WxWindow 3.0 Documentation

Tom Smith

Updated 03/28/2025

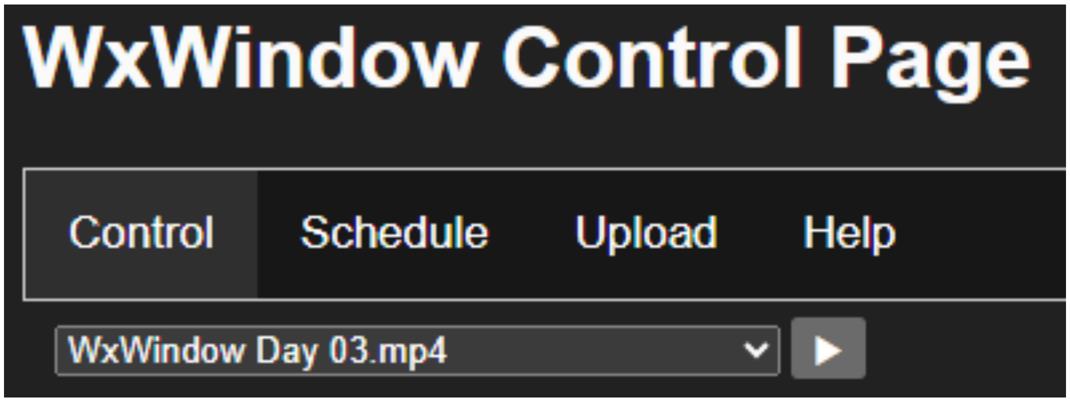
Overview

WxWindow 3.0 is a browser based solution for displaying/changing a video loop within a single connected display. A combination of HTML, CSS, PHP, and Javascript, it's comprised of a control page and a display page. Users can change the current loop, schedule changes to specified loops at desired times, and upload new video loops, all from any browser that can connect to the studio server. The control page is located at http://studiosrv.woodtv.net/wxwindow/control.html while the display page is located at http://studiosrv.woodtv.net/wxwindow.

Operation

All operation is handled through a browser (Chrome or Firefox) at http://studiosrv.woodtv.net/wxwindow/control.html. There are four tabs on this page: Control, Schedule, Upload, and Help.

