### README: Stream Deck Macro Scripts

#### Overview

This document describes the functionality of batch and shell scripts designed to be used with Elgato Stream Deck devices. These scripts automate tasks such as launching applications or executing commands on systems where the Stream Deck is connected. The scripts are stored on a USB drive to facilitate portability and ease of use across multiple systems. This is due to the inability of Stream deck to store non-volatile memory on the unit itself, thereby requiring either local machine presence of these scripts, intranet presence, or removable flash media presence (the least invasive of these three options for the moment.)

#### Script Descriptions

1. **Batch Scripts (.bat)**
   1. **Purpose:** Designed to run on Windows operating systems, these scripts execute predefined Windows commands or launch applications.
   2. **Location:** Stored on a USB drive designated as Drive D: in the folder Stream deck Scripts Nov 2024 -> WINDOWS MACROS. The way these were tested originally was on my own device, at home on local intranet without outside internet connection. These have not been tested yet on any RTC intranet devices for compliance reasons.
2. **Shell Scripts (.sh)**
   1. **Purpose:** Intended for use on Unix-like systems, such as Linux or macOS, to perform similar functions as their batch counterparts, including opening applications or running system commands. I have written this to be tested on a Raspberry Pi running on Raspbian/Debian stock OS which were tested on RTC Raspberry Pis that were likewise not online to outside or local intranet
   2. **Location:** Co-located with the batch scripts for ease of access and consistency.

### Detailed Macro Descriptions with System Components Affected

#### Macro 1: LaunchApp.bat

* **Functionality:** Launches a specific application.
* **Command Used:** start "" "path\to\application.exe"
* **System Components Affected:** Primarily affects the software's install directory by invoking the executable. Does not interact with system settings or MSC.
* **Security Rationale:** Executes only pre-approved applications from secure locations to mitigate risks associated with running unauthorized software.
* **Control Measures:** File permissions restrict modifications, and execution paths are logged for audit purposes.

#### Macro 2: OpenDirectory.sh

* **Functionality:** Opens a predefined directory.
* **Command Used:** xdg-open /path/to/directory
* **System Components Affected:** Accesses file system directories but does not alter any system settings or configurations.
* **Security Rationale:** Ensures that only non-sensitive directories are accessed to prevent data exposure.
* **Control Measures:** Directories are whitelisted, and script permissions are managed to prevent unauthorized changes.

#### Macro 3: ShutdownPC.bat

* **Functionality:** Shuts down the Windows operating system safely.
* **Command Used:** shutdown /s /t 0
* **System Components Affected:** Interacts directly with the Windows operating system's power management settings through the shutdown MSC command.
* **Security Rationale:** The use of a direct shutdown command prevents misuse of system shutdowns and ensures that the command can be audited and traced.
* **Control Measures:** Execution is restricted to users with administrative privileges, and logs are maintained for all shutdown events.

#### Macro 4: RestartService.sh

* **Functionality:** Restarts a specific service.
* **Command Used:** sudo systemctl restart serviceName
* **System Components Affected:** Affects system services managed through services.msc or the Linux equivalent system management daemon.
* **Security Rationale:** Controls the service lifecycle securely by leveraging native system tools with built-in security features.
* **Control Measures:** Only designated services are targeted, with script access restricted to system administrators and actions logged for auditability.

### Ensuring Minimal Invasiveness and Security

* **Command Simplicity and Specificity:** Each script uses the most straightforward command necessary to accomplish its purpose, specifically targeting individual components without unnecessary access to broader system resources.
* **Hardcoded Paths and Services:** Ensuring that paths and services are hardcoded and pre-approved mitigates risks associated with script manipulation, providing a clear audit trail.

### Compliance Verification and Audits

* **MSC Snap-ins and Directory Access:** Detailed records are kept regarding which MSC snap-ins (like services.msc for service management or system tools for directory access) and directories are accessed or modified by the macros. This information is crucial for compliance audits to verify that all operations adhere to corporate policies and IT security standards.
* **Regular Security Reviews:** Each macro is periodically reviewed to ensure it remains compliant with the latest security policies and that any changes to system directories or MSC configurations are properly authorized and documented.

#### Security and Compliance

* **Code Signing:** These macros are currently untested and not suitable for digital signing until thorough instutional tests have been completed. I am fully expecting there may be requirements to fix or alter these scripts before or even if these can be signed. In the eventuality these are used on campus, it’s common IT policy with most public and private institutions that any scripts used internally must be signed with an approved digital signature to verify the integrity and origin of the scripts. This prevents unauthorized modifications which could lead to security vulnerabilities.
* **Access Controls:** Scripts should be stored on encrypted USB drives with access limited to authorized personnel only, using both physical and digital access controls.
* **Audit Trails:** Enable logging mechanisms to record when scripts are executed, including timestamp, user ID, and outcome of the script execution to maintain a clear audit trail.

#### Testing Procedures

1. **Functional Testing:**
   1. **Objective:** Ensure each script executes as intended on all target systems.
   2. **Method:** Manually trigger each script via the Stream Deck interface on a test system configured similarly to the production environment.
   3. **Expected Outcome:** Each script performs the actions for which it was designed without errors.
2. **Security Testing:**
   1. **Objective:** Identify any potential security vulnerabilities introduced by the scripts.
   2. **Method:** Conduct a static code analysis to inspect the scripts for common vulnerabilities (e.g., injection flaws, unhandled errors). Perform dynamic testing to monitor the script behavior under normal and malicious conditions.
   3. **Expected Outcome:** Scripts should not exhibit vulnerabilities or expose the system to potential security threats.
3. **Performance Testing:**
   1. **Objective:** Confirm that the scripts do not adversely affect system performance.
   2. **Method:** Monitor system resources (CPU, memory, disk usage) during script execution.
   3. **Expected Outcome:** Script execution should not significantly impact system performance metrics.

#### Deployment

* **Staging:** Deploy scripts to a controlled staging environment before production rollout. Validate functionality and security posture in this setting.
* **Change Management:** Use a structured change management process for deploying scripts, including review and approval by IT security and compliance teams.
* **User Documentation:** Provide detailed documentation on the use of each script, including prerequisites, steps for execution, and troubleshooting tips.

#### Compliance Verification

* **Internal Audits:** Regularly audit the use and functionality of the scripts to ensure they comply with internal security policies and standards.
* **External Audits:** Subject the scripts and their deployment processes to external audits as required by regulatory standards applicable to the organization.