Scene Scheduler - User Manual

Version: 0.4 **Date:** October 28, 2025 **Application:** Scene Scheduler for OBS Studio

---

Table of Contents

---

1. Getting Started

1.1 What is Scene Scheduler?

Scene Scheduler is an **external automation tool** for OBS Studio that automates your broadcast schedule like a television station. It manages programmed content playback based on precise time schedules, enabling fully automated 24/7 streaming without manual intervention.

Core Purpose: Scene Scheduler is designed to **automate broadcast schedules** - allowing you to plan ahead what content will air and when, then letting the system execute those transitions automatically. Think of it as creating a TV channel programming guide that OBS follows automatically.

How it works: - You create a **schedule** (programming grid) defining what content plays at specific times - Scene Scheduler monitors the clock and **automatically triggers scene/source changes** when each event's time arrives - The system runs continuously, executing your programmed schedule 24/7 without human intervention - A web-based **Monitor View** lets you observe the schedule and current status from anywhere - An optional **Editor View** provides a visual calendar interface to modify the schedule

Why use Scene Scheduler? - **24/7 automation**: Perfect for streaming channels, digital signage, worship services, conferences, or any scheduled broadcast - **Zero manual intervention**: Once programmed, the schedule executes automatically - **Network accessible**: Monitor and edit from any device on your network (laptop, tablet, phone) - **Reduced server load**: Web interface runs on client devices, not the OBS machine - **Safe external architecture**: Runs outside OBS, so crashes don't affect your broadcast

1.2 Prerequisites

Before installing Scene Scheduler, ensure you have:

1. **OBS Studio** (version 28.0 or higher recommended) - Download from: https://obsproject.com/

2. **OBS WebSocket Plugin** (version 5.x) - OBS Studio 28+ includes this by default - For older versions, install from: https://github.com/obsproject/obs-websocket

3. **Operating System**: - **Linux**: Tested on Ubuntu 20.04+, other distributions should work - **Windows**: Windows 10/11 (64-bit)

4. **Network**: OBS and Scene Scheduler must be on the same machine or accessible via network

1.3 Quick Start Installation

Linux Installation

Step 1: Download Scene Scheduler

```
bash
# Extract the downloaded archive
tar -xzf scenescheduler-linux-amd64.tar.gz
cd scenescheduler
```

Windows Installation

Step 1: Download Scene Scheduler 1. Extract the downloaded `scenescheduler-windows-amd64.zip` file 2. Extract the ZIP file to a folder (e.g., `C:\scenescheduler\`) 3. Open Command Prompt or PowerShell in that folder

Step 2: Configure OBS WebSocket

1. Open OBS Studio
2. Go to **Tools → WebSocket Server Settings**
3. Enable "Enable WebSocket server"
4. Set a password (recommended) or leave blank for local-only access
5. Note the port (default: 4455)
6. Click **OK**

Step 3: Configure Scene Scheduler

Edit `config.json`:

```
json
{
  "obsWebSocket": {
    "host": "localhost",
    "port": 4455,
    "password": "your-obs-password"
  },
  "webServer": {
    "host": "0.0.0.0",
    "port": 8080,
    "hlsPath": "hls"
  },
  "schedule": {
    "jsonPath": "schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "./hls-generator"
  }
}
```

Critical configuration notes: - `obsWebSocket.password`: Must match your OBS WebSocket password - `webServer.hlsPath`: Directory for HLS preview files (relative to executable) - `schedule.scheduleSceneAux`: Name of the auxiliary OBS scene (created automatically if it doesn't exist)

Step 4: Auxiliary Scene

Scene Scheduler automatically creates an auxiliary scene in OBS when it starts. This scene is used as a "staging area" to prepare sources before transitioning to them, ensuring smooth transitions without visible loading delays.

Scene name configuration: - The auxiliary scene name is configured in `config.json` under `schedule.scheduleSceneAux` - Default name: `scheduleSceneAux` - If the scene doesn't exist, Scene Scheduler creates it automatically - The scene is managed entirely by Scene Scheduler (don't add sources manually)

Step 5: Start Scene Scheduler

Linux:

```
bash
# Make executable
chmod +x scenescheduler

# Run
./scenescheduler
```

Windows:

```
cmd
REM Run in Command Prompt
scenescheduler.exe

REM Or double-click scenescheduler.exe in File Explorer
```

You should see output like:

```
2025/10/28 10:30:15 INFO Scene Scheduler starting version=1.6
2025/10/28 10:30:15 INFO WebSocket connecting host=localhost port=4455
2025/10/28 10:30:15 INFO Connected to OBS Studio version=30.0.0
2025/10/28 10:30:15 INFO Web server listening address=http://0.0.0.0:8080
2025/10/28 10:30:15 INFO Schedule loaded events=0
```

Step 6: Access Web Interface

Open your browser and navigate to Scene Scheduler. You can access it from:

- **Same machine**: `http://localhost:8080`
- **Other devices on network**: `http://<server-ip>:8080`

- Example: `http://192.168.1.100:8080`
- Replace `<server-ip>` with the actual IP address of the machine running Scene Scheduler

Finding your server IP address:

Linux:

```
bash
ip addr show | grep inet
```

Windows:

```
cmd
ipconfig
```

Look for the IPv4 address on your active network interface (usually starts with 192.168.x.x or 10.x.x.x).

Why remote access? The key benefit of Scene Scheduler is that you can control and monitor OBS from **any device on your network** (laptop, tablet, phone), reducing load on the machine running OBS and allowing multiple people to monitor the schedule simultaneously.

You should see the Scene Scheduler web interface with two main views: - **Monitor View**: Displays current and upcoming events (read-only) - **Editor View**: Visual editor for schedule.json

1.4 Your First Schedule Event

Let's create a simple event that switches to a scene at a specific time:

1. **Open Editor View** (click "Editor" button in top navigation)

2. **Add a new event** (click "+ Add Event" button)

3. **Configure the event** in the modal dialog: - **Time**: Set to a few minutes from now (e.g., if it's 10:30, set 10:35) - **OBS Scene**: Select an existing scene from your OBS (e.g., "Scene 1") - **Duration**: Leave at default (00:05:00 = 5 minutes) - **Sources Tab**: Leave empty for now (just scene switching)

4. **Save** the event (click "Save Event")

5. **Observe**: - The event appears in your schedule list - When the scheduled time arrives, OBS automatically switches to the selected scene - Monitor view shows "CURRENT EVENT" highlighting

Congratulations! You've created your first automated scene transition.

1.5 Understanding the Monitor View

The Monitor View is designed for **passive observation**. It's perfect for: - Displaying on a secondary monitor in a control room - Sharing with team members who need visibility but not edit access - Checking current status without risk of accidental changes

What you see: - **Current time** (updates every second) - **Active event** (highlighted with countdown timer) - **Next events** (upcoming schedule preview) - **Color coding**: - Green: Current active event - Yellow: Next event (starts soon) - ○ White: Future events

1.6 Understanding the Editor View

The Editor View provides **full schedule control**. Use it to: - Add, edit, and delete events - Reorder schedule entries - Configure complex source setups - Preview sources before committing

Key interface elements: - **+ Add Event**: Creates new schedule entry - **Event list**: Shows all scheduled events with controls - **Edit button** (pencil icon): Opens event configuration modal - **Delete button** (trash icon): Removes event - **Drag handle**: Reorder events by dragging

---

2. Understanding Scene Scheduler

2.1 Core Concepts

Before diving into advanced features, let's understand how Scene Scheduler thinks about **time-based automation**:

Events (Scheduled Programs)

An **event** is a **time-scheduled instruction** that tells OBS to: 1. **At a specific time** (e.g., 14:30:00): Switch to a specific OBS scene 2. **Optionally**: Add/configure scene-specific sources (media files, streams, browser sources) 3. **For a duration** (e.g., 30 minutes): Keep that configuration active 4. **Then cleanup**: Remove added sources when the event ends

Think of events as the individual "shows" or "segments" in your broadcast schedule. Events are the fundamental building blocks of your programming grid.

Scenes

A **scene** in OBS is a collection of sources (video, audio, images, etc.) arranged in a specific layout. Scene Scheduler doesn't create scenes—it uses your existing OBS scenes and enhances them by: - Dynamically adding/removing sources based on the schedule - Preparing sources in the background before they're visible - Cleaning up after an event ends

Sources

A **source** is any content element in OBS: - Media files (videos, audio) - Browser sources (web pages, HTML overlays) - Streaming inputs (RTMP, RTSP, RTP, SRT) - Images - VLC playlists

Scene Scheduler can configure these sources automatically per event.

The Auxiliary Scene (`scheduleSceneAux`)

This is Scene Scheduler's "backstage area"—a hidden scene where: - Sources are loaded and prepared before they're needed - Streaming inputs are tested for connectivity - Media files are preloaded to avoid visible delays

You never see this scene during broadcast, but it's critical for smooth operation.

2.2 Architecture Overview

Scene Scheduler uses a **distributed client-server architecture** that allows remote access and operation:

```
┌──────────────────────────────────────────────────────────────┐
│                   Production Server                          │
│                                                              │
│  ┌────────────────┐        WebSocket           ┌─────────┐  │
│  │ OBS Studio     │ ◄──────────────────────────│ Scene   │  │
│  │                │       (localhost)           │ Scheduler│ │
│  │  - Scenes      │                             │          │ │
│  │  - Sources     │                             │ (Backend)│ │
│  │  - Rendering │                             └────┬─────┘ │
│  └──────────────┘                                  │       │
│                                                     │       │
│                                HTTP Server (0.0.0.0:8080)  │
│                                                     │       │
└─────────────────────────────────────────────────────┼───────┘
                                                      │
                             Network (LAN/Internet)   │
                                                      │
        ┌─────────────────────────────────────────────┴───────────┐
        │                                                         │
   ┌────▼─────┐                  ┌────▼─────┐            ┌───▼──────┐
   │ Laptop   │                  │  Tablet  │            │  Phone   │
   │ Browser  │                  │ Browser  │            │ Browser  │
   │          │                  │          │            │          │
   │ Monitor  │                  │  Editor  │            │ Monitor  │
   │  View    │                  │   View   │            │   View   │
   └──────────┘                  └──────────┘            └──────────┘
```

Communication flow: 1. **Backend ↔ OBS**: WebSocket connection for scene/source control (localhost) 2. **Backend → Internet**: HTTP server binds to 0.0.0.0 (accessible from network) 3. **Remote Clients → Backend**: HTTP/WebSocket from any device on network 4. **Backend → All Clients**: Real-time broadcasts of schedule updates

Key architectural benefits: - **Distributed access**: Control from anywhere on the network (or internet if exposed) - **Reduced server load**: Web UI runs on client devices, not the OBS machine - **Multi-user monitoring**: Multiple people can view Monitor View simultaneously - **Flexible deployment**: Server doesn't need display, keyboard, or GUI - **Scalability**: Add as many monitoring clients as needed without affecting performance

2.3 How Scene Transitions Work

Scene Scheduler uses a sophisticated **staging system** to ensure smooth transitions without visual artifacts. The system operates when a scheduled event's time arrives.

The Staging Process (5 Steps):

```
Step 1: STAGING (Preparation in Background)
│  - New source created in scheduleSceneAux (auxiliary/temporary scene)
│  - Source fully configured but remains invisible to viewers
│  - All transformations applied (position, scale, crop, etc.)
│  - Media files and streams begin loading
│  - If this step fails: Process stops, current broadcast unaffected
│
Step 2: ACTIVATION (Move to Visible Scene)
```

```
│  - Source moved from scheduleSceneAux to target OBS scene
│  - Source made visible to audience
│  - Transition happens instantly (source already prepared)
│  - If this step fails: Fallback to previous content
│
Step 3: SCENE SWITCH (OBS Scene Change)
│  - OBS transitions to the target scene
│  - Audience sees new content immediately
│  - No buffering or loading delays (thanks to staging)
│  - If this step fails: Rollback, previous content maintained
│
Step 4: CLEANUP (Remove Temporary Elements)
│  - Temporary element removed from scheduleSceneAux
│  - Resources freed for next event
│  - Auxiliary scene ready for next staging operation
│
Step 5: MONITOR (Ongoing Management)
│  - Scene remains active for programmed duration
│  - When event ends: Source automatically removed
│  - System ready for next scheduled event
```

Key Benefits: - **No visible loading**: Sources prepared before they're shown - **Atomic transitions**: Either complete success or safe rollback - **Resource efficiency**: Cleanup prevents memory leaks - **Continuous operation**: System handles 24/7 automated scheduling

2.4 Schedule Execution Model

Scene Scheduler uses a **time-based trigger system**:

1. **Schedule Loading**: On startup, `schedule.json` is loaded and parsed
2. **Event Queue**: Events are sorted by time and monitored continuously
3. **Trigger Detection**: Every second, the scheduler checks if any event's time has arrived
4. **Execution**: When event time arrives, the 5-step staging process begins (see Section 2.3)
5. **Cleanup**: After event duration expires, resources are cleaned up

Important: Events are **time-triggered**, not sequential. If an event's time is missed (e.g., Scene Scheduler was stopped), it won't execute when restarted—only upcoming events run.

2.5 Real-Time Synchronization

All connected clients (Monitor and Editor views) receive **instant updates** via WebSocket:

- **Schedule changes**: Adding/editing/deleting events updates all clients immediately
- **Current event tracking**: All views highlight the active event
- **OBS state changes**: If you manually change scenes in OBS, clients are notified
- **Connection status**: Visual indicators show OBS connection state

This enables **collaborative operation**: multiple team members can monitor the same schedule from different devices.

---

3. Web Interface Overview

3.1 Interface Modes

Scene Scheduler provides two distinct web interfaces optimized for different use cases:

Monitor View (`/`)

Purpose: Passive observation and status monitoring

Use cases: - Wall-mounted displays in broadcast control rooms - Secondary monitors for operators - Public-facing status boards - Mobile devices for quick status checks

Features: - Large, readable typography - Current event prominently displayed - Countdown timer to next event - No edit controls (prevents accidental changes) - Auto-updating every second

Access URLs: - Same machine: `http://localhost:8080/` - Network access: `http://<server-ip>:8080/`

Editor View (`/editor.html`)

Purpose: Visual editor for `schedule.json`

What it does: - Edit the schedule.json file through a web interface - Add, modify, or delete events - Configure event sources (media, browser, streams) - Save changes back to schedule.json

Features: - Event list with add/edit/delete buttons - Modal dialog for event configuration - Optional source preview (testing tool) - Visual time/duration pickers

Access URLs: - Same machine: `http://localhost:8080/editor.html` - Network access: `http://<server-ip>:8080/editor.html`

3.2 Navigation

Both views include a **top navigation bar** with:

```
┌─────────────────────────────────────────────────────────────┐
│  Scene Scheduler    [Monitor] [Editor]    ● Connected     │
└─────────────────────────────────────────────────────────────┘
```

- **Scene Scheduler**: Application title/logo
- **[Monitor]**: Button to switch to Monitor view
- **[Editor]**: Button to switch to Editor view
- **Connection indicator**:
- **Green dot**: Connected to OBS and backend
- **Red dot**: Disconnected (check OBS and backend status)

3.3 Monitor View Layout

```
┌───────────────────────────────────────────────────────────────┐
│                    CURRENT TIME: 14:35:22                     │
├───────────────────────────────────────────────────────────────┤
│                                                               │
│   ┌─────────────────────────────────────────────────────┐   │
│   │   CURRENT EVENT                                     │   │
│   │                                                     │   │
│   │ 14:30:00 → Scene: Afternoon Show                    │   │
│   │ Duration: 1h 00m                                    │   │
│   │ Ends in: 24m 38s                                    │   │
│   │                                                     │   │
│   │ Sources:                                            │   │
│   │  • Media: /videos/intro.mp4                         │   │
│   │  • Browser: https://overlay.example.com             │   │
│   └─────────────────────────────────────────────────────┘   │
│                                                               │
│  UPCOMING EVENTS                                              │
│   ┌─────────────────────────────────────────────────────┐   │
│    │ 15:30:00 → Scene: News Segment                     │   │
│    │ Starts in: 54m 38s                                 │   │
│    └────────────────────────────────────────────────────┘   │
│                                                             │
│    ┌────────────────────────────────────────────────────┐   │
│    │ 16:00:00 → Scene: Weather Report                   │   │
│    │ Starts in: 1h 24m 38s                              │   │
│    └────────────────────────────────────────────────────┘   │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```

Color coding: - **Green background**: Currently active event - **Yellow background**: Next event (starts within 30 minutes) - **White background**: Future events

Time displays: - **Absolute time**: Event start time (HH:MM:SS format) - **Relative time**: Countdown or time remaining - **Duration**: How long the event runs

3.4 Editor View Layout

```
┌───────────────────────────────────────────────────────────────┐
│                  [+ Add Event]            Current: 14:35      │
├───────────────────────────────────────────────────────────────┤
│                                                               │
│                                                             │
│   ┌─────────────────────────────────────────────────────┐   │
│   │   14:30:00 → Afternoon Show        [Edit] [Delete]│    │
│   │    Duration: 1h 00m  │  Ends: 15:30:00              │     │
│   │    Sources: 2 configured                            │     │
│   └─────────────────────────────────────────────────────┘   │
│                                                               │
│   ┌─────────────────────────────────────────────────────┐   │
│   │   15:30:00 → News Segment           [Edit] [Delete]│    │
│   │    Duration: 30m  │  Ends: 16:00:00                 │     │
│   │    Sources: 1 configured                            │     │
│   └─────────────────────────────────────────────────────┘   │
│                                                               │
│   ┌─────────────────────────────────────────────────────┐   │
│   │   16:00:00 → Weather Report         [Edit] [Delete]│    │
│   │    Duration: 15m  │  Ends: 16:15:00                 │     │
│   │    Sources: 0 configured                            │     │
│   └─────────────────────────────────────────────────────┘   │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```

Interface elements: - **+ Add Event button**: Creates new schedule entry (opens modal) - **Event cards**: Each event shown as a card with summary info - **Edit button** (pencil icon): Opens configuration modal for that event - **Delete button** (trash icon):

Removes event (with confirmation) - **Drag handle** (⋮⋮ icon): Reorder events by dragging

Event card information: - **Time and scene name**: Primary identification - **Duration and end time**: Calculated automatically - **Source count**: Number of configured sources - **Current event indicator**: Green background for active event

3.5 Event Configuration Modal

When you click "Add Event" or "Edit" on an existing event, a modal dialog opens with five tabs:

```
┌─────────────────────────────────────────────────────────┐
│  Configure Event                                   [×]  │
├─────────────────────────────────────────────────────────┤
│  [General] [Media] [Browser] [FFMPEG] [Preview]        │
├─────────────────────────────────────────────────────────┤
│                                                         │
│  (Tab content appears here)                             │
│                                                         │
│                                                         │
├─────────────────────────────────────────────────────────┤
│                          [Cancel]  [Save Event]        │
└─────────────────────────────────────────────────────────┘
```

Five tabs: 1. **General**: Time, scene selection, duration 2. **Media**: Video/audio file sources 3. **Browser**: Web page and HTML overlay sources 4. **FFMPEG**: Network streaming inputs (RTMP, RTSP, RTP, SRT, NDI) 5.**Preview**: Real-time source preview (test before saving)

We'll explore each tab in detail in Section 5.