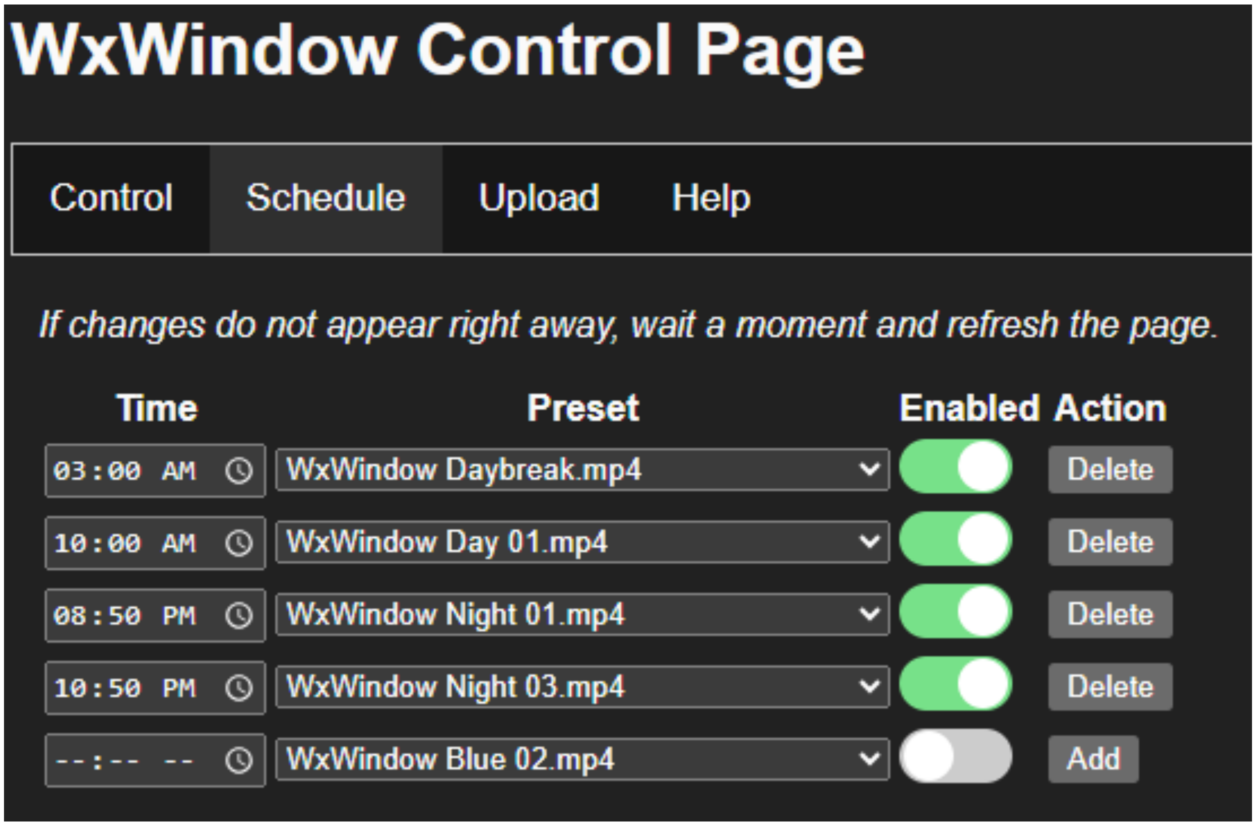
Control



The control page features a drop down menu with a list of available loops. This list is populated automatically from the server. Just make your selection and hit the play button.

***

Schedule



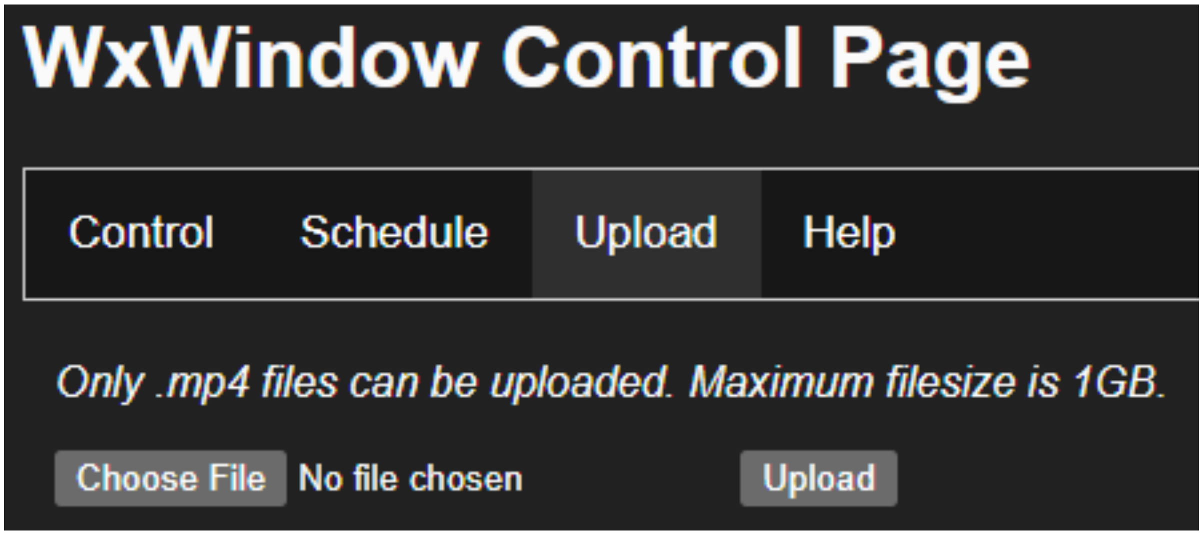
If changes do not appear right away, wait a moment and refresh the page.

Time	Preset	Enabled	Action
03:00 AM	WxWindow Daybreak.mp4	●	Delete
10:00 AM	WxWindow Day 01.mp4	●	Delete
08:50 PM	WxWindow Night 01.mp4	●	Delete
10:50 PM	WxWindow Night 03.mp4	●	Delete
--:-- --	WxWindow Blue 02.mp4	○	Add

The schedule page allows users to add, edit, enable/disable, and remove automated loop changes. To add a new preset, choose a time from the time picker, a preset from the dropdown, and click add. It will be enabled automatically after added. To stop a preset from triggering, you can toggle the green enabled toggle button, or if you want to remove it permanently, you can press Delete. Changes in this screen do not always appear right away. If that happens, please wait a few moments, then refresh the page before trying again.

