# Guide to Customizing controls.ini

This document explains how to configure your input devices by editing the controls.ini file. This file gives you full control over how your keyboard, mouse, joysticks, and gamepads are mapped to in-game actions.

## 1. Generating the Default File

Before you start, it's highly recommended to generate a fresh, default controls.ini file. This gives you a complete template with all possible actions and recommended starting points for your controls.
To generate the file, run the application from your terminal with the --create flag, followed by the argument controls.
Basic Command:
This will create a file named controls.ini in the current directory.
./lindbergh --create controls

Specifying a Path:
You can also specify a path to create the file in a different location. The folder must already exist.

- **Create in a specific folder:**
  ./lindbergh --create controls ./my/folder/

- **Create with a custom name and location:**
  ./lindbergh --create controls ./my/path/to/mycontrols.ini

## 2. Structure of the INI File

The file is organized into sections, each marked with a name in square brackets (e.g., [Config]). Under each section are key-value pairs, separated by an equals sign (=).
[SectionName]
Key = Value1, Value2, ...

- **Section**: A category of settings (e.g., [Driving], [Shooting]). The game automatically loads the [Common] section plus one other section that matches the game's type.
- **Key**: The name of the logical in-game action you want to map (e.g., P1_Start, Steer, Trigger).
- **Value**: One or more physical input strings, separated by commas, that will trigger the action.

## 3. Configuration Sections

### [Config]

This section contains global settings that affect all input devices.

- **<ActionName>_DeadZone**: Sets a custom analog deadzone for a specific logical action. This gives you precise control over actions like steering and pedals. The value is from 0 (no deadzone) to 32767. If not specified, a sensible default is used (a large deadzone for centering sticks, a small one for non-centering pedals/triggers).
    - **Example**: Steer_DeadZone = 7000
    - **Example for Player 2**: P2_Gas_DeadZone = 500
- **ShakeIncreaseRate / ShakeDecayRate**: Controls the gun shaking effect in games like *House of the Dead 4*.

### [ControllerGUIDs]

This section is managed automatically by the application to ensure your controllers are always assigned to the correct player.

- P<N>_GUID: Stores the unique hardware ID for the controller assigned to Player <N>. You should not need to edit this section manually.
- **Swapping Players**: If you have multiple controllers and want to change which one is Player 1 vs. Player 2, you can do so by swapping the GUID strings between the P1_GUID and P2_GUID keys.
- **Resetting Assignments**: If you want to reset all controller assignments, you can delete this entire section. The application will regenerate it on the next launch based on the order the controllers are detected.

### Game Type Sections

These sections contain the actual control mappings. The application will load [Common] and one other section based on the game being played.

- [Common]: For actions shared by all games (Test, Coin, Start, Service).
- [Digital]: For standard arcade games with an 8-way joystick and buttons.
- [Driving]: For racing games with analog steering, gas, and brake.
- [Shooting]: For lightgun games.
- [ABC]: For games with flight-stick style controls like *After Burner Climax*.
- [Mahjong]: For mahjong panel controls.

# 4. Understanding Controller Mappings (gamecontrollerdb.txt)

For the application to recognize your gamepad (like an Xbox or PlayStation controller) and allow you to use the simple GC bindings (e.g., GC0_BUTTON_A), it needs a database of controller mappings.

- **Built-in Mappings**: The application already includes a large, built-in database of mappings for hundreds of common controllers, so many devices will work out of the

box.

- **Adding Custom Mappings**: If you have a rare controller or want to override the built-in behavior, you can provide an external mapping file. The file can have any name (though gamecontrollerdb.txt is standard) but must follow the SDL mapping format. You can load a custom file by running the application with the --controllerdb flag: ./lindbergh --controllerdb ./path/to/your_mappings.txt

  The mappings from your file will be added to the existing built-in database.
- **Where to get mapping files**: You can find the community-sourced gamecontrollerdb.txt file, which is a great starting point for custom mappings, at the official SDL GameControllerDB GitHub repository.

# 5. Binding String Syntax

The value for each key is a comma-separated list of physical input strings. Here is the format for each device type:

## Keyboard

- **Format**: KEY_<ScanCodeName>
- **Example**: KEY_A, KEY_Space, KEY_ArrowUp
- **Note**: The name is the SDL Scancode Name, not necessarily the character on the key.

## Mouse

- **Buttons**: MOUSE_LEFT_BUTTON, MOUSE_RIGHT_BUTTON, MOUSE_MIDDLE_BUTTON
- **Axes**: MOUSE_AXIS_X, MOUSE_AXIS_Y

## Generic Joysticks (JOY) and Standard Gamepads (GC)

- **Player Index <N>**: The number <N> after JOY or GC **always refers to the player**. The application uses the [ControllerGUIDs] section to determine which physical device is assigned to which player. JOY0_ and GC0_ will always be the controller assigned to **Player 1**, JOY1_ and GC1_ will be for **Player 2**, and so on.
- **Buttons**:
  - JOY<N>_BUTTON_<ButtonNumber> (e.g., JOY0_BUTTON_0)
  - GC<N>_BUTTON_<ButtonName> (e.g., GC0_BUTTON_A, GC0_BUTTON_LEFTSHOULDER)
- **Hats/D-Pads**: JOY<N>_HAT<HatNumber>_<Direction> (e.g., JOY0_HAT0_UP)
- **Axes**:
  - JOY<N>_AXIS_<AxisNumber> (e.g., JOY0_AXIS_0)
  - GC<N>_AXIS_<AxisName> (e.g., GC0_AXIS_LEFTX, GC0_AXIS_TRIGGERRIGHT)

## Axis Suffixes (for JOY and GC axes)

You can add suffixes to an axis string to change its behavior:

- _POSITIVE / _NEGATIVE: Treats one direction of an analog axis like a digital button press.

    - **Example**: P1_Right = GC0_AXIS_LEFTX_POSITIVE
- _POSITIVE_HALF / _NEGATIVE_HALF: Used to map two different triggers or pedals to a single combined analog action (like a throttle).
    - **Example**: Throttle = JOY0_AXIS_5_POSITIVE_HALF, JOY0_AXIS_2_NEGATIVE_HALF
- _INVERTED: Reverses the direction of an analog axis.
    - **Example**: Throttle = GC0_AXIS_RIGHTY_INVERTED

## Example

Here is a complex mapping for Player 1's "Start" button:
P1_Start = KEY_1, GC0_BUTTON_START, JOY0_BUTTON_9

This means the action will be triggered by pressing the '1' key on the keyboard, the 'Start' button on Player 1's gamepad, OR button number 9 on Player 1's generic joystick.