
#define INTERACT_P1 'U'
#define INTERACT_P2 '['
#define INTERACT_P3 VK_RSHIFT
#define INTERACT_CONTROLLER XINPUT_GAMEPAD_X

// Attack keys
#define ATTACK_P0 'Q'
#define ATTACK_P1 'T'
#define ATTACK_P2 'O'
#define ATTACK_P3 '/'
#define ATTACK_CONTROLLER XINPUT_GAMEPAD_A
```