***

Upload



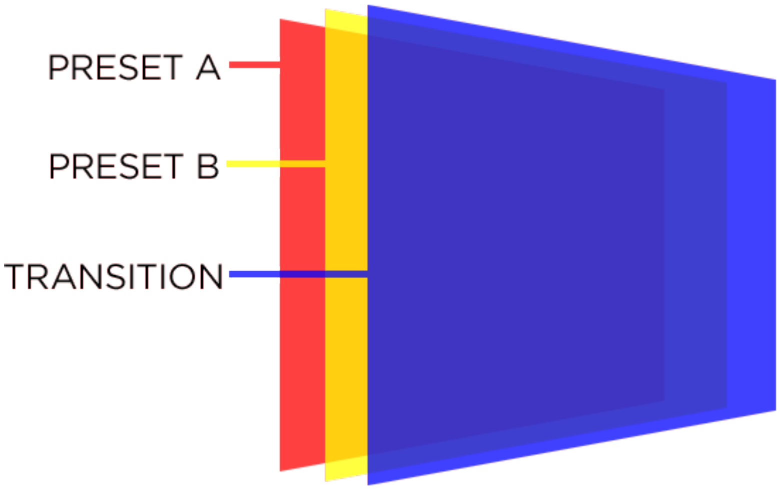
Only .mp4 files can be uploaded. Maximum filesize is 1GB.

No file chosen

Here, a user can upload a new preset to the server. The file should be a .mp4 file with a resolution of 1920x1080 and be a seamless loop. The maximum file size is 1GB. After the upload, the file will appear in the dropdown menu in both the Control and Schedule tabs.

Technical Information

Display Page



The display page is comprised of three layers of video overlayed on one another with CSS at the same X,Y coordinates but different spots on the Z axis. The display page alternates between hiding the Preset A and Preset B layer to change what's being displayed. In this way, it functions like preview and program in a switcher. To make the change happen cleanly, the transition layer (which is the topmost layer) plays, then the switch occurs after 0.5 seconds (approximately 15 frames) when the transition is covering the screen completely. To avoid revealing to an empty frame, the code checks that the next video is loaded completely in the DOM before the transition begins. Video files must be located in a ./videos/ subdirectory of the folder holding the HTML.

For updates, the page also references command.txt, where preset file paths are written by the control page, and times.json, where schedule details are written by the control page.

Reboot Failsafe

To ensure the display persists if the PC is rebooted for any reason, Chrome should be configured to open the display page on boot. Then, launch Run by searching for it in the start menu, and enter shell:startup to open the startup folder. Put a shortcut to Chrome in this folder, then right click it, click Properties, navigate to the Shortcut tab, and in the Target field, add --start-fullscreen after the application path.

Required PHP

- sse.php: EventSource the page monitors in order to know when to reference command.txt for an updated preset.

---

times.json

The times.json file is what the control page references to keep track of automated preset changes. It follows a structure of keys with 24 hour timestamps that contain subitems with keys for the preset to be played at that time and whether or not that automation is enabled, like so:

```
{
    "10:00": {
        "preset": "WxWindow Day 01.mp4",
        "enabled": "true"
    },
    "03:00": {
        "preset": "WxWindow Daybreak.mp4",
        "enabled": "true"
    },
    "20:50": {
        "preset": "WxWindow Night 01.mp4",
        "enabled": "true"
    },
    "22:50": {
        "preset": "WxWindow Night 03.mp4",
        "enabled": "true"
    }
}
```

***

Control Page

The control page does not have any specific underlying concepts that need to be discussed in great detail. Mainly it relies on a series of PHP files to accomplish all its tasks.

Required PHP

- list_videos.php: Grabs a list of videos from the ./videos/ folder to populate the drop down menus with.
- update_time.php: Updates time values for automations in times.json
- add_time.php: Adds automations to times.json
- delete_time.php: Removes automations from times.json
- send_command.php: Updates the text in command.txt to the selected video's file path
- upload.php: Handles uploading .mp4 files into the ./videos/ folder

***

Source Codes

Display Page

HTML (index.html)

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body, html {
            margin: 0;
            padding: 0;
            height: 100%;
            width: 100%;
            overflow: hidden;
          cursor: none;
        }

        #main1, #main2, #main3 {
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            opacity: 1; /* Default opacity for main1 and main3 */
            transition: opacity 1s ease; /* Smooth opacity transition */
        }

