

Requirements - Mac Mouse Controller

Application Description

The application, after the user has granted it permissions, must be able to intercept mouse events (left/right click/alternative keys, scroll, drag & drop, movement, etc.) and give the user the ability to change the standard behavior of each key or leave the existing behavior but modify its execution (accelerate scrolling, etc.). Finally, the user must also be able to create combinations between keys and mouse buttons (`ctrl + scroll = scroll acceleration`, `opt + scroll = precision scrolling`, etc.).

REQ-001: Initializing and Managing Accessibility Permissions

Description

When the application starts, the system must immediately check whether it has **Accessibility** privileges (`Accessibility API`). These permissions are mandatory for using the `CGEventTap` (Core Graphics Event Taps), which are needed to intercept and modify mouse input. The expected flow is as follows:

1. The application starts.
2. The system checks the permissions status via the system APIs (`AXIsProcessTrustedWithOptions`).
3. **If permissions are missing:**
 - The application displays a "Lock" screen (permission request UI) that prevents access to other features.
 - The operating system is prompted to display the native macOS prompt ("*This application would like to control the computer...").
 - The application starts a background *polling* process to detect changes in the permissions status without requiring the app to be restarted.

4. If permissions are granted:

- The interface automatically updates to show the "Active" status.
- The interception engine (`MouseHookService`) is instantiated and started.

Input

- **User:**
 - Launch the executable.
 - Click the "Open System Settings" button (in the app UI).
 - Interact with the "ON/OFF" toggle in the macOS *Privacy & Security* → *Accessibility* panel.
- **System:**
 - Boolean state returned by the `AXIsProcessTrusted` function.
 - Application window focus/activation event (trigger to recheck permissions).

Output

- **User Interface (UI):**
 - **Denied State:** Display a clear warning (e.g., red/yellow icon) with text instructions on how to enable permissions.
 - **Granted State:** Immediate transition to the main or configuration screen (e.g., green icon), enabling mapping controls.
- **Operating System:**
 - Automatically open the "System Settings" window to the correct page (via URL scheme `x-apple.systempreferences:...`) if the user presses the dedicated button.
- **Backend:**
 - Console log confirming when the `MouseHookService` is started or stopped.

Acceptance and Validation Constraints

- **Security Constraint:** The application must never attempt to bypass the permissions lock or fire the Tap Event if `AXIsProcessTrusted` returns `false`.

Doing so would cause an immediate crash or undefined behavior.

- **UX Constraint:** The user must not be forced to restart the application after granting permissions. The application must "notice" this on its own (within 2-3 seconds max).
- **Validation:** The interception engine (`MouseHookService`) can be initialized (`.start()`) only if the `hasPermissions` state variable is true.

What If (Exception Handling and Unexpected Behavior)

"What If" Scenario	Expected System Behavior
The user denies permission in the system prompt.	The application remains on the "Lock" screen. Polling continues to run. A "Retry" or "Open Settings" button is offered to allow the user to remedy manually.
Permissions are revoked while the app is running.	The <i>polling</i> or event listener detects the loss of permissions. The application must: <ol style="list-style-type: none">1. Immediately stop the <code>MouseHookService</code> (to avoid crashes).2. Return the user interface to the "Lock/Permission Request" state.
The application is updated (new build).	macOS often resets permissions for unsigned or modified binaries. The app must treat this scenario exactly like a "First Launch" (see point 1 Description), requesting authorization again without assuming it persisted.
The control Timer (Polling) freezes or fails.	As a fallback mechanism, the application must perform a permission check also every time the app window returns to the foreground (<code>onAppear</code> or <code>scenePhase.active</code>).

REQ-002: Modular Event Processing Architecture

Description

To ensure scalability and maintainability, the application must process input events using a **Pipeline Architecture**-based on the *Chain of Responsibility* design pattern. The core interception engine (`MouseHookService`)

acts as a dispatcher. It maintains an ordered list of active **Event Handlers**.

When a hardware event is intercepted:

1. The event is passed to the first Handler in the chain.
2. The Handler analyzes the event type and properties.
3. The Handler returns one of three possible outcomes:
 - **Pass-through:** The event is ignored and passed unmodified to the next Handler.
 - **Modification:** The event properties are altered (e.g., scroll delta changed), and the modified event is passed to the next Handler.
 - **Consumption:** The Handler executes a specific action (e.g., triggers a shortcut) and returns `null`. The propagation stops immediately; the event is suppressed and never reaches the Operating System. This architecture allows new features (e.g., "Smooth Scroll", "Zoom Gesture") to be added as isolated modules without modifying the core engine logic.

Input

- **Hardware Input:** Raw `CGEvent` objects originating from the `Quartz Event Tap` (mouse clicks, movement, scroll wheel deltas).
- **Configuration:** An ordered list of active Handlers (determined by user settings).
- **State:** Contextual data required by specific handlers (e.g., current velocity for inertia calculations).

Output

- **System Event Stream:**
 - The final `CGEvent` is posted to the active application or window server.
 - OR: No event is posted (if consumed).
- **Side Effects:** Synthetic events generated by Handlers (e.g., keyboard key-press simulation, virtual trackpad gestures).