---

4. Schedule Management

4.1 Understanding Your Schedule

Your schedule is stored in `schedule.json` as a time-ordered list of events. Each event defines: - **When**: The exact time to execute (HH:MM:SS format) - **What**: Which OBS scene to activate - **How long**: Duration the scene remains active - **Content**: Optional sources to add to the scene

Key principles: 1. **Events are time-triggered**: They execute at their scheduled time regardless of previous events 2. **Events can overlap**: Multiple events can be configured for the same time (though this may cause conflicts) 3. **Events don't loop**: Each event runs once per day unless explicitly repeated 4. **Changes are immediate**: Editing the schedule updates OBS in real-time

4.2 Creating Your First Event

Let's walk through creating a complete event step by step:

Step 1: Open Editor View - Navigate to Editor View: - Same machine: `http://localhost:8080/editor.html` - Network access: `http://<server-ip>:8080/editor.html` - Click the **"+ Add Event"** button in the top bar - The event configuration modal opens

Step 2: Configure General Settings (General Tab)

The General tab contains the essential event parameters:

1. **Time** (required) - Format: HH:MM:SS (24-hour clock) - Example: `14:30:00` for 2:30 PM - Must be a valid time (00:00:00 to 23:59:59) - **Tip**: Set times a few minutes in the future for testing

2. **OBS Scene** (required) - Dropdown shows all scenes currently configured in OBS - Select the scene you want to activate - **Important**: The scene must exist in OBS before scheduling - If the dropdown is empty, check your OBS connection

3. **Duration** (required) - Format: HH:MM:SS (hours:minutes:seconds) - Default: `00:05:00` (5 minutes) - Examples:

    - `00:30:00` = 30 minutes
    - `01:00:00` = 1 hour
    - `00:00:30` = 30 seconds
    - **Tip**: Duration determines when sources are cleaned up

4. **Event Name** (optional) - A descriptive label for your event - Shown in monitor and editor views - Example: "Morning News", "Afternoon Show", "Commercial Break" - If empty, the scene name is used

Example configuration:

```
Time:     14:30:00
Scene:    Afternoon Show
```

```
Duration: 01:00:00
Name:     Daily Afternoon Broadcast
```

Step 3: Add Sources (Optional)

If your event needs specific content (videos, overlays, streams), configure them in the source tabs: - **Media Tab**: Add video or audio files - **Browser Tab**: Add web pages or HTML overlays - **FFMPEG Tab**: Add network streaming inputs

We'll cover source configuration in detail in Section 5.

Step 4: Preview Sources (Optional)

Before saving, you can test your sources using the **Preview Tab**: - Generates a real-time HLS stream of each source - Plays in the browser for verification - Automatically stops after 30 seconds - See Section 5.6 for comprehensive preview documentation

Step 5: Save the Event

Click **"Save Event"** at the bottom of the modal. The system: 1. Validates all fields 2. Adds the event to `schedule.json` 3. Broadcasts the update to all connected clients 4. Closes the modal 5. Shows the new event in the editor list

Step 6: Verify

After saving: - The event appears in the Editor View event list - Monitor View shows it in upcoming events - OBS will automatically switch to the scene at the scheduled time

4.3 Editing Existing Events

To modify an event:

1. **Locate the event** in Editor View
2. **Click the Edit button** (pencil icon) on the event card
3. **Make changes** in any of the five tabs
4. **Save** to apply changes immediately

Important notes: - Changes to past events have no effect (events are time-triggered) - Editing an active event updates OBS immediately - Source changes take effect on the next event trigger

Common edits: - **Adjust timing**: Change the time field to reschedule - **Change duration**: Extend or shorten the event - **Swap scenes**: Select a different OBS scene - **Update sources**: Add, remove, or modify source configurations - **Fix errors**: Correct invalid file paths or URLs

4.4 Deleting Events

To remove an event:

1. **Click the Delete button** (trash icon) on the event card
2. **Confirm deletion** in the dialog (if enabled)
3. The event is immediately removed from the schedule

What happens: - Event disappears from all views (Monitor and Editor) - `schedule.json` is updated - If the event is currently active: - Sources are cleaned up immediately - OBS remains on the current scene (no automatic switch) - Next scheduled event will trigger normally

Tip: To temporarily disable an event without deleting it, you can: - Change its time to far in the future (e.g., 23:59:59) - Or delete and re-add later using your browser's undo function

4.5 Reordering Events

Events in Editor View can be reordered for visual organization:

1. **Hover over the drag handle** (⋮⋮ icon) on an event card
2. **Click and drag** the event to a new position
3. **Release** to drop it in place

Important: Reordering in the UI is purely visual—events still execute based on their **time** field, not their position in the list. Reordering is useful for: - Grouping related events together - Separating different "shows" or time blocks - Matching a physical run-down sheet

4.6 Schedule Validation

Scene Scheduler performs validation when you save events:

Field validation: - **Time format**: Must be HH:MM:SS (e.g., 14:30:00) - **Scene exists**: Selected scene must be present in OBS - **Duration format**: Must be HH:MM:SS and greater than zero - **Source paths**: File paths must exist (checked at staging time) - **URLs**: Browser and FFMPEG URLs must be valid format

Common validation errors:

Error Message	Cause	Solution
"Invalid time format"	Time not in HH:MM:SS	Use 24-hour format: 14:30:00
"Scene not found"	Scene deleted from OBS	Create scene in OBS first
"Invalid duration"	Duration is zero or negative	Set positive duration
"Invalid URL"	Malformed URL in browser/FFMPEG	Check URL syntax
"File not found"	Media path doesn't exist	Verify file path on disk

When validation occurs: - **Client-side**: Form fields are validated as you type - **Server-side**: Full validation when saving event - **Event trigger time**: File existence and connectivity checked when event executes

4.7 Schedule Persistence

Your schedule is stored in `schedule.json` in the application directory:

Auto-save behavior: - Every change (add/edit/delete) immediately writes to disk - No manual "save" required - Changes persist across application restarts

Backup recommendations: 1. **Manual backups**: Copy `schedule.json` periodically 2. **Version control**: Store in git for change tracking 3. **Automated backups**: Use system backup tools to include the application directory

Restoring from backup:

```
bash
# Stop Scene Scheduler
pkill scenescheduler

# Restore backup
cp schedule.json.backup schedule.json

# Restart Scene Scheduler
./scenescheduler
```

4.8 Multi-Day Schedules

Scene Scheduler operates on a **24-hour clock** (00:00:00 to 23:59:59). For multi-day operations:

Approach 1: Daily repetition (manual) - Configure events for a single day - Copy and adjust times for subsequent days - Use descriptive names to track days (e.g., "Monday Morning Show")

Approach 2: 24/7 continuous schedule - Create events that span the entire day - Use long durations for overnight periods - Example: An event at 23:00:00 with 8-hour duration covers overnight

Approach 3: Weekly patterns - Create a template for each day of the week - Manually switch schedules daily (replace `schedule.json`) - Consider scripting for automation

Note: Scene Scheduler v1.6 does not support automatic daily repetition or day-of-week conditions. Each event runs once per day at its scheduled time.

4.9 Handling Schedule Conflicts

What is a schedule conflict? Two or more events scheduled for the exact same time.

How Scene Scheduler handles conflicts: - Events are processed in the order they appear in `schedule.json` - Later events **override** earlier events - Only the last event's sources are visible

Example conflict:

```
json
[
  {
    "time": "14:00:00",
    "scene": "Scene A",
    "duration": "01:00:00"
  },
  {
    "time": "14:00:00",
    "scene": "Scene B",
    "duration": "00:30:00"
  }
]
```

Result: At 14:00:00, both events trigger, but Scene B is activated (it's last). Scene A's sources may be staged but never shown.

Best practices to avoid conflicts: 1. **Stagger times**: Use minute or second offsets (14:00:00, 14:01:00) 2. **Review schedule visually**: Editor View shows all events chronologically 3. **Use unique times**: Avoid duplicating times unless intentional 4. **Plan transitions**: Allow buffer time between events (e.g., 30 seconds)

4.10 Testing Your Schedule

Before relying on your schedule for production:

Test Checklist:

1. **Create test events** - Set times 2-3 minutes in the future - Use short durations (1-2 minutes) - Test with simple scenes first (no sources)

2. **Verify scene transitions** - Watch OBS at the scheduled time - Confirm scene switches automatically - Check Monitor View highlights the correct event

3. **Test source loading** - Add a media source to an event - Verify it appears in OBS at the scheduled time - Confirm playback starts automatically

4. **Test event editing** - Edit an upcoming event - Change the time slightly - Verify OBS responds to the new time

5. **Test cleanup** - Wait for event duration to expire - Confirm sources are removed from OBS - Check `scheduleSceneAux` is cleared

6. **Test multiple events** - Schedule 3-4 events in sequence - Verify each transitions smoothly - Watch for any overlap issues

Troubleshooting test failures: - See Section 10 (Troubleshooting) for detailed diagnostic steps

---

5. Configuring Events

This section covers the five configuration tabs in the event modal dialog. Each tab manages a different aspect of your event's sources and behavior.

5.1 General Tab

The General tab contains the core event parameters:

Time Field

Format: HH:MM:SS (24-hour clock)

Examples: - `00:00:00` - Midnight - `09:30:00` - 9:30 AM - `14:45:30` - 2:45 PM and 30 seconds - `23:59:59` - One second before midnight

Validation: - Hours: 00-23 - Minutes: 00-59 - Seconds: 00-59 - Leading zeros required (use `09:00:00`, not `9:0:0`)

Tips: - For testing, set times 2-5 minutes in the future - Use seconds for precise timing (e.g., commercial break syncing) - Remember: events trigger once per day at this time

OBS Scene Field

Purpose: Select which scene OBS switches to when this event triggers.

How it works: 1. Dropdown is populated from OBS Studio's current scene list 2. List updates automatically when you add/remove scenes in OBS 3. Selected scene must exist when event triggers (or event fails)

Troubleshooting: - **Empty dropdown**: OBS WebSocket connection lost (check connection indicator) - **Scene missing**: Scene was deleted in OBS (recreate or select different scene) - **Scene grayed out**: `scheduleSceneAux` cannot be selected (reserved for staging)

Duration Field

Format: HH:MM:SS (hours:minutes:seconds)

What it controls: - How long sources remain active - When cleanup occurs - Implicit "end time" of the event (start time + duration)

Examples: - `00:00:30` - 30-second short event (bumpers, stingers) - `00:05:00` - 5-minute event (news segments, commercials) - `00:30:00` - 30-minute event (shows, programs) - `01:00:00` - 1-hour event (long-form content) - `02:30:00` - 2.5-hour event (movies, extended programming)

Important notes: - Duration determines when **sources are cleaned up**, not when the scene changes - OBS stays on the scene until the next event triggers - Minimum duration: 1 second (`00:00:01`) - Maximum duration: 23:59:59 (just under 24 hours)

Event Name Field (Optional)

Purpose: A human-readable label for the event.

Usage: - Shown in Monitor View and Editor View - Helps identify events in lists - Does not affect OBS (purely for organization)

Best practices: - Use descriptive names: "Morning News", "Afternoon Show", "Commercial Block 1" - Include day if scheduling multi-day: "Monday Morning Show" - Keep concise (displays better in UI)

Default behavior: If left empty, the scene name is used as the event name.

5.2 Media Tab

The Media tab configures **media_source** and **vlc_source** types—video and audio files that play during the event.

When to Use Media Sources

- Playing pre-recorded video files
- Background music or audio tracks
- Video loops (with loop option enabled)
- Intro/outro videos for shows

Adding a Media Source

Step 1: Source Type Select the source type: - **media_source**: OBS native media source (recommended) - Supports: MP4, MOV, AVI, MKV, FLV - Hardware decoding support - Better performance for most files

- **vlc_source**: VLC-based media source
- Supports: All VLC-compatible formats
- Playlist support (multiple files)
- Better codec compatibility

Step 2: Source Name Enter a unique name for this source in OBS.

Rules: - Must be unique within the event - Only alphanumeric, spaces, hyphens, underscores - Example: `IntroVideo`, `Background Music`, `Main Content`

Tip: Use descriptive names that indicate the content (e.g., `MondayIntro`, `CommercialBlock1`)

Step 3: File Path Enter the absolute path to the media file on disk.

Format: - **Linux**: `/home/user/videos/intro.mp4` - **Windows**: `C:\Videos\intro.mp4` or `C:/Videos/intro.mp4` (both work) - Must be readable by the Scene Scheduler process - File must exist when event triggers (checked during staging)

Tips: - Use the file browser if available in your UI - Avoid spaces in file paths (use underscores: `my_video.mp4`) - Test preview before saving event

Step 4: Additional Settings

Loop (checkbox): - Enabled: Video repeats continuously during event duration - Disabled: Video plays once and stops

Use case for looping: - Background videos (e.g., animated backgrounds) - Short videos that need to fill long durations - Music tracks that repeat

Use case for non-looping: - One-time intro videos - News segments - Event-specific content that shouldn't repeat

Restart on activate (checkbox): - Enabled: Video restarts from beginning when scene activates - Disabled: Video continues from where it was

Hardware decoding (checkbox, media_source only): - Enabled: Uses GPU for video decoding (better performance) - Disabled: Uses CPU decoding

Recommended: Enable for high-resolution videos (1080p, 4K)

Step 5: Preview Before saving, click the **Preview** tab to test the media file (see Section 5.6).

Multiple Media Sources

You can add multiple media sources to a single event:

Example use case:

```
Event: Morning Show
├── BackgroundVideo (media_source, /videos/bg.mp4, loop=true)
├── IntroMusic (media_source, /audio/intro.mp3, loop=false)
└── Overlay (browser_source, https://overlay.com)
```

All sources are added to the scene simultaneously when the event triggers.

Media Source Limitations

- **File size**: Large files (>2GB) may have slow load times
- **Codec support**: Depends on OBS and system codecs (H.264 recommended)
- **Network paths**: SMB/NFS mounts may cause delays (use local copies)

5.3 Browser Tab

The Browser tab configures **browser_source** types—web pages, HTML overlays, and interactive web content rendered in OBS.

When to Use Browser Sources

- Dynamic overlays (chat, alerts, timers)
- Web-based graphics (HTML/CSS/JavaScript)
- Streaming dashboards
- Interactive visualizations
- Remote content (APIs, data feeds)

Adding a Browser Source

Step 1: Source Name Enter a unique name for the browser source.

Rules: Same as media sources (alphanumeric, unique within event)

Step 2: URL Enter the full URL to render.

Supported protocols: - `https://` - Secure web pages (recommended) - `http://` - Non-secure web pages - `file:///` - Local HTML files

Examples: - `https://example.com/overlay.html` - Remote hosted overlay - **Linux**: `file:///home/user/overlays/timer.html` - **Windows**: `file:///C:/overlays/timer.html` - `http://localhost:3000` - Local development server

Important: - URL must be accessible from the machine running OBS - HTTPS is recommended for security - Test URL in browser before adding to event

Step 3: Dimensions

Width and **Height** (pixels): - Defines the browser viewport size - Common sizes: - `1920 x 1080` - Full HD overlay - `1280 x 720` - HD overlay - `400 x 300` - Small widget - `800 x 100` - Ticker/banner

Why dimensions matter: - Affects how the page is rendered - Responsive designs adapt to these dimensions - Larger sizes consume more resources

Step 4: CSS (Optional) Custom CSS to inject into the page.