        #main2 {
            z-index: 3; /* Transition video on top */
            opacity: 1; /* No need to toggle opacity for transition video */
        }

        #main1 {
            z-index: 1; /* The video behind */
        }

        #main3 {
            z-index: 2; /* The video in front */
        }
    </style>
    <script>
        window.onload = function () {
            let lastCommand = ""; // Variable to track the last received
command
            let lastVisible = "main1"; // Variable to track which video is
currently visible
            let nextVideo = "main3"; // Variable to track the next div for
video loading
          let lastChangedTime = ""; // Variable to keep the video from
changing every ten seconds
```

```
            // Fetch the initial source for main1 from command.txt
            fetch("command.txt")
                .then(response => response.text())
                .then(data => {
                    const initialSrc = data.trim(); // Get the video URL from
the file
                    console.log("Initial video source from command.txt:",
initialSrc);

                    // Set the initial video source for main1
                    const main1 = document.getElementById("main1");
                    const main1Video = document.getElementById("1clip1");
                    main1Video.src = initialSrc; // Set the src of the first
video
                    main1.style.opacity = 1; // Make main1 visible initially
                    main1Video.play(); // Start playing main1 immediately
                })
                .catch(error => {
                    console.error("Error fetching command.txt:", error);
                });

            const eventSource = new EventSource("sse.php");

            eventSource.onmessage = function(event) {
                console.log("SSE Message Received:", event.data);
                const data = JSON.parse(event.data);

                if (data.command && data.command !== lastCommand) { // Only
act if the command is new
                    console.log("New command:", data.command);
                    changeVideo(data.command);
                    lastCommand = data.command; // Update the last received
command
                }
            };

// Fetch the time-specific video mapping from a JSON file
            function fetchVideoForTime() {
    fetch("times.json")
        .then(response => response.json())
        .then(data => {
            // Get the current time formatted as HH:mm (24-hour format)
            const currentTime = new Date();
            const hours = String(currentTime.getHours()).padStart(2, '0'); //
Format as two digits
            const minutes = String(currentTime.getMinutes()).padStart(2,
'0'); // Format as two digits
            const currentTimeFormatted = `${hours}:${minutes}`;
            console.log("Current Time:", currentTimeFormatted);

            // Check if the current time exists as a key in the JSON
            if (data[currentTimeFormatted] && currentTimeFormatted !==
lastChangedTime) {
                const videoSrc = data[currentTimeFormatted].preset;
                const enabled = data[currentTimeFormatted].enabled;
                console.log("Video for this time:", videoSrc);

                // If the video is enabled, change the video
                if (enabled === true) {
                console.log("Changing video.")
                    changeVideo(`./videos/${videoSrc}`);
                }

              if (enabled === "true") {
                console.log("Changing video.")
                changeVideo(`./videos/${videoSrc}`);
            }

                lastChangedTime = currentTimeFormatted; // Update the last
changed time
            } else {
                console.log("No video available for this time or already
changed.");
            }
        })
        .catch(error => {
            console.error("Error fetching times.json:", error);
        });
}

            // Set an interval to check every minute for time-based video
change
            setInterval(fetchVideoForTime, 10000); // Check every 10,000
milliseconds (10 seconds)

            // Initial video check on page load
            fetchVideoForTime();

            function changeVideo(newSrc) {
                console.log("Changing video to:", newSrc);

                let main1 = document.getElementById("1clip1");
                let main2 = document.getElementById("2clip2");
                let main3 = document.getElementById("3clip3");

                // Load new video into the next div (alternating between
main1 and main3)
                if (nextVideo === "main3") {
                    main3.src = newSrc; // Load new video into main3
                } else {
                    main1.src = newSrc; // Load new video into main1
                }

                // Play transition video
                main2.play(); // Transition video plays for 0.5 seconds
before switching
                main2.currentTime = 0; // Reset the transition video to start
from the beginning

                // Wait for 0.5 seconds into the transition animation before
switching
                setTimeout(() => {
                    // Ensure that the transition video is playing before
continuing
                    if (main2.currentTime > 0 && !main2.paused) {
                        // Wait for 0.5 seconds before switching the video
                        setTimeout(() => {
                            // Fade out the current visible video and fade in
the new one
                            if (lastVisible === "main1") {
                                main1.style.opacity = 0; // Hide old video
                                main3.style.opacity = 1; // Show new video
                                lastVisible = "main3"; // Update visibility
tracking
                                nextVideo = "main1"; // Next video should go
to main1
                            } else {
                                main3.style.opacity = 0; // Hide old video
                                main1.style.opacity = 1; // Show new video
                                lastVisible = "main1"; // Update visibility
tracking
                                nextVideo = "main3"; // Next video should go
to main3
                            }
                        }, 500); // Wait for 0.5 seconds before switching the
video
                    }
                }, 100); // Short wait to ensure the transition video has
started
            }
        };