Acceptance Constraints & Validation

- **Performance Constraint (Critical):** The entire chain of handlers must process a single event in **under 10 milliseconds**. Exceeding this threshold will cause the operating system to disable the Event Tap automatically (to prevent system-wide input lag).
- **Fail-Safe:** If a Handler encounters an internal error, it must default to "Pass-through" mode to ensure the mouse remains usable.
- **Modularity:** A specific feature (e.g., Back Button) must be completely contained within its own class/struct. Disabling a feature in the settings must result in the complete removal of its Handler from the processing chain.

What If (Exception Handling)

What If Scenario	Expected System Behavior
A Handler "consumes" an event.	The processing chain terminates immediately. Subsequent handlers in the list do not receive the event. The OS does not receive the original hardware input (e.g., the physical "Button 4" click is blocked).
Two Handlers conflict (e.g., both listen for "Button 3").	Priority is determined by the order of registration in the chain. The first Handler in the list gets priority to consume or modify the event. The architectural design must enforce a strict initialization order.
An infinite loop occurs within a Handler.	The macOS watchdog for <code>CGEventTap</code> will time out the tap. The application must detect this disablement (via <code>kCGEventTapDisabledByTimeout</code> callback) and attempt to restart the engine automatically.
System is under heavy load.	Handlers must avoid heavy computations (e.g., network calls, disk I/O) on the main event thread. Complex logic must be dispatched asynchronously, although the event decision (Pass/Consume) must remain synchronous.

REQ-003: User Configuration & Input Mapping Strategy

Description

The application must provide a mechanism for users to define custom behaviors for hardware input events. This system relies on a "**Trigger-Action**" model, where a specific hardware state triggers a defined software response.

The configuration engine must support three distinct types of customization:

1. **Direct Remapping (Substitution):** Replacing a standard mouse event with a keyboard shortcut or system command (e.g., *Button 4* → `CMD + C` or *Mission Control*).
2. **Behavior Modification (Transformation):** Intercepting an event and altering its properties before releasing it to the OS. This includes:
 - **Scroll Acceleration:** applying a multiplier to the `deltaY` of scroll events based on velocity.
 - **Precision Scrolling:** reducing the `deltaY` for fine-grained control.
 - **Inversion:** flipping the axis of movement or scroll.
3. **Modifier-Based Combinations:** The system must distinguish between a raw mouse event and a mouse event occurring while keyboard modifiers (`Shift`, `Ctrl`, `Opt`, `Cmd`) are held down. This allows for "layered" controls (e.g., `Scroll` = Standard, `Ctrl + Scroll` = Zoom).

The configuration must be persistent, editable via a GUI, and dynamically reloadable by the `MouseHookService` (REQ-002) without restarting the application.

Input

- **User Configuration (GUI):**
 - Selection of the **Trigger Event** (Left/Right/Middle/Other Click, Scroll Up/Down, Drag Start).
 - Selection of **Required Modifiers** (Checkbox set: `^`, `⌥`, `⇧`, `⌘`).
 - Definition of the **Target Action:**
 - *Type:* Keystroke, System Function, or Property Modifier.
 - *Parameters:* (e.g., "Sensitivity: 1.5x", "Key: 'F'").
- **Hardware Event Stream:**
 - The live stream of mouse events plus the current state of the global keyboard modifier flags (`CGEventSourceFlagsState`).

Output

- **Configuration Schema (JSON/Plist):** A structured data model saved to disk representing the mapping profile. Example structure:

```
{  
  "trigger": "scrollWheel",  
  "modifiers": ["control"],  
  "actionType": "propertyModification",  
  "parameters": { "accelerationFactor": 2.5 }  
}
```

- **Runtime Lookup Table:** An optimized in-memory dictionary used by the `MouseHookService` to instantly find the action associated with the current `(EventType + ModifierState)`.

Acceptance Constraints & Validation

- **Specificity Priority:** If a conflict exists, the most specific rule must win.
 - *Example:* If the user maps `Scroll` to "Page Down" and `Cmd + Scroll` to "Zoom", holding Cmd must trigger "Zoom", not both.
- **Safety Lock:** The application must prevent (or warn against) remapping the primary **Left Click** without a modifier, as this renders the computer difficult to use. A "Panic Reset" (e.g., holding `Esc` for 5 seconds) should be implemented to restore default bindings.
- **Latency Budget:** The lookup of a mapping rule must happen in O(1) or very fast O(n) time to adhere to the 10ms requirement defined in REQ-002.
- **Drag & Drop Integrity:** Remapping "Left Mouse Down" must not break the system's ability to detect a "Drag". The system must distinguish between a "Click" (Down + Up) and a "Hold/Drag" (Down + Move).

What If (Exception Handling)

What If Scenario	Expected System Behavior
User maps <code>Ctrl + Scroll</code> to acceleration, but simply Scrolls.	The system checks the Modifier Mask. If <code>Ctrl</code> is not pressed, it falls back to the default handler (or the generic <code>Scroll</code> handler if one exists).

User connects a new mouse with different button IDs.	The configuration is hardware-agnostic by default (Button 3 is Button 3 on any mouse). Optionally, the system may store profiles linked to <code>VendorID/ProductID</code> to allow per-device settings.
A "Combo" action conflicts with a System Shortcut.	The application intercepts the event <i>before</i> the OS. If the user maps <code>Cmd + Right Click</code> , the OS context menu (if triggered by that combo) should ideally be suppressed if the handler returns <code>consumption</code> .
User sets an invalid value (e.g., scroll speed 0).	Validation logic in the GUI input fields must reject values that would render the input unusable (negative scroll speeds or zero multipliers).