Use cases: - Hide specific elements: `#ads { display: none; }` - Override colors: `body { background: transparent; }` - Adjust positioning: `.widget { margin-top: 50px; }`

Example CSS:

```
css
body {
  background-color: transparent;
  margin: 0;
  padding: 0;
}
#header {
  display: none;
}
```

Step 5: Additional Settings

Shutdown when not visible (checkbox): - Enabled: Browser stops rendering when source is hidden (saves CPU/GPU) - Disabled: Browser continues rendering (use for animations that need to run continuously)

Refresh when scene becomes active (checkbox): - Enabled: Page reloads every time scene activates (resets state) - Disabled: Page persists across scene changes (maintains state)

FPS (frames per second): - Default: 30 - Range: 1-60 - Higher FPS = smoother animations, more CPU usage - Recommended: 30 for most overlays, 60 for smooth animations

Step 6: Preview Use the **Preview** tab to test the browser source before saving (see Section 5.6).

Browser Source Performance Tips

- **Optimize web pages**: Minimize JavaScript, compress assets
- **Use transparent backgrounds**: Set `background: transparent` in CSS
- **Limit animations**: Excessive animations can cause frame drops
- **Shutdown when not visible**: Enable to save resources

Browser Source Security

- **HTTPS only**: Avoid HTTP for sensitive data
- **Trust the source**: Only use URLs you control or trust
- **Local files**: Use `file:///` for controlled HTML overlays
- **No user input**: Browser sources in OBS don't handle user input

5.4 FFMPEG Tab

The FFMPEG tab configures **ffmpeg_source** types—network streaming inputs from RTMP, RTSP, RTP, SRT, HLS, and other protocols.

When to Use FFMPEG Sources

- IP cameras (RTSP streams)
- Remote contribution feeds (SRT, RTMP)
- Network video sources (NDI via FFMPEG)
- Live streaming inputs (HLS, RTMP pull)
- Professional broadcast equipment outputs

Adding an FFMPEG Source

Step 1: Source Name Enter a unique name for the FFMPEG source.

Examples: `Camera1`, `RemoteFeed`, `IP_Camera_Front`, `SRT_Input`

Step 2: Input URL Enter the streaming URL.

Supported protocols:

RTSP (IP Cameras):

```
rtsp://192.168.1.100:554/stream
rtsp://username:password@camera.local/live
```

RTMP (Streaming servers):

```
rtmp://server.example.com:1935/live/stream
rtmp://192.168.1.50/live/feed
```

SRT (Secure Reliable Transport):

```
srt://192.168.1.200:9000?mode=caller
srt://remote.server.com:9000?passphrase=secret
```

RTP (Real-time Protocol):

```
rtp://239.0.0.1:5004
```

HTTP/HLS:

```
https://stream.example.com/live/playlist.m3u8
http://192.168.1.100/stream.m3u8
```

File (for testing):

```
file:///home/user/test.mp4
```

Step 3: Input Format (Optional) Specify the container format if FFMPEG can't auto-detect.

Common formats: - `rtsp` - RTSP streams - `mpegts` - MPEG Transport Stream - `flv` - Flash Video - `mp4` - MP4 container - Leave empty for auto-detection (recommended)

Step 4: Additional Settings

Buffering (MB): - Default: 2 MB - Range: 1-10 MB - Higher buffering = more latency, more stable playback - Lower buffering = less latency, potential stuttering

Reconnect delay (seconds): - How long to wait before retrying connection after disconnect - Default: 5 seconds - Useful for unstable network streams

Hardware decoding (checkbox): - Enabled: Use GPU decoding (lower CPU usage) - Disabled: Use CPU decoding - Recommended for high-bitrate streams

Step 5: Preview Use the **Preview** tab to test connectivity before saving (see Section 5.6).

Important for FFMPEG sources: Preview verifies the stream is reachable and decodes properly. This catches connection issues before your event goes live.

FFMPEG Source Troubleshooting

Issue	Cause	Solution
Connection timeout	Network unreachable	Check IP, firewall, routing
Authentication failed	Wrong credentials	Verify username/password in URL
Protocol not supported	Missing codec	Install required FFMPEG libraries
Choppy playback	Network bandwidth	Increase buffering, check network quality
High latency	Large buffer	Reduce buffering value

FFMPEG Source Best Practices

1. **Test before production**: Use Preview tab to verify connectivity
2. **Use static IPs**: Avoid DHCP for critical sources
3. **Monitor bandwidth**: High-bitrate streams need adequate network capacity
4. **Enable reconnect**: Network streams can drop; auto-reconnect is essential
5. **Secure credentials**: Use environment variables or config files (don't hardcode passwords)

5.5 Common Source Configuration Patterns

Here are real-world examples of combining source types:

Pattern 1: News Show

```
Event: Morning News (08:00:00, duration 01:00:00)
├── Media: IntroVideo (/videos/news_intro.mp4, loop=false)
├── FFMPEG: LiveFeed (rtsp://camera.local/stream)
├── Browser: LowerThird (https://graphics.local/lowerthird.html)
└── Browser: TickerBar (https://graphics.local/ticker.html)
```

Pattern 2: Automated Playlist

```
Event: Music Videos (14:00:00, duration 02:00:00)
├── VLC: Playlist (/playlists/afternoon.xspf)
└── Browser: SongInfo (https://overlay.local/nowplaying.html)
```

Pattern 3: Live Stream Relay

```
Event: Remote Event (19:00:00, duration 03:00:00)
├── FFMPEG: MainFeed (srt://remote.server:9000?mode=caller)
├── FFMPEG: BackupFeed (rtmp://backup.server/live)
└── Browser: EventInfo (https://overlay.local/event_details.html)
```

Pattern 4: Looping Background

```
Event: Holding Screen (23:00:00, duration 08:00:00)
├── Media: BackgroundLoop (/videos/holding.mp4, loop=true)
└── Browser: Clock (file:///overlays/clock.html)
```

5.6 Preview Tab - Optional Testing Tool

The Preview tab is an **optional auxiliary feature** that allows you to test source configurations before committing them to your schedule. While Scene Scheduler's primary purpose is automated time-based scheduling, this preview tool helps catch configuration errors during setup.

Important: Preview is NOT required for Scene Scheduler operation. It's a convenience feature for the Editor View - the backend scheduler operates independently and doesn't use preview functionality.

5.6.1 Why Use Preview?

Benefits of this optional tool: 1. **Verify connectivity**: Test network streams before scheduling 2. **Check file paths**: Ensure media files exist and are readable 3. **Test visual appearance**: See how sources render before going live 4. **Catch errors early**: Identify issues during configuration, not during scheduled broadcast 5. **Save time**: No need to wait for event time to verify configuration

Common issues caught by preview: - Invalid file paths (typos, missing files) - Unreachable network streams (firewall, wrong IP) - Malformed URLs (syntax errors) - Broken browser sources (404 errors, CORS issues) - Codec problems (unsupported formats)

5.6.2 How Preview Works

Scene Scheduler's preview system uses **HLS (HTTP Live Streaming)** to generate a browser-playable stream of your source:

Technical flow: 1. User clicks "▶ Preview Source" button 2. Backend spawns `hls-generator` process with source configuration 3. `hls-generator` uses OBS libraries to: - Create a temporary OBS scene - Add the source to the scene - Encode to H.264 - Segment into HLS chunks (.ts files) - Generate playlist manifest (.m3u8) 4. Frontend polls for playlist availability (max 30 seconds) 5. Once ready, HLS.js player loads and plays the stream 6. Preview automatically stops after 30 seconds (resource cleanup)

Why HLS? - **Browser-native**: Works in all modern browsers without plugins - **Adaptive**: Handles varying network conditions - **Standard**: Industry-standard streaming protocol - **Efficient**: Low latency, small overhead

5.6.3 Using the Preview Tab

Step 1: Configure your source Before previewing, fill out the source configuration in the appropriate tab: - Media tab: Set source name and file path - Browser tab: Set source name, URL, and dimensions - FFMPEG tab: Set source name and input URL

Step 2: Switch to Preview Tab Click the **"Preview"** tab in the event modal.

Step 3: Select Source The Preview tab shows a dropdown with all configured sources for this event:

```
┌─────────────────────────────────────────────────────────────┐
│  Preview Source                                             │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Source: [IntroVideo ▼]                                     │
│                                                             │
│  ┌───────────────────────────────────────────────────────┐ │
│  │                                                       │ │
│  │              [▶ Preview Source]                      │ │
│  │                                                       │ │
│  └───────────────────────────────────────────────────────┘ │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Select the source you want to preview from the dropdown.

Step 4: Start Preview Click the **"▶ Preview Source"** button.

What happens: 1. Button text changes to show status: - " Starting preview..." (requesting from backend) - " Waiting for stream..." (waiting for HLS playlist) - Video player appears and starts playback 2. Video plays in the modal 3. Preview automatically stops after 30 seconds

Step 5: Observe Watch the video to verify: -   Correct content (right file/stream) -   Visual quality (resolution, bitrate) -   Audio (if applicable) -   No errors or artifacts

Step 6: Stop Preview (Optional) You can manually stop preview before the 30-second timeout: - Click the **"■ Stop Preview"** button - Or close the modal (cleanup happens automatically)

5.6.4 Preview Button States and Messages

The preview button changes appearance and text to indicate status:

State 1: Idle (Default)

```
┌─────────────────────────┐
│   ▶ Preview Source      │  (Blue background, clickable)
└─────────────────────────┘
```

Meaning: Ready to start preview. Click to begin.

State 2: Starting

```
┌─────────────────────────┐
│   Starting preview...   │  (Blue background, disabled)
└─────────────────────────┘
```

Meaning: Request sent to backend, waiting for response. **Duration:** 1-2 seconds typically.

State 3: Waiting for Stream

```
┌───────────────────────────────┐
│   Waiting for stream...       │  (Blue background, disabled)
└───────────────────────────────┘
```

Meaning: Backend is generating HLS stream, frontend is polling for playlist. **Duration:** 5-30 seconds depending on source type. **Timeout:** 30 seconds max. If stream doesn't start, see troubleshooting below.

State 4: Playing

```
┌────────────────────────┐
│   ■ Stop Preview       │  (Red background, clickable)
└────────────────────────┘
```

Meaning: Stream is playing. Video visible below button. **Action:** Click to stop early (or wait for 30-second auto-stop).

State 5: Error

```
┌──────────────────────────────────────────┐
│ ⚠ Error: Connection timeout              │  (Amber background, disabled)
└──────────────────────────────────────────┘
```

Meaning: Preview failed. Error message explains why. **Duration:** Message shown for 5 seconds, then resets to idle state. **Action:** Fix the error (see error messages below) and retry.

State 6: Stopped (Auto or Manual)

```
┌───────────────────────────────────────────────────────┐
│ i Preview automatically stopped after 30 seconds      │  (Sky blue, disabled)
└───────────────────────────────────────────────────────┘
```

Meaning: Preview completed normally (30-second timeout reached). **Duration:** Message shown for 5 seconds, then resets to idle state. **Action:** None required. Click again to re-preview.

5.6.5 Preview Error Messages

When preview fails, the button displays an error message with specific details:

Error: "Connection timeout"

Full message: `⚠ Error: Connection timeout`

Cause: Backend couldn't reach the source within 30 seconds.

Common reasons: - FFMPEG source: Network stream is unreachable (wrong IP, firewall blocking) - Browser source: URL doesn't respond (server down, DNS failure) - Media source: File system is slow (network mount delay)

Solutions: 1. **Network streams**: Check IP address, port, and firewall rules 2. **Browser sources**: Test URL in a regular browser 3. **Media files**: Verify file path and permissions 4. **Network issues**: Check connectivity with `ping` or `curl`

Error: "File not found"

Full message: `⚠ Error: File not found: /path/to/file.mp4`

Cause: Media source file doesn't exist at the specified path.

Solutions: 1. Check file path for typos 2. Verify file exists: `ls -la /path/to/file.mp4` 3. Check permissions: File must be readable by Scene Scheduler process 4. Use absolute paths (not relative)

Error: "Invalid URL"

Full message: `⚠ Error: Invalid URL format`

Cause: Browser source or FFMPEG source URL is malformed.

Common issues: - Missing protocol: Use `https://example.com`, not `example.com` - Invalid characters: URL encoding needed for special chars - Wrong protocol: Use `rtsp://` for RTSP, not `http://`

Solutions: 1. Check URL syntax 2. Test URL in browser or media player 3. URL-encode special characters 4. Verify protocol matches source type

Error: "Stream failed to start"

Full message: `⚠ Error: Stream failed to start`

Cause: `hls-generator` process crashed or failed to initialize.

Common reasons: - Unsupported codec (source uses codec OBS can't decode) - Corrupted media file - OBS libraries missing or misconfigured

Solutions: 1. Check `hls-generator` logs (if available) 2. Test file in OBS directly 3. Re-encode media file to H.264/AAC 4. Verify OBS libraries are installed

Error: "Browser source load error"

Full message: `⚠ Error: Browser source failed to load`

Cause: Browser source URL returned an error (404, 500, etc.) or failed to render.

Common reasons: - 404 Not Found (wrong URL path) - CORS errors (cross-origin restrictions) - JavaScript errors in page - SSL certificate errors (HTTPS)

Solutions: 1. Open URL in regular browser, check console for errors 2. Verify URL is publicly accessible (or locally reachable) 3. Check server logs for errors 4. For HTTPS: Ensure valid SSL certificate

Error: "Preview already running"

Full message: `⚠ Error: Preview already in progress`

Cause: Attempted to start a second preview while one is already active.

Solution: Wait for current preview to finish (30 seconds) or manually stop it first.

Error: "Authentication required"

Full message: `⚠ Error: Authentication required`

Cause: Network stream (RTSP, RTMP) requires credentials that weren't provided or are incorrect.

Solutions: 1. Include credentials in URL: - RTSP: `rtsp://username:password@camera.local/stream` - RTMP: `rtmp://username:password@server.local/live` 2. Verify credentials are correct 3. Check camera/server authentication settings

5.6.6 Preview Timeout (30 seconds)

All previews automatically stop after **30 seconds** to prevent resource exhaustion.

Why 30 seconds? - **Resource management**: Each preview consumes CPU/GPU (encoding) and disk space (HLS segments) - **Sufficient for testing**: 30 seconds is enough to verify source functionality - **Prevents forgotten previews**: Users might leave modal open; auto-stop ensures cleanup

What happens at timeout: 1. Frontend receives `previewStopped` WebSocket message 2. HLS.js player is destroyed gracefully 3. Video element is cleared 4. Button shows info message: "**i** Preview automatically stopped after 30 seconds" 5. Message auto-clears after 5 seconds 6. Button returns to idle state 7. Backend cleans up: - Kills `hls-generator` process - Deletes temporary HLS files - Releases resources

Want to preview longer? Click "▶ Preview Source" again after timeout to restart the preview.

5.6.7 Previewing Multiple Sources

If your event has multiple sources (e.g., background video + overlay + camera feed), preview each individually:

Workflow: 1. Configure all sources in their respective tabs 2. Switch to Preview tab 3. Select first source from dropdown 4. Click "▶ Preview Source", watch it play for 30 seconds 5. After timeout (or manual stop), select next source from dropdown 6. Repeat until all sources are verified

Why individual preview? - **Isolation**: Test each source independently - **Troubleshooting**: Identify which specific source has issues - **Performance**: Generating multiple previews simultaneously is resource-intensive

Note: Preview shows sources **individually**, not combined. To see all sources together as they'll appear in OBS, save the event and trigger it manually (set time 2 minutes in future).

5.6.8 Preview Performance Considerations

Preview generation is **resource-intensive**:

CPU/GPU usage: - Encoding to H.264 requires CPU or GPU - Browser source rendering uses GPU (CEF chromium) - Multiple previews compound resource usage

Disk usage: - Each preview generates 30 seconds of HLS segments - Typical size: 5-15 MB per preview - Automatic cleanup after preview stops

Network usage: - FFMPEG sources download the network stream - Browser sources fetch remote content - HLS segments are served over local HTTP

Best practices: 1. **Preview one source at a time**: Don't run multiple previews in parallel 2. **Close modal when done**: Releases resources immediately 3. **Use preview sparingly**: Only when configuring new sources 4. **Trust working configs**: Once a source is verified, no need to preview every time

5.6.9 Preview Troubleshooting Checklist

If preview fails or behaves unexpectedly, work through this checklist:

Backend connectivity - Is Scene Scheduler running? (check logs) - Is WebSocket connected? (check connection indicator in UI) - Can you create events and see them in the schedule? (verify basic functionality)

Source configuration - Is source name filled in? - Is file path/URL correct? (no typos) - For media sources: Does file exist? `ls -la /path/to/file.mp4` - For network sources: Is source reachable? `ping <ip>` or `curl <url>`

hls-generator binary - Does `hls-generator` exist in paths.hlsGenerator location? - Is it executable? `chmod +x hls-generator` - Does it run standalone? `./hls-generator --help`

HLS output directory - Does `webServer.hlsPath` directory exist? - Is it writable? `touch hls/test.txt && rm hls/test.txt` - Check disk space: `df -h`

Browser/network - Try in different browser (rule out browser-specific issues) - Check browser console for JavaScript errors (F12 → Console) - Disable browser extensions that might block video playback - Check network: Can browser reach backend? (try http://localhost:8080)

Logs - Check Scene Scheduler logs for preview-related errors - Look for messages containing "preview", "hls-generator", or "sourcepreview"

If all checks pass and preview still fails, see Section 10 (Troubleshooting) for advanced diagnostics.

5.6.10 Preview vs. Production Behavior

Important: Preview shows a **close approximation** of how sources will appear in OBS, but there are subtle differences:

Similarities: - Same source content (file, URL, stream) - Same decoding (OBS libraries) - Same video/audio output - Verifies connectivity and file existence

Differences: - Preview uses isolated OBS instance (not your main OBS) - Preview doesn't show source positioning/cropping (these are scene-level settings) - Preview doesn't show filters or effects (applied in OBS, not at source level) - Preview uses HLS encoding (slight quality loss vs. OBS direct output) - Preview latency is higher (HLS segmenting adds 3-6 seconds)

What this means: - **Use preview for**: Verifying source works, content is correct, connectivity is good - **Don't rely on preview for**: Exact color grading, precise timing, filter effects, final positioning

Final validation: After saving your event and before production use, trigger it manually in OBS (set time to 2 minutes in future) and observe the full scene composition.

5.6.11 Advanced: Preview of browser_source with CEF

Browser sources require **CEF (Chromium Embedded Framework)** to render:

How it works: 1. `hls-generator` initializes CEF 2. CEF loads the specified URL in a headless browser 3. Page JavaScript/CSS executes 4. Rendered frames are captured and encoded to H.264 5. HLS segments generated from encoded stream

Special considerations for browser_source preview:

Longer startup time: - CEF initialization: 2-5 seconds - Page loading (JavaScript, assets): 2-10 seconds - **Total:** 5-15 seconds before stream starts - Be patient with "Waiting for stream..." status

Transparency: - Preview shows transparency as **black background** - In OBS, transparency is respected (overlay shows over underlying sources) - Don't worry if preview background is black

Interactive elements: - Mouse/keyboard input doesn't work in preview (CEF is headless) - Animations and timers work normally - WebSocket/API calls work (if page uses them)

Resource usage: - CEF is memory-intensive (200-500 MB per instance) - GPU usage can be high for complex pages - Limit browser_source previews to avoid system overload

Debugging browser_source issues: 1. Open URL in regular Chrome/Chromium (check for JavaScript errors) 2. Use simple test page first (ensure CEF works): `file:///home/user/test.html` 3. Check CEF logs (if enabled in config) 4. Verify OBS browser source plugin is installed (CEF dependency)

---

6. System Configuration

Scene Scheduler is configured through `config.json`, located in the application directory. This file controls OBS connectivity, web server settings, scheduling behavior, and file paths.

6.1 Configuration File Location

Default location:

```
/path/to/scenescheduler/config.json
```

The configuration file must be in the same directory as the `scenescheduler` executable. If the file doesn't exist, create it with the template below.

6.2 Complete Configuration Template

Here's a fully commented `config.json` template:

```
json
{
  "obsWebSocket": {
    "host": "localhost",
    "port": 4455,
    "password": "your-obs-websocket-password"
  },
  "webServer": {
    "host": "0.0.0.0",
    "port": 8080,
    "hlsPath": "hls"
  },
  "schedule": {
    "jsonPath": "schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "./hls-generator"
  },
  "logging": {
    "level": "info",
    "format": "text"
  }
}
```

6.3 Configuration Sections

6.3.1 OBS WebSocket Configuration (`obsWebSocket`)

Controls connection to OBS Studio via the WebSocket protocol.

`host` (string, required) - OBS Studio hostname or IP address - **Default:** `"localhost"` (OBS on same machine) - **Examples:** - `"localhost"` - OBS on local machine - `"192.168.1.100"` - OBS on remote machine (same network) - `"obs.local"` - OBS via hostname (requires DNS/mDNS)

`port` (integer, required) - WebSocket server port - **Default:** `4455` (OBS default) - **Range:** 1024-65535 - **Note:** Must match port configured in OBS (Tools → WebSocket Server Settings)

`password` (string, optional) - WebSocket authentication password - **Default:** `""` (empty = no authentication) - **Security:** Highly recommended for production use - **Must match:** Password set in OBS WebSocket settings

Example configurations:

Local OBS, no password:

```
json
"obsWebSocket": {
  "host": "localhost",
  "port": 4455,
  "password": ""
}
```

Local OBS, with password:

```
json
"obsWebSocket": {
  "host": "localhost",
  "port": 4455,
  "password": "s3cur3p@ssw0rd"
}
```

Remote OBS:

```
json
"obsWebSocket": {
  "host": "192.168.1.50",
  "port": 4455,
  "password": "remote_password"
}
```

Troubleshooting OBS connection: - **Connection refused**: Check OBS is running and WebSocket server is enabled - **Authentication failed**: Verify password matches OBS settings - **Host unreachable**: Check firewall, network routing (ping `<host>`)

6.3.2 Web Server Configuration (`webServer`)

Controls the HTTP server that hosts the web interface and HLS preview streams.

`host` (string, required) - Network interface to bind to - **Options:** - `"0.0.0.0"` - Bind to all interfaces (accessible from network) - `"localhost"` or `"127.0.0.1"` - Local only (most secure) - Specific IP - Bind to specific interface (e.g., `"192.168.1.10"`)

When to use each: - **`0.0.0.0`**: Multi-device access (recommended for production) - **`localhost`**: Testing, single-user, security-sensitive environments

`port` (integer, required) - HTTP server port - **Default:** `8080` - **Range:** 1024-65535 (avoid 80/443 unless running as root) - **Note:** Must be free (not used by other applications)

`hlsPath` (string, required) - Directory for HLS preview files (relative to executable) - **Default:** `"hls"` - **Important:** This is the **correct** configuration field (NOT `paths.hlsBase`) - **Directory must:** - Exist (create with `mkdir hls`) - Be writable by Scene Scheduler process - Have sufficient disk space (each preview uses 5-15 MB)

Example configurations:

Network-accessible (recommended):

```
json
"webServer": {
  "host": "0.0.0.0",
  "port": 8080,
  "hlsPath": "hls"
}
```

Local-only (secure):

```
json
"webServer": {
  "host": "localhost",
  "port": 8080,
  "hlsPath": "hls"
}
```

Custom port:

```
json
"webServer": {
  "host": "0.0.0.0",
  "port": 3000,
  "hlsPath": "preview_files"
}
```

Accessing the web interface: - Local: `http://localhost:8080` - From network: `http://<server-ip>:8080` (e.g., `http://192.168.1.100:8080`)

Troubleshooting web server: - **Port already in use**: Change `port` to unused value (e.g., 8081, 3000) - **Cannot access from network**: Check `host` is `0.0.0.0`, verify firewall allows port - **Permission denied (port 80)**: Use port ≥1024 or run as root (not recommended)

6.3.3 Schedule Configuration (`schedule`)

Controls schedule file location and scene naming.

`jsonPath` (string, required) - Path to schedule JSON file (relative to executable) - **Default:** `"schedule.json"` - **Format:** JSON array of events (see Section 11.1 for schema) - **Permissions:** Must be readable and writable - **Backup:** Recommended to keep backups of this file

`scheduleSceneAux` (string, required) - Name of the auxiliary OBS scene used for staging - **Default:** `"scheduleSceneAux"` - **Auto-creation:** Scene Scheduler automatically creates this scene if it doesn't exist - **Purpose:** Hidden scene where sources are preloaded before transitions

Important notes: - The auxiliary scene name is case-sensitive - The scene should remain empty (Scene Scheduler manages its content automatically) - Don't delete this scene while Scene Scheduler is running - If you change the scene name in config.json, Scene Scheduler will create a new scene with that name

Example configurations:

Default:

```
json
"schedule": {
  "jsonPath": "schedule.json",
  "scheduleSceneAux": "scheduleSceneAux"
}
```

Custom schedule file location:

```
json
"schedule": {
  "jsonPath": "/var/lib/scenescheduler/production_schedule.json",
  "scheduleSceneAux": "scheduleSceneAux"
}
```

Custom auxiliary scene name:

```
json
"schedule": {
  "jsonPath": "schedule.json",
  "scheduleSceneAux": "staging_scene"
}
```

Troubleshooting schedule: - **Schedule not loading**: Check `jsonPath` file exists and is valid JSON - **Scene not found error**: Check that scheduleSceneAux name in config.json matches (Scene Scheduler creates it automatically) - **Permission denied**: Ensure Scene Scheduler can read/write schedule file

6.3.4 Paths Configuration (`paths`)

Controls locations of external binaries and tools.

`hlsGenerator` (string, required) - Path to `hls-generator` executable (relative to main executable) - **Default:** `"./hls-generator"` - **Purpose:** Generates HLS preview streams - **Requirements:** - Must exist and be executable (`chmod +x hls-generator`) - Must be compatible with your system (Linux x86_64) - Must have OBS libraries available

Example configurations:

Default (same directory):

```
json
"paths": {
  "hlsGenerator": "./hls-generator"
}
```

Absolute path:

```
json
"paths": {
  "hlsGenerator": "/usr/local/bin/hls-generator"
}
```

Subdirectory:

```
json
"paths": {
  "hlsGenerator": "./bin/hls-generator"
}
```

Troubleshooting hls-generator: - **File not found**: Verify file exists at specified path - **Permission denied**: Run `chmod +x hls-generator` - **Exec format error**: Binary not compatible with your system (wrong architecture)

6.3.5 Logging Configuration (`logging`)

Controls application logging behavior.

`level` (string, optional) - Log verbosity level - **Options:** `"debug"`, `"info"`, `"warn"`, `"error"` - **Default:** `"info"` - **Recommendation:** - Production: `"info"` or `"warn"` - Debugging: `"debug"` - Critical only: `"error"`

`format` (string, optional) - Log output format - **Options:** - `"text"` - Human-readable (default) - `"json"` - Machine-parseable (for log aggregation tools) - **Default:** `"text"`

Example configurations:

Production (default):

```
json
"logging": {
  "level": "info",
  "format": "text"
}
```

Debugging:

```
json
"logging": {
  "level": "debug",
  "format": "text"
}
```

Log aggregation:

```
json
"logging": {
  "level": "info",
  "format": "json"
}
```

Minimal logging:

```
json
"logging": {
  "level": "error",
  "format": "text"
}
```

Log levels explained: - **debug**: All messages (very verbose, includes internal state changes) - **info**: General information (startup, connections, event triggers) - **warn**: Warnings (non-critical issues, deprecated features) - **error**: Errors only (failures, exceptions)

6.4 Environment Variables

Some settings can be overridden with environment variables (useful for Docker, systemd):

`OBS_WS_HOST` - Override `obsWebSocket.host`

```
bash
export OBS_WS_HOST="192.168.1.50"
./scenescheduler
```

`OBS_WS_PORT` - Override `obsWebSocket.port`

```
bash
export OBS_WS_PORT="4456"
./scenescheduler
```

`OBS_WS_PASSWORD` - Override `obsWebSocket.password` (recommended for security)

```
bash
export OBS_WS_PASSWORD="s3cur3p@ss"
./scenescheduler
```

`WEB_SERVER_PORT` - Override `webServer.port`

```
bash
export WEB_SERVER_PORT="3000"
./scenescheduler
```

Priority: Environment variables > config.json > defaults

6.5 Configuration Validation

Scene Scheduler validates configuration on startup:

Validation checks: 1. Config file exists and is valid JSON 2. Required fields are present 3. Port numbers are in valid range (1-65535) 4. File paths are accessible 5. HLS directory exists and is writable

Startup behavior: - **Valid config**: Application starts normally - **Invalid config**: Error logged and application exits - **Missing config**: Uses defaults (may fail if OBS requires password)

Example validation errors:

Invalid JSON:

```
FATAL: Failed to parse config.json: invalid character '}' looking for beginning of objec
```

Solution: Fix JSON syntax (check for missing commas, quotes)

Missing required field:

```
FATAL: Missing required config field: obsWebSocket.host
```

Solution: Add missing field to config.json

Invalid port:

```
FATAL: Invalid port number: 99999 (must be 1-65535)
```

Solution: Use valid port number

6.6 Configuration Examples for Common Scenarios

Scenario 1: Single Computer Setup

Both OBS and Scene Scheduler on the same machine, accessed locally only.

```
json
{
  "obsWebSocket": {
    "host": "localhost",
    "port": 4455,
    "password": ""
  },
  "webServer": {
    "host": "localhost",
    "port": 8080,
    "hlsPath": "hls"
  },
  "schedule": {
    "jsonPath": "schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "./hls-generator"
  }
}
```

Access: `http://localhost:8080`

Scenario 2: Production Server (Network Access)

Scene Scheduler accessible from multiple devices on the network.

```
json
{
  "obsWebSocket": {
    "host": "localhost",
    "port": 4455,
    "password": "production_password_123"
  },
  "webServer": {
    "host": "0.0.0.0",
    "port": 8080,
    "hlsPath": "hls"
  },
  "schedule": {
    "jsonPath": "/var/lib/scenescheduler/schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "/opt/scenescheduler/hls-generator"
  },
  "logging": {
    "level": "info",
    "format": "text"
  }
}
```

Access: `http://192.168.1.100:8080` (use server's IP)

Scenario 3: Remote OBS Control

Scene Scheduler on a different machine than OBS.

```
json
{
  "obsWebSocket": {
    "host": "192.168.1.50",
    "port": 4455,
    "password": "obs_remote_password"
  },
  "webServer": {
    "host": "0.0.0.0",
    "port": 8080,
    "hlsPath": "hls"
  },
  "schedule": {
    "jsonPath": "schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "./hls-generator"
  }
}
```

Requirements: - OBS machine (192.168.1.50) must have WebSocket enabled - Firewall must allow port 4455 - Both machines on same network (or VPN)

Scenario 4: Docker Deployment

Using environment variables for dynamic configuration.

config.json (minimal):

```
json
{
  "schedule": {
    "jsonPath": "/data/schedule.json",
    "scheduleSceneAux": "scheduleSceneAux"
  },
  "paths": {
    "hlsGenerator": "/app/hls-generator"
  }
}
```

Docker run command:

```
bash
docker run -d \
  -e OBS_WS_HOST=192.168.1.50 \
  -e OBS_WS_PORT=4455 \
  -e OBS_WS_PASSWORD=secure_password \
  -e WEB_SERVER_PORT=8080 \
  -v /path/to/schedule.json:/data/schedule.json \
  -p 8080:8080 \
  scenescheduler:latest
```

6.7 Security Considerations

OBS WebSocket Password: - Always set a password in production - Use strong, unique passwords (16+ characters) - Don't commit passwords to version control - Use environment variables for sensitive values

Web Server Access: - Use `host: "localhost"` if network access not needed - Configure firewall to restrict web server port access - Don't expose to public internet without authentication - Consider reverse proxy (nginx) with HTTPS for production

File Permissions: - Schedule file: `chmod 600 schedule.json` (owner read/write only) - Config file: `chmod 600 config.json` - HLS directory: `chmod 700 hls/`

Backup Strategy: - Regular backups of `schedule.json` - Store backups securely (encrypted if sensitive) - Test restore procedure periodically

6.8 Updating Configuration

Without restarting: - Schedule changes (`schedule.json`) apply automatically (hot-reload) - Source configuration changes apply on next event trigger

Requires restart: - OBS WebSocket settings - Web server host/port - File paths - Logging configuration

How to restart:

Linux:

```
bash
# Stop
pkill scenescheduler

# Restart
./scenescheduler
```

Or with systemd:

```
bash
sudo systemctl restart scenescheduler
```

Windows:

```
cmd
REM Stop: Press Ctrl+C in the command prompt window
REM Or: Close the command prompt window
REM Or: Use Task Manager to end scenescheduler.exe

REM Restart
scenescheduler.exe
```

Configuration change checklist: 1. Edit `config.json` with a text editor 2. Validate JSON syntax (use `jq . config.json` or online validator) 3. Back up previous config (optional but recommended) 4. Restart Scene Scheduler if needed 5. Verify connection (check web interface, OBS connection indicator) 6. Test functionality (create test event)

7. How It Works Internally

Understanding Scene Scheduler's internal mechanisms helps you optimize configurations, troubleshoot issues, and predict behavior during complex scenarios.

7.1 Staging System

One of Scene Scheduler's most important features is the **staging system**, which ensures smooth, seamless transitions without visible loading delays.

Why Staging Exists

Without staging, scene transitions would show loading delays:

```
Event triggers → OBS switches scene → Sources load → User sees buffering
```

With staging, sources are prepared in advance:

```
Event triggers → Staging process (5 steps from Section 2.3) → Instant transition
```

The staging system uses the `scheduleSceneAux` auxiliary scene as a backstage area where sources are created and initialized before being moved to the visible scene.

Staging Process Overview

When an event time arrives, Scene Scheduler executes the 5-step process described in Section 2.3:

1. **STAGING**: Sources created in scheduleSceneAux (invisible to viewers)
2. **ACTIVATION**: Sources moved to target scene
3. **SCENE SWITCH**: OBS transitions to target scene
4. **CLEANUP**: Temporary elements removed from scheduleSceneAux
5. **MONITOR**: Event runs for its configured duration

Benefits

- **No visible loading**: Sources prepared before shown to viewers
- **Atomic transitions**: Either complete success or safe rollback
- **Resource efficiency**: Cleanup prevents memory leaks and orphaned sources
- **Continuous operation**: System handles 24/7 automated scheduling

Staging Optimization Tips

1. **Use local files when possible**: Network-mounted files may add latency
2. **Optimize browser sources**: Keep pages simple, minimize JavaScript
3. **Test network streams**: Verify connectivity before scheduling
4. **Monitor resource usage**: Check CPU/GPU during transitions

7.2 EventBus System (Real-Time Synchronization)

Scene Scheduler uses an **EventBus** architecture to synchronize state across all components:

```
┌─────────────────────────────────────────────────────────────┐
│                         EventBus                            │
│                                                             │
│  ┌─────────────┐   ┌─────────────┐   ┌─────────────┐       │
│  │  Scheduler  │   │   OBS WS    │   │  WebSocket  │       │
│  │   Engine    │   │ Connection  │   │   Clients   │       │
│  └─────────────┘   └─────────────┘   └─────────────┘       │
│         │                  │                  │             │
│         ▼                  ▼                  ▼             │
│  ┌─────────────────────────────────────────────────┐       │
│  │              Central Event Stream               │       │
│  └─────────────────────────────────────────────────┘       │
└─────────────────────────────────────────────────────────────┘
```

Event Types

Schedule Events: - `scheduleUpdated`: Triggered when schedule.json changes - `eventTriggered`: An event's time has arrived - `eventCompleted`: An event's duration has expired

OBS Events: - `obsConnected`: WebSocket connection established - `obsDisconnected`: WebSocket connection lost - `sceneChanged`: OBS switched scenes (manual or automatic) - `sourceCreated`: A source was added to OBS - `sourceRemoved`: A source was deleted from OBS

Frontend Events: - `clientConnected`: Web browser connected via WebSocket - `clientDisconnected`: Web browser closed connection - `scheduleRequest`: Client requested current schedule - `previewRequest`: Client requested source preview

Event Flow Example: Adding an Event

```
1. User clicks "Save Event" in web interface
   └→ Frontend sends WebSocket message: { type: "addEvent", payload: {...} }

2. Backend receives message
   └→ Validates event data
   └→ Appends to schedule.json
   └→ EventBus emits: scheduleUpdated

3. All subscribed components react:
   ├→ Scheduler Engine: Recalculates next trigger time
   ├→ WebSocket Handler: Broadcasts update to all clients
   └→ File Watcher: Updates in-memory schedule

4. All connected web clients receive update
   └→ Monitor View: Updates event list
   └→ Editor View: Shows new event in list
```

Why EventBus Matters

Benefits: - **Loose coupling**: Components don't directly depend on each other - **Real-time sync**: All clients instantly reflect changes - **Extensibility**: New features can subscribe to events without modifying existing code - **Debugging**: Event log shows complete system activity

7.3 OBS WebSocket Communication

Scene Scheduler communicates with OBS through the **OBS WebSocket protocol (v5.x)**.

Connection Lifecycle

Phase 1: Initial Connection

```
1. Scene Scheduler starts
2. WebSocket client attempts connection to OBS
3. If password required, authentication challenge occurs
4. Connection established, identified message received
5. Scene Scheduler subscribes to OBS events
```

Phase 2: Steady State

```
- Heartbeat messages every 10 seconds (keepalive)
- Scene Scheduler sends requests (GetSceneList, CreateInput, etc.)
- OBS sends responses and event notifications
```

Phase 3: Reconnection

```
If connection lost:
1. Scene Scheduler detects disconnect
2. Automatic reconnection attempts begin (exponential backoff)
3. Retry intervals: 1s, 2s, 4s, 8s, 16s, max 30s
4. On reconnect: Re-sync state, re-subscribe to events
```

Key OBS WebSocket Operations

Scene Management: - `GetSceneList`: Retrieve all scenes (for dropdown population) - `SetCurrentProgramScene`: Switch to a scene - `GetCurrentProgramScene`: Query active scene

Source Management: - `CreateInput`: Add a new source to a scene - `RemoveInput`: Delete a source - `SetInputSettings`: Configure source properties - `GetInputSettings`: Query source configuration

Scene Item Management: - `GetSceneItemList`: List sources in a scene - `SetSceneItemEnabled`: Show/hide a source - `SetSceneItemTransform`: Position, scale, crop sources

Error Handling

Connection errors: - **Authentication failure**: Log error, exit (requires manual config fix) - **Host unreachable**: Retry with exponential backoff - **Protocol mismatch**: Log error, exit (OBS version incompatible)

Command errors: - **Scene not found**: Log warning, skip transition - **Source creation failed**: Log error, continue with other sources - **Timeout**: Retry once, then log error and continue

7.4 Schedule Hot-Reload

Scene Scheduler watches `schedule.json` for changes and automatically reloads without restarting:

How Hot-Reload Works

File Watching:

```
1. On startup, Scene Scheduler begins watching schedule.json
2. File system events (write, modify) trigger reload
3. Debouncing prevents multiple reloads (waits 500ms after last change)
```

Reload Process:

```
1. Detect file change
2. Read updated schedule.json
3. Parse and validate JSON
4. If valid:
   ├→ Replace in-memory schedule
   ├→ Recalculate next event trigger time
   ├→ Broadcast update to all connected clients
   └→ Log: "Schedule reloaded (N events)"
5. If invalid:
   ├→ Keep previous schedule (don't break running system)
   ├→ Log error with JSON parse details
   └→ Notify clients of error
```

What Triggers Reload

Automatic triggers: - Frontend saves event (add, edit, delete) - External editor modifies schedule.json (vim, nano, etc.) - Script writes updated schedule (automation)

Does NOT trigger reload: - Config.json changes (requires restart) - Source file changes (affects next staging)

Reload Edge Cases

During staging (event trigger in progress): - Reload occurs immediately - Currently staging event continues with **old configuration** - Next event uses new configuration

During active event (event currently running): - Reload occurs immediately - Active event continues with old configuration - Sources remain as originally configured - Cleanup happens at original duration

Recommendation: Avoid editing events that are actively running. Edit future events instead.

7.5 Preview Process Lifecycle

Understanding the preview system's internal flow helps troubleshoot issues:

Preview Request Flow

Phase 1: Request Initiated (Frontend)

```
1. User clicks "▶ Preview Source" button
2. Frontend gathers source configuration (name, type, URL/path, settings)
3. WebSocket message sent: { type: "startPreview", payload: {...} }
4. Button state: "  Starting preview..."
```

Phase 2: Backend Processing

```
1. Backend receives startPreview message
2. Validates source configuration
3. Generates unique preview ID and connection ID
4. Creates preview session in memory
5. Spawns hls-generator subprocess with source config
6. Response sent to frontend: { previewID, hlsURL }
```

Phase 3: HLS Generation (hls-generator process)

```
1. hls-generator initializes OBS libraries
2. Creates temporary OBS scene
3. Adds source to scene (media/browser/ffmpeg)
4. Source loads and initializes:
   - Media: File opened and buffered
   - Browser: CEF starts, loads URL
   - FFMPEG: Network connection established
5. Encoding begins (H.264 + AAC)
6. HLS segments generated (.ts files)
7. Playlist manifest written (.m3u8)
8. First segment verification (checks for #EXTINF tag)
```

Phase 4: Frontend Polling

```
1. Frontend polls for playlist: GET /hls/{previewID}/playlist.m3u8
2. Retry intervals: 500ms, 1s, 2s, 4s, 8s (exponential backoff)
```

```
3. Timeout: 30 seconds max
4. Button state: "  Waiting for stream..."
```

Phase 5: Playback Begins

```
1. Playlist found, HLS.js initializes
2. HLS.js downloads segments and plays
3. Video element shows stream
4. Button state: "■ Stop Preview"
5. 30-second timeout timer starts
```

Phase 6: Cleanup (Auto or Manual)

```
1. Timeout reached (30s) OR user clicks "Stop Preview"
2. Backend sends previewStopped WebSocket message
3. Frontend gracefully destroys HLS.js (prevents 404 errors)
4. Video element cleared
5. Backend cleanup:
   ├→ Kill hls-generator process (SIGTERM)
   ├→ Wait 5s for graceful shutdown
   ├→ Force kill if still running (SIGKILL)
   ├→ Delete HLS directory and all segments
   └→ Remove preview session from memory
6. Button state: "ℹ Preview automatically stopped after 30 seconds"
7. After 5s, button resets to idle state
```

Preview Connection Tracking

Scene Scheduler v1.6 uses **connection IDs** (not IP addresses) to track preview sessions:

Why connection IDs? -   NAT-safe (multiple clients behind same NAT have unique IDs) -   Secure (no IP address exposure in logs) -   Reliable (survives network changes)

Connection ID lifecycle:

```
1. WebSocket connection established
2. Unique ID generated: "conn_<timestamp>_<random>"
3. ID associated with preview session
4. On disconnect: All previews for that connection are cleaned up
5. Prevents orphaned preview processes
```

7.6 Cleanup and Resource Management

Scene Scheduler implements thorough cleanup to prevent resource leaks:

Event Cleanup (After Duration Expires)

```
1. Event duration timer fires
2. Cleanup sequence begins:
   ├→ Get list of dynamic sources created by this event
   ├→ For each source:
   │  ├→ Check if still exists in OBS
   │  ├→ Remove from target scene
   │  └→ Delete source from OBS
   ├→ Clear scheduleSceneAux scene
   └→ Log: "Event cleanup completed"
3. System ready for next event
```

Idempotent cleanup: All cleanup operations are safe to call multiple times: - `delete()` on non-existent map key: no-op - `os.RemoveAll()` on missing directory: no error - OBS source deletion of missing source: ignored

Preview Cleanup

```
1. Preview timeout (30s) or manual stop
2. Cleanup sequence:
   ├→ Stop timeout timer (if running)
   ├→ Send previewStopped WebSocket message
   ├→ Wait 100ms (ensure message delivered)
   ├→ Kill hls-generator process:
   │  ├→ Send SIGTERM (graceful shutdown)
   │  ├→ Wait 5 seconds
   │  └→ Send SIGKILL if still running (force)
   ├→ Delete HLS directory: rm -rf hls/{previewID}/
   ├→ Remove preview session from memory map
   └→ Log: "Preview cleanup completed"
```

WebSocket Client Cleanup

```
When client disconnects:
1. WebSocket handler detects connection close
2. Get connection ID
3. Check for active previews with this connection ID
4. For each preview:
   └→ Run full preview cleanup
```

```
5. Remove client from broadcast list
6. Log: "Client disconnected, cleaned up N previews"
```

Application Shutdown Cleanup

```
When Scene Scheduler exits (SIGTERM/SIGINT):
1. Signal handler triggered
2. Graceful shutdown sequence:
   ├→ Stop accepting new requests
   ├→ Stop all active previews
   ├→ Clean up all HLS directories
   ├→ Stop all staging processes
   ├→ Close OBS WebSocket connection
   ├→ Close web server
   └→ Flush logs
3. Exit with code 0
```

7.7 State Synchronization

Scene Scheduler maintains consistency across multiple components:

Initial State Synchronization (Client Connect)

```
When web client connects:
1. WebSocket connection established
2. Client sends: { type: "getInitialState" }
3. Backend responds with:
   ├→ Current schedule (all events)
   ├→ Active event (if any)
   ├→ OBS connection status
   ├→ Available OBS scenes
   └→ Current OBS scene
4. Client renders UI with this state
```

Ongoing State Sync

Schedule changes:

```
User adds event → Backend updates schedule.json → EventBus emits scheduleUpdated → All c
```

OBS state changes:

```
Manual scene change in OBS → WebSocket event received → EventBus emits sceneChanged → Al
```

Preview state:

```
Preview starts → WebSocket message to requesting client only → Button state updates to "
Preview stops → WebSocket message to requesting client only → Button resets to idle
```

Conflict Resolution

Multiple clients editing simultaneously: - Last write wins (no optimistic locking) - All clients receive final state via broadcast - Race conditions are rare (human editing speed is slow)

Client out of sync: - Client can request full state re-sync at any time - Happens automatically on reconnection

---

8. Use Cases and Examples

This section provides real-world scenarios showing how to use Scene Scheduler effectively.

8.1 24/7 Automated Channel

Scenario: A community TV channel that runs 24 hours a day with scheduled programming.

Requirements: - Automated scene switching - Minimal manual intervention - Overnight programming - Ad breaks between shows

Schedule Design:

```
json
[
  {
    "time": "06:00:00",
    "scene": "Morning Show",
    "duration": "02:00:00",
    "sources": [
      {
        "type": "media_source",
        "name": "MorningIntro",
        "file": "/media/intros/morning.mp4",
        "loop": false
      },
      {
        "type": "browser_source",
        "name": "Clock",
        "url": "file:///overlays/clock.html",
        "width": 1920,
        "height": 1080
      }
    ]
  },
  {
    "time": "08:00:00",
    "scene": "News Block",
    "duration": "01:00:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "NewsFeed",
        "input": "rtsp://newscamera.local/live"
      },
      {
        "type": "browser_source",
        "name": "LowerThird",
        "url": "https://graphics.local/news_lower_third.html",
        "width": 1920,
        "height": 200
      }
    ]
  },
  {
    "time": "09:00:00",
    "scene": "Ad Break",
    "duration": "00:03:00",
    "sources": [
      {
        "type": "vlc_source",
        "name": "Commercials",
        "playlist": "/media/ads/morning_ads.xspf"
      }
    ]
  },
  {
    "time": "09:03:00",
    "scene": "Feature Film",
    "duration": "02:00:00",
    "sources": [
      {
        "type": "media_source",
        "name": "Movie",
        "file": "/media/films/morning_feature.mp4",
        "loop": false,
        "hw_decode": true
      }
    ]
  },
  {
    "time": "23:00:00",
    "scene": "Overnight Loop",
    "duration": "07:00:00",
    "sources": [
      {
        "type": "media_source",
        "name": "NightLoop",
        "file": "/media/overnight/holding_screen.mp4",
        "loop": true
      },
      {
        "type": "browser_source",
        "name": "Schedule",
        "url": "file:///overlays/tomorrow_schedule.html",
        "width": 400,
        "height": 800
      }
    ]
  }
]
```

Best practices for this use case: - Test all transitions at least once before going live - Keep backup content ready (use long-duration loops) - Monitor the system remotely via Monitor View - Set up alerts for OBS crashes or connection loss

8.2 Conference or Event Automation

Scenario: Multi-day conference with scheduled speakers, breaks, and sponsor content.

Requirements: - Speaker intro slides - Live camera feeds during talks - Sponsor ads during breaks - Countdown timers

Example: Single Day Schedule

```
json
[
  {
    "time": "08:00:00",
    "scene": "Welcome Screen",
    "duration": "01:00:00",
    "sources": [
      {
        "type": "media_source",
        "name": "WelcomeLoop",
        "file": "/conference/welcome_loop.mp4",
        "loop": true
      },
      {
        "type": "browser_source",
        "name": "Countdown",
        "url": "https://timer.local/countdown?target=09:00:00",
        "width": 400,
        "height": 200
      }
    ]
  },
  {
    "time": "09:00:00",
    "scene": "Keynote",
    "duration": "01:00:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "MainCamera",
        "input": "rtsp://camera1.local/stream"
      },
      {
        "type": "browser_source",
        "name": "SpeakerInfo",
        "url": "https://graphics.local/speaker?id=keynote",
        "width": 500,
        "height": 150
      }
    ]
  },
  {
    "time": "10:00:00",
    "scene": "Break",
    "duration": "00:15:00",
    "sources": [
      {
        "type": "vlc_source",
        "name": "SponsorAds",
        "playlist": "/conference/sponsor_ads.xspf"
      }
    ]
  },
  {
    "time": "10:15:00",
    "scene": "Talk 1",
    "duration": "00:30:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "Speaker1Camera",
        "input": "rtsp://camera2.local/stream"
      },
      {
        "type": "browser_source",
        "name": "Slides",
        "url": "https://slides.local/talk1",
        "width": 1280,
        "height": 720
      }
    ]
  }
]
```

Tips: - Use Preview tab to verify all camera feeds before the event - Have backup scenes ready for technical difficulties - Keep event durations slightly longer than expected (buffer time) - Run a full rehearsal the day before

8.3 Digital Signage

Scenario: Retail store display showing promotions, product videos, and announcements.

Requirements: - Looping content throughout business hours - Special promotions at specific times - Emergency announcement capability

Example Schedule:

```
json
[
  {
    "time": "09:00:00",
    "scene": "Store Opening",
    "duration": "00:05:00",
    "sources": [
      {
        "type": "media_source",
        "name": "OpeningVideo",
        "file": "/signage/store_opening.mp4",
        "loop": false
      }
    ]
  },
  {
    "time": "09:05:00",
    "scene": "General Promotions",
    "duration": "03:55:00",
    "sources": [
      {
        "type": "vlc_source",
        "name": "PromoPlaylist",
        "playlist": "/signage/general_promos.xspf"
      },
      {
        "type": "browser_source",
        "name": "Clock",
        "url": "file:///overlays/store_clock.html",
        "width": 300,
        "height": 100
      }
    ]
  },
  {
    "time": "13:00:00",
    "scene": "Lunch Special",
    "duration": "01:00:00",
    "sources": [
      {
        "type": "media_source",
        "name": "LunchPromo",
        "file": "/signage/lunch_special.mp4",
        "loop": true
      }
    ]
  },
  {
    "time": "14:00:00",
    "scene": "General Promotions",
    "duration": "07:00:00",
    "sources": [
      {
        "type": "vlc_source",
        "name": "PromoPlaylist",
        "playlist": "/signage/general_promos.xspf"
      }
    ]
  },
  {
    "time": "21:00:00",
    "scene": "Store Closing",
    "duration": "00:05:00",
    "sources": [
      {
        "type": "media_source",
        "name": "ClosingVideo",
        "file": "/signage/store_closing.mp4",
        "loop": false
      }
    ]
  },
  {
    "time": "21:05:00",
    "scene": "Closed",
    "duration": "11:55:00",
    "sources": [
      {
        "type": "media_source",
        "name": "ClosedScreen",
        "file": "/signage/closed_screen.mp4",
        "loop": true
      }
    ]
  }
]
```

Emergency announcements: - Manually trigger scene change in OBS - Or: Edit schedule to insert urgent event with near-future time - After emergency: Schedule automatically resumes at next event

8.4 Live Stream Production

Scenario: Weekly live stream with pre-roll, main content, and post-roll.

Requirements: - Automated pre-roll video before going live - Switch to live camera at exact time - Post-roll video after stream ends

Example:

```
json
[
  {
    "time": "19:55:00",
    "scene": "Pre-Roll",
    "duration": "00:05:00",
    "sources": [
      {
        "type": "media_source",
        "name": "PreRollVideo",
        "file": "/stream/pre_roll.mp4",
        "loop": false
      },
      {
        "type": "browser_source",
        "name": "StartingSoonOverlay",
        "url": "https://overlay.local/starting_soon?time=20:00",
        "width": 1920,
        "height": 1080
      }
    ]
  },
  {
    "time": "20:00:00",
    "scene": "Live Stream",
    "duration": "01:00:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "MainCamera",
        "input": "rtsp://camera.local/main"
      },
      {
        "type": "ffmpeg_source",
        "name": "ScreenCapture",
        "input": "rtmp://localhost/screen"
      },
      {
        "type": "browser_source",
        "name": "ChatOverlay",
        "url": "https://chat.local/embed",
        "width": 400,
        "height": 600
      }
    ]
  },
  {
    "time": "21:00:00",
    "scene": "Post-Roll",
    "duration": "00:03:00",
    "sources": [
      {
        "type": "media_source",
        "name": "PostRollVideo",
        "file": "/stream/post_roll.mp4",
        "loop": false
      },
      {
        "type": "browser_source",
        "name": "ThankYouOverlay",
        "url": "https://overlay.local/thanks",
        "width": 1920,
        "height": 1080
      }
    ]
  },
  {
    "time": "21:03:00",
    "scene": "Offline Screen",
    "duration": "22:52:00",
    "sources": [
      {
        "type": "media_source",
        "name": "OfflineLoop",
        "file": "/stream/offline.mp4",
        "loop": true
      }
    ]
  }
]
```

Pro tips: - Start OBS recording 5 minutes before scheduled time - Use Scene Scheduler for timing, but monitor chat manually - Have backup scenes for technical difficulties - Test preview of all sources 30 minutes before going live

8.5 Worship Service Automation

Scenario: Automated elements of a church service while allowing manual control for live elements.

Requirements: - Pre-service announcements and countdown - Automated hymn/song lyrics - Manual sermon control - Post-service loop

Mixed Automation Approach:

```
json
[
  {
    "time": "09:30:00",
    "scene": "Pre-Service",
    "duration": "00:30:00",
    "sources": [
      {
        "type": "vlc_source",
        "name": "Announcements",
        "playlist": "/worship/announcements.xspf"
      },
      {
        "type": "browser_source",
        "name": "ServiceCountdown",
        "url": "https://timer.local/countdown?target=10:00:00",
        "width": 600,
        "height": 200
      }
    ]
  },
  {
    "time": "10:00:00",
    "scene": "Welcome",
    "duration": "00:05:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "MainCamera",
        "input": "rtsp://camera.local/front"
      },
      {
        "type": "browser_source",
        "name": "WelcomeSlide",
        "url": "file:///slides/welcome.html",
        "width": 1920,
        "height": 1080
      }
    ]
  },
  {
    "time": "10:05:00",
    "scene": "Worship Songs",
    "duration": "00:25:00",
    "sources": [
      {
        "type": "ffmpeg_source",
        "name": "WideShot",
        "input": "rtsp://camera.local/wide"
      },
      {
        "type": "browser_source",
        "name": "Lyrics",
        "url": "https://lyrics.local/service?set=1",
        "width": 1920,
        "height": 400
      }
    ]
  }
]
```

Manual override: - Operator can manually switch scenes in OBS during sermon - Schedule resumes after sermon with post-service loop - Scene Scheduler handles repetitive elements, human handles dynamic parts

---

9. Best Practices

9.1 Media File Recommendations

Video Encoding

Recommended codec settings: - **Codec:** H.264 (AVC) - **Profile:** High - **Level:** 4.2 or higher - **Bitrate:** - 1080p: 8-12 Mbps (CBR for consistent playback) - 720p: 5-8 Mbps - 4K: 25-40 Mbps (test system performance) - **Frame rate:** Match OBS output (typically 30 or 60 fps) - **Keyframe interval:** 2 seconds (60 frames at 30fps, 120 frames at 60fps)

Audio encoding: - **Codec:** AAC - **Bitrate:** 192 kbps (stereo) or 384 kbps (5.1) - **Sample rate:** 48 kHz (match OBS) - **Channels:** Stereo (2.0) for most use cases

Container format: - **Preferred:** MP4 (.mp4) - **Alternative:** MKV (.mkv) for flexibility - **Avoid:** AVI (dated), MOV (codec issues)

ffmpeg encoding example:

```
bash
ffmpeg -i input.mov -c:v libx264 -preset medium -crf 20 \
  -c:a aac -b:a 192k -ar 48000 \
  -movflags +faststart output.mp4
```

Why these settings? - H.264: Universal compatibility, hardware decoding support - CBR: Prevents buffer underruns during playback - Keyframes: Allows seeking, smooth looping - AAC: Standard audio codec, low latency - `faststart`: Metadata at beginning (faster loading)

File Organization

Recommended directory structure:

```
/media/
├── intros/
│   ├── morning_intro.mp4
│   ├── evening_intro.mp4
│   └── weekend_intro.mp4
├── content/
│   ├── show1/
│   │   ├── episode01.mp4
│   │   └── episode02.mp4
│   └── show2/
│       └── episode01.mp4
├── ads/
│   ├── commercial_1.mp4
│   └── commercial_2.mp4
├── overlays/
│   ├── lower_third.html
│   └── clock.html
└── backgrounds/
    ├── holding_screen.mp4
    └── offline_loop.mp4
```

Benefits: - Easy to locate files when configuring events - Clear naming prevents mistakes - Organized backups

Naming conventions: - Use lowercase with underscores: `morning_intro.mp4` - Include date/version if applicable: `news_2025_10_28.mp4` - Avoid spaces (use `my_file.mp4` not `my file.mp4`) - Be descriptive: `commercial_acme_30s.mp4` not `comm1.mp4`

File Storage

Local vs. Network Storage:

Local storage (recommended): - Fastest access (no network latency) - Most reliable (no network dependencies) - Best for large files (4K video) - Limited by disk size

Network storage (NFS/SMB): - Centralized management - Easy to update content remotely - Network latency affects load times - Single point of failure (network/server)

Hybrid approach: - Store frequently-used files locally (intros, loops) - Store large archives on network (past episodes) - Cache network files locally when possible

9.2 Performance Optimization

System Requirements

Minimum specs: - CPU: 4 cores, 2.5 GHz - RAM: 8 GB - Disk: SSD (for fast media access) - Network: 100 Mbps (for network streams)

Recommended specs: - CPU: 6-8 cores, 3.0+ GHz - RAM: 16 GB - Disk: NVMe SSD - GPU: Dedicated GPU for hardware encoding/decoding - Network: 1 Gbps

For 4K or multiple network streams: - CPU: 8+ cores or dedicated encoding GPU - RAM: 32 GB - Disk: RAID for redundancy

OBS Configuration

Reduce CPU usage: - Enable hardware encoding (NVENC, QuickSync, or AMF) - Lower output resolution if possible (1080p vs. 4K) - Disable unused sources/scenes - Use hardware decoding for media sources

Optimize sources: - Limit browser source count (high CPU/GPU usage) - Use static images instead of browser sources when possible - Disable "Shutdown when not visible" for always-on sources

Scene Scheduler-specific: - Test event transitions with all configured sources - Monitor CPU/GPU usage during event triggers - Use shorter durations if sources are lightweight (reduce cleanup overhead)

Browser Source Optimization

HTML/CSS/JS optimization: - Minimize JavaScript (avoid heavy frameworks) - Use CSS animations (not JavaScript setTimeout loops) - Optimize images (compress, use SVG when possible) - Lazy-load assets (don't load everything on page load)

Example: Optimized clock overlay

```
html
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background: transparent;
      margin: 0;
      padding: 20px;
      font-family: Arial, sans-serif;
      color: white;
      font-size: 48px;
      text-shadow: 2px 2px 4px black;
    }
  </style>
</head>
<body>
  <div id="clock"></div>
  <script>
    // Update only once per second (not 60fps)
    setInterval(() => {
      document.getElementById('clock').textContent =
        new Date().toLocaleTimeString();
    }, 1000);
  </script>
</body>
</html>
```

FPS setting: - Static overlays: 1-5 FPS - Animated overlays: 30 FPS - Smooth animations: 60 FPS (only if needed)

9.3 Reliability and Uptime

For 24/7 Operation

System-level: - **Linux**: More stable for long-running services, use systemd for auto-restart - **Windows**: Use Task Scheduler or NSSM (Non-Sucking Service Manager) for service installation - Disable automatic updates (manual maintenance windows) - Monitor system resources (CPU, RAM, disk)

OBS configuration: - Disable auto-updates - Configure auto-reconnect for streaming - Use scene collections (easy recovery) - Regular profile backups

Scene Scheduler: - **Linux**: Run as systemd service (auto-restart on crash) - **Windows**: Run as Windows Service using NSSM or Task Scheduler - Configure log rotation (prevent disk fill) - Monitor logs for errors - Alert on OBS disconnection

Linux systemd service example:

```
ini
[Unit]
Description=Scene Scheduler
After=network.target

[Service]
Type=simple
User=obs
WorkingDirectory=/opt/scenescheduler
ExecStart=/opt/scenescheduler/scenescheduler
Restart=always
RestartSec=10
Environment="OBS_WS_PASSWORD=yourpassword"

[Install]
WantedBy=multi-user.target
```

Windows service with NSSM:

```
cmd
REM Install NSSM from https://nssm.cc/

REM Install Scene Scheduler as service
nssm install SceneScheduler "C:\scenescheduler\scenescheduler.exe"
nssm set SceneScheduler AppDirectory "C:\scenescheduler"
nssm set SceneScheduler AppEnvironmentExtra OBS_WS_PASSWORD=yourpassword
nssm start SceneScheduler
```

Backup Strategy

What to backup: 1. `schedule.json` (critical) 2. `config.json` 3. OBS scene collections 4. Media files (if not easily replaceable)

Backup frequency: - schedule.json: After every major change - config.json: After initial setup and changes - Media files: Weekly or after adding new content

Backup methods:

Linux:

```
bash
# Simple backup script
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups/scenescheduler"

# Backup configuration and schedule
cp /opt/scenescheduler/config.json "$BACKUP_DIR/config_$DATE.json"
cp /opt/scenescheduler/schedule.json "$BACKUP_DIR/schedule_$DATE.json"

# Keep only last 30 backups
ls -t $BACKUP_DIR/schedule_*.json | tail -n +31 | xargs rm -f
```

Run automatically with cron:

```
cron
# Backup every day at 3 AM
0 3 * * * /opt/scenescheduler/backup.sh
```

Windows:

```
batch
REM backup.bat
@echo off
set TIMESTAMP=%date:~-4%%date:~3,2%%date:~0,2%_%time:~0,2%%time:~3,2%%time:~6,2%
set BACKUP_DIR=C:\backups\scenescheduler

REM Backup configuration and schedule
copy C:\scenescheduler\config.json "%BACKUP_DIR%\config_%TIMESTAMP%.json"
copy C:\scenescheduler\schedule.json "%BACKUP_DIR%\schedule_%TIMESTAMP%.json"

REM Delete backups older than 30 days
forfiles /p "%BACKUP_DIR%" /m schedule_*.json /d -30 /c "cmd /c del @path"
```

Run automatically with Task Scheduler: 1. Open Task Scheduler 2. Create Basic Task → Daily at 3:00 AM 3. Action: Start a program → `C:\scenescheduler\backup.bat`

9.4 Testing and Validation

Pre-Production Testing

Schedule validation checklist: 1. All event times are in the future 2. No overlapping events (or intentional overrides) 3. All scenes exist in OBS 4. All media file paths are valid 5. All network URLs are reachable 6. Event durations are appropriate 7. 24-hour coverage (or intentional gaps)

Source validation: 1. Preview each source in Preview tab 2. Verify visual appearance 3. Check audio levels 4. Test looping behavior (if enabled) 5. Verify browser sources load completely

System validation: 1. Test full day schedule in fast-forward (set times 1 minute apart) 2. Verify staging works (sources appear instantly) 3. Check cleanup (sources removed after duration) 4. Test reconnection (kill OBS, restart, verify reconnect) 5. Test web interface from remote device

Ongoing Monitoring

Daily checks: - Verify schedule loaded correctly (check web interface) - Confirm OBS connection status (green indicator) - Review logs for errors

Weekly checks: - Test a few event transitions manually - Verify media files are accessible - Check disk space (HLS previews, logs)

Monthly checks: - Full schedule review (remove old events) - Update media content - Test backup restoration - Review system resource usage (CPU, RAM trends)

9.5 Security Best Practices

OBS WebSocket: - Always use a strong password - Use environment variables (not hardcoded in config) - Don't expose WebSocket port to public internet - Use firewall rules to restrict access

Web Server: - Bind to localhost if network access not needed - Use reverse proxy with authentication for remote access - Don't expose without authentication - Use HTTPS if accessible over untrusted networks

File Permissions:

Linux:

```
bash
chmod 600 config.json schedule.json
chmod 700 hls/
chown obs:obs /opt/scenescheduler -R
```

Windows: - Right-click config.json → Properties → Security - Ensure only your user and SYSTEM have access - Remove "Everyone" or "Users" group if present

Access Control: - Limit SSH access to server - Use SSH keys (not passwords) - Regularly review user accounts - Monitor access logs

---

10. Troubleshooting

This section helps diagnose and resolve common issues. Start with the FAQ for quick answers, then consult specific problem categories.

10.1 Frequently Asked Questions (FAQ)

Q: Scene Scheduler won't start. What should I check?

A: Follow this diagnostic sequence:

1. **Check config.json exists and is valid JSON:**

```
bash
   jq . config.json
```

If error: Fix JSON syntax (missing commas, quotes, brackets)

1. **Verify OBS is running and WebSocket is enabled:** - Open OBS → Tools → WebSocket Server Settings - Confirm "Enable WebSocket server" is checked - Note the port (should match config.json)

2. **Test WebSocket connection manually:**

```
bash
   nc -zv localhost 4455
```

If connection refused: OBS WebSocket not running

1. **Check password matches:** - config.json: `"password": "yourpassword"` - OBS WebSocket settings: Same password

2. **Review logs for specific error:**

```
bash
   ./scenescheduler
```

Look for `FATAL` or `ERROR` messages in output

Q: Preview button shows "Waiting for stream..." forever (30 second timeout)

A: This indicates HLS playlist generation failed. Common causes:

Cause 1: hls-generator binary missing or not executable

```
bash
ls -la ./hls-generator
# Should show: -rwxr-xr-x (executable)

# If not executable:
chmod +x hls-generator
```

Cause 2: HLS directory doesn't exist or isn't writable

```
bash
mkdir -p hls
chmod 755 hls
```

Cause 3: Source file/URL is invalid - Media source: File doesn't exist (`ls -la /path/to/file.mp4`) - Browser source: URL unreachable (test in regular browser) - FFMPEG source: Network stream down (`curl -I <url>`)

Cause 4: OBS libraries missing (for hls-generator)

```
bash
ldd ./hls-generator
# Check for "not found" lines
```

Q: Events trigger but sources don't appear in OBS

A: Check these common issues:

1. **Staging scene (`scheduleSceneAux`) doesn't exist:** - Open OBS - Verify scene exists with exact name from config.json - Create if missing (right-click in Scenes panel → Add)
2. **File paths are incorrect:** - Check logs for "file not found" errors - Verify paths are absolute (not relative) - Test file access: `cat /path/to/file.mp4 > /dev/null`
3. **Network streams are unreachable:** - Ping IP addresses: `ping 192.168.1.100` - Test URLs: `curl -I rtsp://camera.local/stream` - Check firewall rules
4. **OBS source creation failed:** - Check OBS logs: `~/.config/obs-studio/logs/` - Look for "failed to create source" messages

Q: Can I access Scene Scheduler from another computer/phone/tablet?

A: Yes! This is a **key feature** of Scene Scheduler - controlling OBS remotely without loading the OBS machine.

Setup for network access:

1. **Configure config.json for network binding:**

```
json
   "webServer": {
     "host": "0.0.0.0",
     "port": 8080,
     "hlsPath": "hls"
   }
```

2. **Find your server's IP address:**

Linux:

```
bash
   ip addr show | grep inet
```

Windows:

```
cmd
   ipconfig
```

Look for IPv4 address (e.g., 192.168.1.100)

1. **Access from any device on your network:** - Monitor View: `http://192.168.1.100:8080/` - Editor View: `http://192.168.1.100:8080/editor.html`

2. **If connection fails, check firewall:**

Linux:

```
bash
   sudo ufw allow 8080/tcp
```

Windows: - Open Windows Defender Firewall - Add inbound rule for TCP port 8080

Benefits: Control OBS from your laptop/tablet while OBS runs on a dedicated server, reducing load and allowing multiple people to monitor simultaneously.

Q: Web interface shows "Disconnected" status

A: The frontend cannot reach the backend. Diagnose:

Check 1: Backend is running

```
bash
ps aux | grep scenescheduler
# Should show process running
```

Check 2: Web server port is accessible

```
bash
# Test from same machine
curl http://localhost:8080
# Should return HTML (not "connection refused")

# Test from network (if accessing remotely)
curl http://<server-ip>:8080
```

Check 3: Host configuration - Check config.json `webServer.host` setting - For network access, must be `"0.0.0.0"` not `"localhost"`

Check 4: Firewall allows port

```
bash
sudo ufw status
# Check port 8080 is allowed
```

Check 4: Browser can reach backend - Open browser console (F12 → Console) - Look for WebSocket errors - Check URL is correct (http://localhost:8080, not https://)

Q: Scenes transition but sources from previous event remain visible

A: This indicates cleanup failure. Possible causes:

1. **Event duration is too long:** - Sources cleanup happens AFTER duration expires - Check event duration field - Reduce if needed

2. **Manual sources added in OBS:** - Scene Scheduler only removes sources IT created - Manual sources remain - Solution: Remove manually or use dedicated scenes

3. **Cleanup error:** - Check logs for "cleanup failed" messages - OBS might be unresponsive (restart OBS)

Q: Preview works but event staging fails with same source

A: Preview and staging are separate systems. Staging failures may indicate:

1. **Timing issue (insufficient staging time):** - Large files may not load in 30 seconds - Use smaller files or local copies - Test with Preview tab (if loads in <30s, staging should work)

2. **Race condition (multiple events staging simultaneously):** - Check schedule for events <30 seconds apart - Stagger event times

3. **Resource exhaustion:** - CPU/GPU maxed out during staging - Monitor resources: `htop` during T-30s window - Reduce source count or complexity

Q: How do I reset everything to a clean state?

A: Complete reset procedure:

Linux:

```
bash
# 1. Stop Scene Scheduler
pkill scenescheduler

# 2. Clear schedule
echo "[]" > schedule.json

# 3. Clear HLS preview files
rm -rf hls/*

# 4. Restart OBS (clear all dynamic sources)
killall obs
obs &

# 5. Start Scene Scheduler (auxiliary scene will be recreated automatically)
./scenescheduler
```

Windows:

```
cmd
REM 1. Stop Scene Scheduler (Ctrl+C or close window)

REM 2. Clear schedule
echo [] > schedule.json

REM 3. Clear HLS preview files
```

```
rmdir /s /q hls
mkdir hls

REM 4. Restart OBS
taskkill /IM obs64.exe /F
start "" "C:\Program Files\obs-studio\bin\64bit\obs64.exe"

REM 5. Start Scene Scheduler (auxiliary scene will be recreated automatically)
scenescheduler.exe
```

Q: Can I run multiple Scene Scheduler instances?

A: Yes, but each needs: - Different web server port (config.json: `"port": 8081`) - Different HLS directory (config.json: `"hlsPath": "hls2"`) - Different schedule file (config.json: `"jsonPath": "schedule2.json"`) - Different auxiliary scene (config.json: `"scheduleSceneAux": "scheduleSceneAux2"`)

All instances can connect to the same OBS.

Q: My schedule file is corrupted. How do I recover?

A: Recovery steps:

1. **Check for auto-backups (if configured):**

   ```
   bash
      ls -la /backups/scenescheduler/schedule_*.json
      # Restore latest
      cp /backups/scenescheduler/schedule_LATEST.json schedule.json
   ```

2. **Manually repair JSON:**

   ```
   bash
      # Validate current file
      jq . schedule.json
      # Shows error line/column

      # Edit with text editor
      nano schedule.json
   ```

3. **Start from scratch:**

   ```
   json
      []
   ```

Save as `schedule.json`, then rebuild schedule in web interface

Q: Event triggers at wrong time (timezone issues?)

A: Scene Scheduler uses **system local time**, not UTC.

Check system timezone:

```
bash
timedatectl
# Shows: Time zone: America/New_York (EST, -0500)
```

If wrong, set correct timezone:

```
bash
sudo timedatectl set-timezone America/Los_Angeles
```

Verify system time:

```
bash
date
# Should match your local time
```

Events trigger when system time matches event `time` field (HH:MM:SS).

Q: How do I upgrade to a new Scene Scheduler version?

A: Safe upgrade procedure:

Linux:

```
bash
# 1. Backup current config and schedule
cp config.json config.json.backup
cp schedule.json schedule.json.backup

# 2. Stop current version
pkill scenescheduler

# 3. Extract new version
tar -xzf scenescheduler-v0.4-linux.tar.gz

# 4. Restore config and schedule
cp config.json.backup config.json
cp schedule.json.backup schedule.json

# 5. Start new version
./scenescheduler
```

Windows:

```
cmd
REM 1. Backup current config and schedule
copy config.json config.json.backup
copy schedule.json schedule.json.backup

REM 2. Stop current version (Ctrl+C or close window)

REM 3. Download and extract new version to same folder
REM (Overwrite scenescheduler.exe and hls-generator.exe)

REM 4. Restore config and schedule (if overwritten)
copy config.json.backup config.json
copy schedule.json.backup schedule.json

REM 5. Start new version
scenescheduler.exe
```

Check release notes for breaking changes or config schema updates.

10.2 Diagnostic Workflow

When encountering issues, follow this systematic diagnostic process:

Step 1: Identify the Symptom

Categorize your issue: - **Application won't start**: See Section 10.3 - **OBS connection problems**: See Section 10.4 - **Preview failures**: See Section 10.5 - **Event staging/transition issues**: See Section 10.6 - **Web interface problems**: See Section 10.7 - **Performance issues**: See Section 10.8

Step 2: Gather Information

Collect diagnostic data:

1. **Scene Scheduler logs:**

```
bash
   ./scenescheduler 2>&1 | tee scenescheduler.log
```

2. **OBS logs:**

Linux:

```
bash
   tail -f ~/.config/obs-studio/logs/$(ls -t ~/.config/obs-studio/logs/ | head -1)
```

Windows:

```
Open: %APPDATA%\obs-studio\logs\
   View the most recent log file in Notepad
```

1. **System resource usage:**

Linux:

```
bash
   htop
   # Note CPU, RAM, disk usage
```

Windows:

```
Open Task Manager (Ctrl+Shift+Esc)
   Check Performance tab for CPU, RAM, disk usage
```

1. **Configuration:**

```
bash
   cat config.json
   cat schedule.json
```

2. **Network connectivity (if using remote OBS or network sources):**

```
bash
   ping <obs-host>
   curl -I <stream-url>
```

Step 3: Reproduce the Issue

Create a minimal test case:

1. **Simplify schedule:** - Remove all events except one - Use simple source (local video file) - Set time 2 minutes in future

2. **Test in isolation:** - Does the simplified event work? - If yes: Complexity issue (too many sources/events) - If no: Fundamental configuration problem

3. **Document reproduction steps:** - Exact sequence of actions - Expected vs. actual behavior - Any error messages

Step 4: Apply Fix

After identifying the root cause (Sections 10.3-10.8), apply the fix and verify:

1. **Make one change at a time**
2. **Test after each change**
3. **Document what fixed the issue**
4. **Update your configuration/schedule accordingly**

10.3 Application Startup Issues

Problem: Scene Scheduler fails to start or exits immediately.

Error: "Failed to parse config.json"

Symptom:

```
FATAL: Failed to parse config.json: invalid character '}' looking for beginning of objec
```

Cause: Invalid JSON syntax in config.json.

Solution: 1. Validate JSON:

```
bash
   jq . config.json
```

1. Common issues: - Missing commas between fields - Trailing commas before closing braces - Missing quotes around strings - Mismatched brackets/braces

2. Fix syntax and retry

Error: "OBS WebSocket connection failed"

Symptom:

```
ERROR: Failed to connect to OBS WebSocket: dial tcp [::1]:4455: connect: connection refu
```

Cause: OBS not running or WebSocket server disabled.

Solution: 1. Start OBS Studio 2. Enable WebSocket: Tools → WebSocket Server Settings → "Enable WebSocket server" 3. Verify port matches config.json 4. Restart Scene Scheduler

Error: "Authentication failed"

Symptom:

```
ERROR: OBS WebSocket authentication failed: invalid password
```

Cause: Password mismatch between config.json and OBS settings.

Solution: 1. Check OBS password: Tools → WebSocket Server Settings 2. Update config.json to match:

```
json
   "obsWebSocket": {
     "password": "correct-password-here"
   }
```

1. Or use environment variable:

   ```
   bash
      export OBS_WS_PASSWORD="correct-password"
      ./scenescheduler
   ```

Error: "Schedule file not found"

Symptom:

```
ERROR: Failed to load schedule: open schedule.json: no such file or directory
```

Cause: schedule.json doesn't exist at expected location.

Solution: 1. Create empty schedule:

```
bash
   echo "[]" > schedule.json
```

1. Or specify different path in config.json:

   ```
   json
      "schedule": {
        "jsonPath": "/path/to/your/schedule.json"
      }
   ```

Error: "Port already in use"

Symptom:

```
FATAL: Failed to start web server: listen tcp :8080: bind: address already in use
```

Cause: Another process is using port 8080.

Solution: 1. Find process using port:

```
bash
   sudo lsof -i :8080
```

1. Kill it (if safe):

   ```
   bash
      kill <PID>
   ```

2. Or use different port in config.json:

   ```
   json
      "webServer": {
        "port": 8081
      }
   ```

10.4 OBS Connection Issues

Problem: Scene Scheduler starts but can't communicate with OBS.

Disconnects Immediately After Connecting

Symptoms: - "Connected to OBS" followed by "Disconnected" in logs - Web interface shows red dot (disconnected)

Causes and solutions:

1. OBS WebSocket version mismatch:

```
bash
# Check OBS version
obs --version

# Scene Scheduler v1.6 requires OBS WebSocket 5.x
# OBS 28+ includes WebSocket 5.x by default
# Older OBS: Install obs-websocket 5.x plugin
```

2. Network instability (remote OBS):

```
bash
# Test connection stability
ping -c 100 <obs-host>
# Look for packet loss

# Check network latency
ping <obs-host>
# Should be <50ms for local network
```

3. Firewall blocking reconnection:

```
bash
# Check firewall rules
sudo ufw status

# Allow OBS WebSocket port
sudo ufw allow 4455/tcp
```

Frequent Disconnects/Reconnects

Symptoms: - Logs show repeated disconnect/reconnect cycles - Events trigger inconsistently

Causes and solutions:

1. OBS crashing or freezing: - Check OBS logs for crashes - Reduce OBS scene complexity - Update OBS to latest version

2. System resource exhaustion:

```
bash
# Monitor during disconnects
htop

# If CPU/RAM maxed out:
# - Reduce OBS source count
# - Disable preview in OBS
# - Close other applications
```

3. Network issues (remote OBS): - Check switch/router logs - Test with direct ethernet connection - Use wired instead of wireless

Commands Timeout or Fail

Symptoms: - Scene transitions delayed - Sources not created - Logs show "request timeout" errors

Causes and solutions:

1. OBS overloaded: - Reduce scene complexity - Enable hardware encoding - Close unused scenes

2. Scene Scheduler making too many requests: - Reduce event frequency - Simplify source configurations - Increase staging window (code change required)

10.5 Preview System Issues

Problem: Preview fails to start, shows errors, or behaves unexpectedly.

Preview Timeout (30 Seconds)

Symptom: "Waiting for stream..." never resolves, timeout after 30s.

Diagnostic steps:

1. Check hls-generator exists and runs:

```
bash
# Verify exists
ls -la ./hls-generator

# Try running manually
./hls-generator --help
# Should show usage, not "command not found"
```

2. Check HLS directory permissions:

```
bash
# Check writable
touch hls/test.txt
rm hls/test.txt

# If permission denied:
chmod 755 hls
```

3. Test source accessibility:

For media sources:

```
bash
# File exists?
ls -la /path/to/file.mp4

# File readable?
cat /path/to/file.mp4 > /dev/null
```

For browser sources:

```
bash
# URL reachable?
curl -I https://example.com/overlay.html

# DNS works?
nslookup example.com
```

For FFMPEG sources:

```
bash
# Stream reachable?
ffprobe rtsp://camera.local/stream

# Network route exists?
traceroute camera.local
```

4. Check disk space:

```
bash
df -h
# Ensure partition with hls/ has free space
```

Preview Shows Black Screen

Symptom: Preview starts but video is black/blank.

Causes:

1. Browser source with transparent background: - This is normal for browser sources - Transparency shows as black in preview - In OBS, overlay will work correctly

2. Video codec unsupported:

```
bash
# Check codec
ffprobe /path/to/file.mp4

# Look for codec_name (should be h264)
# If not h264, re-encode:
ffmpeg -i input.mp4 -c:v libx264 -c:a aac output.mp4
```

3. Network stream not sending video: - Test stream in VLC or other player - Check camera/encoder settings

Preview Audio But No Video (or vice versa)

Symptom: Can hear audio but no video, or video plays silently.

Causes:

1. Single-track media file: - File may only contain video or audio - Check with ffprobe:

```
bash
  ffprobe file.mp4
  # Look for both "Video:" and "Audio:" streams
```

2. Codec issue: - Video codec unsupported but audio works - Re-encode with standard codecs (H.264 + AAC)

3. Browser source audio disabled in OBS: - This is OBS-level setting - Preview uses isolated OBS instance - Audio should work in production

10.6 Event Staging and Transition Issues

Problem: Events trigger but scenes don't switch, or sources don't appear.

Scene Doesn't Switch at Event Time

Symptom: Event time arrives, but OBS stays on current scene.

Diagnostic:

1. Check event time format:

```
json
// CORRECT:
"time": "14:30:00"

// WRONG:
"time": "2:30 PM"
"time": "14:30"
"time": "14:30:00:000"
```

2. Check system time:

```
bash
date
# Should show correct local time
```

3. Check schedule loaded: - Open web interface - Verify event appears in list - Check "Current time" matches system time

4. Check logs for errors:

```
bash
# Look for event trigger message
grep "event triggered" scenescheduler.log
```

Scene Switches But Sources Don't Appear

Symptom: OBS switches to correct scene, but sources missing or black.

Common causes:

1. Auxiliary scene configuration issue:

```
Solution: Check scheduleSceneAux name in config.json (Scene Scheduler creates it automat
```

2. File paths incorrect:

```
bash
# Check paths in schedule.json
# Paths must be absolute:

# Linux:
"/home/user/video.mp4"
"~/video.mp4"
"./video.mp4"
"video.mp4"

# Windows:
"C:/Videos/video.mp4"
"C:\Videos\video.mp4"
"Videos\video.mp4"
"video.mp4"
```

3. Sources failed to create during staging:

```
# Check logs for errors during event trigger
# Look for "failed to create source" messages
```

4. Permissions issue:

```
bash
# Verify Scene Scheduler user can read files
```

```
sudo -u obs cat /path/to/file.mp4 > /dev/null
# Should not show "permission denied"
```

Sources Appear Late (Not Preloaded)

Symptom: Scene switches but sources load visibly (buffering, black screen for few seconds).

Causes:

1. Staging didn't complete: - 30-second window insufficient - Use smaller files - Improve network speed (for streams)

2. File on slow storage: - Network mount with high latency - Copy files locally:

```
bash
  cp /nfs/remote/file.mp4 /local/file.mp4
```

3. Too many sources loading simultaneously: - Reduce source count per event - Test with fewer sources to isolate the issue

10.7 Web Interface Issues

Problem: Web interface doesn't load, shows errors, or updates don't appear.

"Cannot GET /" or Connection Refused

Symptom: Browser shows error when accessing Scene Scheduler web interface.

Causes:

1. Backend not running:

```
bash
ps aux | grep scenescheduler
# If nothing: Start backend
./scenescheduler
```

2. Wrong URL: - Check protocol: `http://` not `https://` - Check port matches config.json - **Accessing from same machine**: Use `http://localhost:8080` - **Accessing from network**: Use `http://<server-ip>:8080` (e.g., `http://192.168.1.100:8080`)

3. config.json host setting: - **For network access**: `"host": "0.0.0.0"` (binds to all interfaces) - **For local-only**: `"host": "localhost"` (blocks network access) - If you can't connect from network but need to, change host to `0.0.0.0` and restart

4. Firewall blocking:

```
bash
# Test locally first
curl http://localhost:8080

# If works locally but not from network:
sudo ufw allow 8080/tcp
```

WebSocket "Disconnected" Status

Symptom: Interface loads but shows red "Disconnected" indicator.

Causes:

1. WebSocket connection blocked: - Check browser console (F12 → Console) - Look for WebSocket errors - Some corporate networks block WebSockets

2. Backend restarted: - Refresh page (F5) - WebSocket should reconnect automatically

3. CORS/proxy issue: - If accessing through proxy/reverse proxy - Configure proxy to allow WebSocket upgrades

Changes Not Appearing in Real-Time

Symptom: Edit event in Editor View, but Monitor View doesn't update.

Causes:

1. WebSocket disconnected: - Check connection indicator - Refresh page

2. Browser cache: - Hard refresh: Ctrl+Shift+R (Linux) - Or clear cache

3. Multiple backend instances: - Each backend has separate state - Ensure all clients connect to same backend

Preview Button Stuck in "Starting..." State

Symptom: Button shows "Starting preview..." indefinitely.

Causes:

1. WebSocket message lost: - Check connection indicator - Stop preview manually (reload page)

2. Backend preview process crashed: - Check logs for "preview failed" errors - Restart backend if necessary

3. Browser JavaScript error: - Check console (F12 → Console) - Reload page

10.8 Performance Issues

Problem: High CPU/RAM usage, lag, or slow response times.

High CPU Usage

Symptom: CPU constantly high (>80%) even with no active events.

Causes:

1. Browser sources running continuously: - Browser sources consume CPU even when scene hidden - Solution: Enable "Shutdown when not visible" in source settings

2. Too many scenes in OBS: - Each scene consumes memory - Delete unused scenes

3. Multiple previews running: - Check for forgotten preview windows - Close preview modal after testing

4. Inefficient browser source JavaScript: - Excessive animations or polling - Optimize JavaScript (see Section 9.2)

High Memory Usage

Symptom: RAM usage grows over time, eventually causes crashes.

Causes:

1. Memory leak in browser sources: - Complex SPAs with leaky JavaScript - Refresh sources periodically (scene switch)

2. Large media files: - 4K videos consume lots of RAM - Use 1080p when possible

3. Orphaned HLS preview files: - Old preview directories not cleaned up - Manual cleanup:

```
bash
  rm -rf hls/*
```

Slow Scene Transitions

Symptom: Scene changes happen several seconds late.

Causes:

1. OBS overloaded: - Too many sources rendering - Reduce scene complexity

2. Network latency (remote OBS):

```
bash
ping <obs-host>
# Should be <10ms for local network
```

3. System resource contention: - Other applications competing for CPU/GPU - Close unnecessary applications

4. Disk I/O bottleneck:

```
bash
iotop
# Check if disk at 100%
```

- Use SSD instead of HDD
- Reduce concurrent file access

---

11. Technical Reference

11.1 Schedule JSON Schema

Complete reference for `schedule.json` structure.

Root Structure

```
json
[
  {
    "time": "HH:MM:SS",
    "scene": "SceneName",
    "duration": "HH:MM:SS",
    "name": "Optional Event Name",
    "sources": [...]
  }
]
```

Event Object Fields

Field	Type	Required	Description
`time`	string	Yes	Time to trigger (HH:MM:SS format, 24-hour)
`scene`	string	Yes	Name of OBS scene to activate
`duration`	string	Yes	How long to keep sources active (HH:MM:SS)
`name`	string	No	Human-readable event name (for UI display)
`sources`	array	No	Array of source objects to add to scene

Source Object: media_source

```
json
{
  "type": "media_source",
  "name": "SourceName",
  "file": "/absolute/path/to/file.mp4",
  "loop": true,
  "restart_on_activate": false,
  "hw_decode": true
}
```

Field	Type	Required	Default	Description
`type`	string	Yes	-	Must be `"media_source"`
`name`	string	Yes	-	Unique source name in OBS
`file`	string	Yes	-	Absolute path to media file
`loop`	boolean	No	`false`	Loop video continuously
`restart_on_activate`	boolean	No	`false`	Restart from beginning on scene activate

Field	Type	Required	Default	Description
`hw_decode`	boolean	No	`false`	Use hardware decoding (GPU)

Source Object: browser_source

```
json
{
  "type": "browser_source",
  "name": "SourceName",
  "url": "https://example.com/overlay.html",
  "width": 1920,
  "height": 1080,
  "css": "body { background: transparent; }",
  "shutdown_when_hidden": true,
  "refresh_on_activate": false,
  "fps": 30
}
```

Field	Type	Required	Default	Description
`type`	string	Yes	-	Must be `"browser_source"`
`name`	string	Yes	-	Unique source name
`url`	string	Yes	-	Full URL (https://, http://, file:///)
`width`	integer	Yes	-	Viewport width (pixels)
`height`	integer	Yes	-	Viewport height (pixels)
`css`	string	No	`""`	Custom CSS to inject
`shutdown_when_hidden`	boolean	No	`false`	Stop rendering when hidden
`refresh_on_activate`	boolean	No	`false`	Reload page on scene activate
`fps`	integer	No	`30`	Frame rate (1-60)

Source Object: ffmpeg_source

```
json
{
  "type": "ffmpeg_source",
  "name": "SourceName",
  "input": "rtsp://camera.local/stream",
  "input_format": "",
  "buffering_mb": 2,
  "reconnect_delay_sec": 5,
  "hw_decode": false
}
```

Field	Type	Required	Default	Description
`type`	string	Yes	-	Must be `"ffmpeg_source"`
`name`	string	Yes	-	Unique source name
`input`	string	Yes	-	Stream URL (rtsp://, rtmp://, srt://, etc.)
`input_format`	string	No	`""`	Container format (auto-detect if empty)
`buffering_mb`	integer	No	2	Buffer size in MB (1-10)
`reconnect_delay_sec`	integer	No	5	Seconds to wait before reconnect attempt
`hw_decode`	boolean	No	`false`	Use hardware decoding

Source Object: vlc_source

```
json
{
  "type": "vlc_source",
  "name": "SourceName",
  "playlist": "/path/to/playlist.xspf",
  "loop": true,
  "shuffle": false
}
```

Field	Type	Required	Default	Description
`type`	string	Yes	-	Must be `"vlc_source"`
`name`	string	Yes	-	Unique source name
`playlist`	string	Yes	-	Path to playlist file (.xspf, .m3u)
`loop`	boolean	No	`false`	Loop playlist
`shuffle`	boolean	No	`false`	Shuffle playback order

Complete Example

```
json
[
  {
    "time": "14:30:00",
    "scene": "Afternoon Show",
    "duration": "01:00:00",
    "name": "Daily Afternoon Broadcast",
    "sources": [
      {
        "type": "media_source",
        "name": "IntroVideo",
        "file": "/media/intros/afternoon.mp4",
        "loop": false,
        "hw_decode": true
      },
      {
        "type": "browser_source",
        "name": "LowerThird",
        "url": "https://graphics.local/lowerthird.html",
        "width": 1920,
        "height": 200,
        "css": "body { background: transparent; }",
        "shutdown_when_hidden": true,
        "fps": 30
      },
      {
        "type": "ffmpeg_source",
        "name": "LiveCamera",
        "input": "rtsp://camera.local:554/stream",
        "buffering_mb": 3,
        "hw_decode": true
      }
    ]
  },
  {
    "time": "15:30:00",
    "scene": "News Segment",
    "duration": "00:30:00",
    "sources": []
  }
]
```

11.2 WebSocket Protocol

Scene Scheduler uses WebSocket for real-time communication between backend and frontend.

Connection

Client initiates connection:

```
WebSocket URL: ws://localhost:8080/ws
Protocol: Standard WebSocket (RFC 6455)
```

Handshake:

```
Client → Server: WebSocket Upgrade request
Server → Client: 101 Switching Protocols
Connection established
```

Message Format

All messages are JSON:

```
json
{
  "type": "messageType",
  "payload": { ... }
}
```

Client → Server Messages

Get current schedule:

```
json
{
  "type": "getSchedule"
}
```

Add event:

```
json
{
  "type": "addEvent",
  "payload": {
    "time": "14:30:00",
    "scene": "SceneName",
    "duration": "01:00:00",
    "sources": [...]
  }
}
```

Edit event:

```
json
{
  "type": "editEvent",
  "payload": {
    "index": 0,
    "event": { ... }
  }
}
```

Delete event:

```
json
{
  "type": "deleteEvent",
  "payload": {
    "index": 0
  }
}
```

Start preview:

```
json
{
  "type": "startPreview",
  "payload": {
    "source": { ... }
  }
}
```

Stop preview:

```
json
{
  "type": "stopPreview",
  "payload": {
    "previewID": "preview_123"
  }
}
```

Server → Client Messages

Schedule updated:

```
json
{
  "type": "scheduleUpdated",
  "payload": {
    "events": [...]
  }
}
```

Current event changed:

```
json
{
  "type": "currentEventChanged",
  "payload": {
    "eventIndex": 0,
    "event": { ... }
  }
}
```

OBS connection status:

```
json
{
  "type": "obsConnectionStatus",
  "payload": {
    "connected": true
  }
}
```

Scene list updated:

```
json
{
  "type": "sceneListUpdated",
  "payload": {
    "scenes": ["Scene 1", "Scene 2", ...]
  }
}
```

Preview started:

```
json
{
  "type": "previewStarted",
  "payload": {
    "previewID": "preview_123",
    "hlsURL": "/hls/preview_123/playlist.m3u8"
  }
}
```

Preview stopped:

```
json
{
  "type": "previewStopped",
  "payload": {
    "reason": "Preview automatically stopped after 30 seconds"
  }
}
```

Error:

```
json
{
  "type": "error",
  "payload": {
    "message": "Error description"
  }
}
```

11.3 Command-Line Tools

hls-generator

Standalone tool for generating HLS preview streams.

Usage:

```
bash
./hls-generator [options]
```

Options:

Flag	Description	Example
`--source-type`	Type of source	`media`, `browser`, `ffmpeg`
`--source-name`	Name for source in OBS	`PreviewSource`
`--source-uri`	URI/path to source	`/path/file.mp4`, `https://...`
`--output-dir`	HLS output directory	`/tmp/hls/preview_123`
`--width`	Video width (px)	`1920`
`--height`	Video height (px)	`1080`
`--duration`	Max duration (seconds)	`30`

Example:

```
bash
./hls-generator \
  --source-type media \
  --source-name TestVideo \
  --source-uri /media/test.mp4 \
  --output-dir /tmp/hls/test \
  --width 1920 \
  --height 1080 \
  --duration 30
```

Output:

```
/tmp/hls/test/
├── playlist.m3u8
├── segment000.ts
├── segment001.ts
└── ...
```

11.4 Configuration Reference

See Section 6 for complete configuration documentation.

Quick reference of all config.json fields:

```
json
{
  "obsWebSocket": {
    "host": "localhost",         // OBS hostname/IP
    "port": 4455,                 // WebSocket port
    "password": "password"        // WebSocket password
  },
  "webServer": {
    "host": "0.0.0.0",           // Bind address
    "port": 8080,                 // HTTP port
    "hlsPath": "hls"             // HLS directory (CORRECT field name)
  },
  "schedule": {
    "jsonPath": "schedule.json",           // Schedule file path
    "scheduleSceneAux": "scheduleSceneAux" // Auxiliary scene name
  },
  "paths": {
    "hlsGenerator": "./hls-generator"      // hls-generator binary path
  },
  "logging": {
    "level": "info",             // debug, info, warn, error
    "format": "text"             // text or json
  }
}
```

11.5 Glossary

Auxiliary Scene (scheduleSceneAux) A hidden OBS scene used by Scene Scheduler for staging sources before they're needed. Sources are preloaded here, then moved to the target scene during transition.

CEF (Chromium Embedded Framework) The embedded browser engine used by OBS to render browser sources. Required for browser_source preview functionality.

Connection ID Unique identifier assigned to each WebSocket client connection. Used to track preview sessions and ensure proper cleanup. Format: `conn_<timestamp>_<random>`.

Duration The length of time an event's sources remain active before cleanup. Specified in HH:MM:SS format.

EventBus Internal pub/sub system that synchronizes state across Scene Scheduler components (scheduler engine, OBS connection, WebSocket clients).

FFMPEG Source OBS source type for network streaming inputs (RTSP, RTMP, SRT, RTP, HLS, etc.).

HLS (HTTP Live Streaming) Streaming protocol used by preview system. Media is segmented into small chunks (.ts files) with a playlist manifest (.m3u8).

Hot-Reload Automatic reloading of schedule.json when file changes are detected, without restarting Scene Scheduler.

Idempotent An operation that can be safely called multiple times without changing the result beyond the initial call. Scene Scheduler's cleanup operations are idempotent.

Media Source OBS source type for local video/audio files (MP4, MKV, MOV, etc.).

Preview Real-time test of a source before adding it to an event. Generates 30-second HLS stream playable in browser.

Preview ID Unique identifier for a preview session. Used in HLS URLs and for tracking/cleanup.

Scene A collection of sources in OBS. Each event switches to a specific scene.

Staging The 30-second window before an event triggers, during which sources are preloaded in the auxiliary scene.

Source Any content element in OBS: media files, browser pages, network streams, images, etc.

VLC Source OBS source type using VLC libraries for media playback. Supports playlists and exotic codecs.

WebSocket Full-duplex communication protocol used by OBS (for control) and Scene Scheduler (for frontend sync).

---

12. Quick Reference Card

Essential Commands

Linux:

```
bash
# Start Scene Scheduler
./scenescheduler

# Start with environment variables
OBS_WS_PASSWORD="secret" ./scenescheduler

# Validate config
jq . config.json

# Validate schedule
jq . schedule.json

# Backup schedule
cp schedule.json schedule.backup.json

# Reset schedule
echo "[]" > schedule.json

# Check if running
ps aux | grep scenescheduler

# Stop
pkill scenescheduler
```

```
# View logs (if redirected)
tail -f scenescheduler.log
```

Windows:

```
cmd
REM Start Scene Scheduler
scenescheduler.exe

REM Start with environment variables
set OBS_WS_PASSWORD=secret
scenescheduler.exe

REM Validate config (requires jq.exe)
jq . config.json

REM Backup schedule
copy schedule.json schedule.backup.json

REM Reset schedule
echo [] > schedule.json

REM Check if running
tasklist | findstr scenescheduler

REM Stop (Ctrl+C or Task Manager)
taskkill /IM scenescheduler.exe /F

REM View logs (open in Notepad)
notepad scenescheduler.log
```

Key Files

File	Purpose	Location
`scenescheduler`	Main executable	Application directory
`config.json`	Configuration	Same as executable
`schedule.json`	Event schedule	Specified in config
`hls-generator`	Preview generator	Specified in config
`hls/`	Preview HLS files	Specified in config

Web Interface URLs

Local Access (same machine):

```
Monitor View:  http://localhost:8080/
Editor View:   http://localhost:8080/editor.html
WebSocket:     ws://localhost:8080/ws
```

Network Access (from other devices):

```
Monitor View:  http://<server-ip>:8080/
Editor View:   http://<server-ip>:8080/editor.html
WebSocket:     ws://<server-ip>:8080/ws

Example:       http://192.168.1.100:8080/
```

Finding server IP: - Linux: `ip addr show | grep inet` - Windows: `ipconfig`

Note: config.json must have `"host": "0.0.0.0"` for network access

Event Time Format

```
Format: HH:MM:SS (24-hour)

Examples:
00:00:00  Midnight
09:30:00  9:30 AM
14:45:30  2:45:30 PM
23:59:59  11:59:59 PM
```

Common Source Types

```
json
// Media file (Linux)
{
  "type": "media_source",
  "name": "Video",
  "file": "/home/user/videos/file.mp4",
  "loop": false
}

// Media file (Windows)
{
  "type": "media_source",
  "name": "Video",
  "file": "C:/Videos/file.mp4",
  "loop": false
}
```

```
// Browser overlay
{
  "type": "browser_source",
  "name": "Overlay",
  "url": "https://example.com/page.html",
  "width": 1920,
  "height": 1080
}

// Network stream
{
  "type": "ffmpeg_source",
  "name": "Camera",
  "input": "rtsp://camera.local/stream"
}
```

Troubleshooting Checklist

Scene Scheduler won't start: - Check config.json syntax: `jq . config.json` - Verify OBS is running - Check OBS WebSocket enabled (Tools → WebSocket Server Settings) - Verify password matches

Events don't trigger: - Check system time: `date` - Verify event time format (HH:MM:SS) - Check schedule loaded (web interface) - Review logs for errors

Sources don't appear: - Verify `scheduleSceneAux` name is correctly configured in config.json - Check file paths are absolute - Test file access: `cat /path/file.mp4 > /dev/null` - Check logs for "failed to create source"

Preview timeout: - Verify hls-generator exists: `ls -la ./hls-generator` - Make executable: `chmod +x hls-generator` - Check HLS directory writable: `touch hls/test; rm hls/test` - Test source (file exists, URL reachable)

Web interface disconnected: - Backend running: `ps aux | grep scenescheduler` - Port accessible: `curl http://localhost:8080` - Check browser console for errors (F12)

Performance Tips

- Use H.264 video codec (universal compatibility)
- Enable hardware decode/encode
- Store media on SSD (not HDD)
- Limit browser sources (high CPU usage)
- Don't schedule events <30s apart
- Use local files (not network mounts)

- Monitor CPU/RAM during staging window
- Optimize browser source JavaScript

Security Checklist

- Set strong OBS WebSocket password
- Use environment variable for password (not config.json)
- Bind web server to localhost if network access not needed
- Configure firewall (allow only necessary ports)

- Set restrictive file permissions:

  ```
  bash
    chmod 600 config.json schedule.json
    chmod 700 hls/
  ```

- Regular backups of schedule.json

- Keep Scene Scheduler and OBS updated

Getting Help

Documentation: - Full manual: Manual English v0.4.md (this document) - Technical specifications available in the docs folder

Before reporting bugs: 1. Check FAQ (Section 10.1) 2. Review relevant troubleshooting section (Section 10) 3. Gather logs and configuration 4. Create minimal reproduction case 5. Include Scene Scheduler version, OBS version, OS

End of Manual English v0.4.md

Scene Scheduler v0.4 - October 28, 2025