    </script>
</head>
<body>
    <div id="main1"><video id="1clip1" autoplay loop muted></video></div>
    <div id="main2"><video id="2clip2" src="./videos/transition.webm"
muted></video></div>
    <div id="main3"><video id="3clip3" autoplay loop muted></video></div>
</body>
</html>
```

sse.php

```
<?php
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");
header("Connection: keep-alive");

while (true) {
    clearstatcache();
    $commandFile = "command.txt";
    $command = file_exists($commandFile) ?
trim(file_get_contents($commandFile)) : "";

    echo "data: " . json_encode(["command" => $command]) . "\n\n";
    ob_flush();
    flush();
    sleep(1);
}
?>
```

Control Page

HTML (control.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="icon" type="image/x-icon" href="favicon.ico">
<title>WxWindow Control Page</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Control</title>
    <style>
        :root {
  color-scheme: light dark;
}

body {
  font-family: sans-serif;
  color: light-dark(#333b3c, #fafafa);
  background-color: light-dark(#efedea, #212121);
}
.tab {
  overflow: hidden;
  border: 1px solid #ccc;
  background-color: light-dark(#f1f1f1, #171717);
}

.tab button {
  background-color: inherit;
  float: left;
  border: none;
  outline: none;
  cursor: pointer;
  padding: 14px 16px;
  transition: 0.3s;
  font-size: 17px;
}

.tab button:hover {
  background-color: light-dark(#ddd, #2f2f2f);
}

.tab button.active {
  background-color: light-dark(#ccc, #2f2f2f);
}

.tabcontent {
  display: none;
  padding: 6px 12px;
  border-top: none;
}

        .toggle-switch {
            display: inline-block;
            width: 50px;
            height: 24px;
            background: #ccc;
            border-radius: 12px;
            position: relative;
            cursor: pointer;
        }

        .toggle-switch .toggle-knob {
            width: 22px;
            height: 22px;
            background: white;
            border-radius: 50%;
            position: absolute;
            top: 1px;
            left: 2px;
            transition: 0.2s;
        }

        .toggle-switch.active {
            background: #77e189;
        }

        .toggle-switch.active .toggle-knob {
            left: 26px;
        }
    </style>

</head>
<body>
<h1>WxWindow Control Page</h1>
<!-- Tab Buttons -->
<div class="tab">
  <button class="tablinks" onclick="openTab(event, 'Control')"
id="defaultOpen">Control</button>
  <button class="tablinks" onclick="openTab(event, 'Schedule')">Schedule</
button>
  <button class="tablinks" onclick="openTab(event, 'Upload')">Upload</button>
  <button class="tablinks" onclick="openTab(event, 'Documentation')">Help</
button>
</div>
<div id="Control" class="tabcontent">
    <select id="videoSelect"></select>
    <button onclick="sendCommand()">►</button>
</div>
<div id="Schedule" class="tabcontent">
<p><em>If changes do not appear right away, wait a moment and refresh the
page.</em></p>
    <table id="timesTable">
        <thead>
            <tr>
                <th>Time</th>
                <th>Preset</th>
                <th>Enabled</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <!-- Dynamic rows will be added here -->
        </tbody>
    </table>
</div>
<div id="Upload" class="tabcontent">
<p><em>Only .mp4 files can be uploaded. Maximum filesize is 1GB.</em></p>
    <form id="uploadForm">
        <input type="file" name="fileToUpload" id="fileToUpload"
accept=".mp4" required>
        <button type="submit">Upload</button>
    </form>
</div>
<div id="Documentation" class="tabcontent">
<h3>Developed and maintained by <a href="mailto:thomas.smith@woodtv.com?
subject=WxWindow">Tom Smith</a></h3></br>
<em>Cell: (231) 343-9803</em></br>
<em>Desk Ext.: 4319</em></br>
<a href="documentation.pdf">Documentation</a>
<script>
function openTab(evt, tabName) {
    document.querySelectorAll(".tabcontent").forEach(tab => tab.style.display
= "none");
    document.querySelectorAll(".tablinks").forEach(tab =>
tab.classList.remove("active"));
    document.getElementById(tabName).style.display = "block";
    evt.currentTarget.classList.add("active");
}

async function loadTimes() {
    const response = await fetch("times.json");
    const timesData = await response.json();

    const tbody = document.querySelector("#timesTable tbody");
    tbody.innerHTML = ""; // Clear the table

    Object.keys(timesData)
        .sort((a, b) => a.localeCompare(b)) // Ensure chronological order
        .forEach(key => appendTableRow(key, timesData[key].preset,
timesData[key].enabled));

    appendAddRow(tbody); // Ensure add row is at the bottom
}

async function loadVideos() {
        const response = await fetch("list_videos.php");
        const videos = await response.json();
        const select = document.getElementById("videoSelect");

        select.innerHTML = ""; // Clear previous entries
        videos.forEach(video => {
            let option = document.createElement("option");
            option.value = video;
            option.textContent = video;
            select.appendChild(option);
        });
      fetch('command.txt')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.text();
  })
  .then(text => {
    console.log(text); // Process the text content here
    let video = text.replace("./videos/","");
    select.value = video;
  })
  .catch(error => {
    console.error('There was a problem fetching the file:', error);
  })}

async function updateTime(key, field, value) {
    const row = [...document.querySelectorAll("#timesTable tbody
tr")].find(tr =>
        tr.querySelector("input[type='time']").value === key
    );

    if (row) {
        if (field === "preset") {
            row.querySelector("select").value = value;
        } else if (field === "enabled") {
            const toggle = row.querySelector(".toggle-switch");
            toggle.classList.toggle("active", value);
        }
    }

    const formData = new FormData();
    formData.append("key", key);
    formData.append("field", field);
    formData.append("value", value);

    await fetch("update_time.php", { method: "POST", body: formData });

    loadTimes(); // Reload to re-sort entries
}

async function addNewEntry(timePicker, presetDropdown, toggle) {
    const key = timePicker.value;
    const preset = presetDropdown.value;
    const enabled = toggle.classList.contains("active");

    if (!key || !preset) {
        return alert("Please fill in both the time and preset.");
    }

    appendTableRow(key, preset, enabled);
    sortTableEntries();

    const formData = new FormData();
    formData.append("key", key);
    formData.append("preset", preset);
    formData.append("enabled", enabled);

    await fetch("add_time.php", { method: "POST", body: formData });

    loadTimes(); // Reload to ensure correct order and placement
}
```

```
function appendTableRow(key, preset, enabled) {
    const tbody =
document.getElementById("timesTable").querySelector("tbody");
    const row = document.createElement("tr");

    const timeCell = document.createElement("td");
    const timePicker = document.createElement("input");
    timePicker.type = "time";
    timePicker.value = key;
    timeCell.appendChild(timePicker);
    row.appendChild(timeCell);

    const presetCell = document.createElement("td");
    const presetDropdown = document.createElement("select");
    fetch("list_videos.php")
        .then(res => res.json())
        .then(videos => {
            videos.forEach(video => {
                const option = document.createElement("option");
                option.value = video;
                option.textContent = video;
                if (video === preset) {
                    option.selected = true;
                }
                presetDropdown.appendChild(option);
            });
        });
    presetCell.appendChild(presetDropdown);
    row.appendChild(presetCell);

    const toggleCell = document.createElement("td");
    const toggle = document.createElement("div");
    toggle.classList.add("toggle-switch");
    if (enabled) toggle.classList.add("active");
    const knob = document.createElement("div");
    knob.classList.add("toggle-knob");
    toggle.appendChild(knob);
    toggleCell.appendChild(toggle);
    row.appendChild(toggleCell);

    const actionCell = document.createElement("td");
    const deleteButton = document.createElement("button");
    deleteButton.textContent = "Delete";
    deleteButton.onclick = () => deleteEntry(key);
    actionCell.appendChild(deleteButton);
    row.appendChild(actionCell);

    tbody.appendChild(row);

    toggle.addEventListener("click", () => {
        toggle.classList.toggle("active");
        updateTime(key, "enabled", toggle.classList.contains("active"));
    });

    presetDropdown.addEventListener("change", (e) => {
        updateTime(key, "preset", e.target.value);
    });
    timePicker.addEventListener("blur", () => handleTimeChange(key,
timePicker));
    timePicker.addEventListener("keydown", (e) => {
        if (e.key === "Enter") {
            handleTimeChange(key, timePicker);
        }
    });

    sortTableEntries();
}

function sortTableEntries() {
    const tbody = document.querySelector("#timesTable tbody");
    const rows = Array.from(tbody.querySelectorAll("tr"));

    // Remove the add row before sorting
    const addRow = rows.pop();

    rows.sort((a, b) => {
        const timeA = a.querySelector("input[type='time']").value;
        const timeB = b.querySelector("input[type='time']").value;
        return timeA.localeCompare(timeB);
    });

    tbody.innerHTML = ""; // Clear table
    rows.forEach(row => tbody.appendChild(row));
    tbody.appendChild(addRow); // Re-add add row at the bottom
}

async function deleteEntry(key) {
    document.querySelectorAll("#timesTable tbody tr").forEach(row => {
        if (row.querySelector("input[type='time']").value === key) {
            row.remove();
        }
    });

    const formData = new FormData();
    formData.append("key", key);

    await fetch("delete_time.php", { method: "POST", body: formData });

    loadTimes(); // Reload to maintain sorting and add row position
}

function appendAddRow(tbody) {
    const addRow = document.createElement("tr");

    const timeCell = document.createElement("td");
    const timePicker = document.createElement("input");
    timePicker.type = "time";
    timeCell.appendChild(timePicker);
    addRow.appendChild(timeCell);

    const presetCell = document.createElement("td");
    const presetDropdown = document.createElement("select");
    fetch("list_videos.php")
        .then(res => res.json())
        .then(videos => {
            videos.forEach(video => {
                const option = document.createElement("option");
                option.value = video;
                option.textContent = video;
                presetDropdown.appendChild(option);
            });
        });
    presetCell.appendChild(presetDropdown);
    addRow.appendChild(presetCell);

    const toggleCell = document.createElement("td");
    const toggle = document.createElement("div");
    toggle.classList.add("toggle-switch");
    const knob = document.createElement("div");
    knob.classList.add("toggle-knob");
    toggle.appendChild(knob);
    toggleCell.appendChild(toggle);
    addRow.appendChild(toggleCell);

    const actionCell = document.createElement("td");
    const addButton = document.createElement("button");
    addButton.textContent = "Add";
    addButton.onclick = () => addNewEntry(timePicker, presetDropdown,
toggle);
    actionCell.appendChild(addButton);
    addRow.appendChild(actionCell);

    tbody.appendChild(addRow);
}

async function handleTimeChange(oldKey, timePicker) {
        const newKey = timePicker.value;
        if (newKey !== oldKey) {
            // First, delete the old entry with the old key
            await deleteEntry(oldKey, true);

            // Then, update the entry with the new key
            const row = timePicker.closest('tr');
            const preset = row.querySelector('select').value;
            const enabled = row.querySelector('.toggle-
switch').classList.contains("active");
            await addUpdatedEntry(newKey, preset, enabled, row, timePicker);
        }
    }

    // Helper function for deleting an entry by key
    async function deleteEntry(key, isUpdated = false) {
        const formData = new FormData();
        formData.append("key", key);

        const response = await fetch("delete_time.php", {
            method: "POST",
            body: formData
        });

        const result = await response.text();
        console.log(result);
        if (isUpdated) {
            loadTimes(); // Reload after deletion and update
        }
    }

 async function addUpdatedEntry(key, preset, enabled, row, timePicker) {
        const formData = new FormData();
        formData.append("key", key);
        formData.append("preset", preset);
        formData.append("enabled", enabled);

        const response = await fetch("add_time.php", {
            method: "POST",
            body: formData
        });

        const result = await response.text();
        console.log(result);

        // After successfully updating the entry, update the UI time picker
        timePicker.value = key; // Update the time picker with the new key
value

        loadTimes(); // Reload the table to reflect changes
    }

async function sendCommand() {
        const selectedVideo = document.getElementById("videoSelect").value;
        if (!selectedVideo) return alert("Please select a video.");

        const formData = new FormData();
        formData.append("video", "./videos/" + selectedVideo);

        const response = await fetch("send_command.php", {
            method: "POST",
            body: formData
        });

        const result = await response.text();
        console.log(result);
    }
document.getElementById("uploadForm").addEventListener("submit",
function(event) {
            event.preventDefault(); // Prevent normal form submission

            let formData = new FormData();
            let fileInput = document.getElementById("fileToUpload");

            if (fileInput.files.length === 0) {
                alert("Please select an MP4 file to upload.");
                return;
            }

            formData.append("fileToUpload", fileInput.files[0]);

            fetch("upload.php", {
                method: "POST",
                body: formData
            })
            .then(response => response.text()) // Get response as text
            .then(data => {
                alert(data); // Show response in an alert dialog
            })
            .catch(error => {
                alert("Upload failed. Please try again.");
                console.error("Error:", error);
            });
        });
document.addEventListener("DOMContentLoaded", () => {
document.getElementById("defaultOpen").click()
    loadVideos();
    loadTimes();

   });
</script>

</body>
</html>
```

listvideos.php

```
<?php
$videoDir = __DIR__ . "/videos/";
$videos = [];

if (is_dir($videoDir)) {
    foreach (scandir($videoDir) as $file) {
        if (preg_match('/\.(mp4|webm|avi|mov)$/i', $file)) {
            $videos[] = $file;
        }
    }
}

header("Content-Type: application/json");
echo json_encode($videos);
?>
```

update_time.php

```
<?php
// update_time.php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $key = $_POST['key'] ?? null;
    $field = $_POST['field'] ?? null;
    $value = $_POST['value'] ?? null;

    if (!$key || !$field || !$value) {
        echo "Invalid entry!";
        exit;
    }

    // Read the current JSON data
    $jsonFile = 'times.json';
    $data = json_decode(file_get_contents($jsonFile), true);

    // Check if the key exists and update the field
    if (isset($data[$key])) {
        $data[$key][$field] = $value;

        // Write back to the JSON file
        file_put_contents($jsonFile, json_encode($data, JSON_PRETTY_PRINT));
        echo "Entry updated!";
    } else {
        echo "Entry not found!";
    }
}
?>
```

---

add_time.php

```
<?php
    if ($_SERVER["REQUEST_METHOD"] === "POST") {
        $key = $_POST['key'];
        $preset = $_POST['preset'];
        $enabled = $_POST['enabled'];

        $jsonFile = 'times.json';

        if (file_exists($jsonFile)) {
            $jsonData = json_decode(file_get_contents($jsonFile), true);

            // Update or add the entry with the new key
            $jsonData[$key] = [
                'preset' => $preset,
                'enabled' => $enabled
            ];

            file_put_contents($jsonFile, json_encode($jsonData,
JSON_PRETTY_PRINT));
            echo "Entry with key $key updated successfully.";
        } else {
            echo "Error: File not found.";
        }
    }
?>
```

---

delete_time.php

```
<?php
    if ($_SERVER["REQUEST_METHOD"] === "POST") {
        $key = $_POST['key'];
        $jsonFile = 'times.json';

        if (file_exists($jsonFile)) {
            $jsonData = json_decode(file_get_contents($jsonFile), true);
            unset($jsonData[$key]); // Remove the entry with the specified
key
```

```
            file_put_contents($jsonFile, json_encode($jsonData,
JSON_PRETTY_PRINT));
            echo "Entry with key $key deleted successfully.";
        } else {
            echo "Error: File not found.";
        }
    }
?>
```

---

send_command.php

```
<?php
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $video = $_POST["video"] ?? "";

    if (!empty($video) && file_exists(__DIR__ . "/" . $video)) {
        file_put_contents("command.txt", $video);
        echo "Command saved: " . $video;
    } else {
        echo "Error: Invalid video file.";
    }
}
?>
```

---

upload.php

```
<?php
$target_dir = "videos/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

// Allow only MP4 files
if ($imageFileType != "mp4") {
    echo "Sorry, only an MP4 file can be uploaded.";
    $uploadOk = 0;
}

// Validate MIME type
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime = finfo_file($finfo, $_FILES["fileToUpload"]["tmp_name"]);
finfo_close($finfo);

if ($mime != "video/mp4") {
    echo "Invalid file type. Only MP4 videos are allowed.";
    $uploadOk = 0;
}

// Limit file size to 1GB
if ($_FILES["fileToUpload"]["size"] > 1073741824) { // 1GB limit
    echo "File is too large. Maximum allowed size is 1GB.";
    $uploadOk = 0;
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
$target_file)) {
        echo "The file " . htmlspecialchars(basename($_FILES["fileToUpload"]
["name"])) . " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```