SoundCoreHero

SoundCoreHero

Client Documentation API

Information:

API version compatible with SCH v3.6.x and later

Revision Date: June 29, 2026

Hero AV Sp. z o. o.
ul. Stróża Rybna 3,
30-714 Kraków, Poland

sale@SoundCoreHero.com
support@SoundCoreHero.com
partner@SoundCoreHero.com

SoundCoreHero.com

Contents

1 SoundCoreHero

SoundCoreHero is a professional multi-zone audio distribution and management system. It runs as a background service on the device and is controlled entirely through its API — either from a dedicated user interface or directly via HTTP, TCP, or WebSocket connections.

The system handles everything from media playback and live inputs to announcements, signal processing, and hardware output routing. All settings are stored in a **project**, which can be saved, loaded, and exported at any time.

1.1 How audio flows through the system

Audio in SoundCoreHero moves through a fixed signal chain:

```
Players  ▶
Inputs   ▶  Zone  ▶  Speaker  ▶  Device output
DSO      ▶
```

- **Players** are media sources — they play back playlists or network streams.
- **Inputs** are live audio sources connected to the device's physical inputs (microphones, line-level sources, or DSO).
- **Zones** are the mixing stages — each zone blends its connected players, inputs, and announcements into a single mix.
- **Speakers** are the output stages — each speaker applies its own EQ, delay, and limiter, then drives a physical hardware output channel.

Every object in this chain is independent and reusable. A player can feed multiple zones simultaneously. A zone can drive multiple speakers. An input can be routed into several zones at once.

1.2 Key concepts

1.2.1 Players and Playlists

A **Player** is a playback engine. It can play from a **Playlist** (an ordered list of local audio files) or from a live network stream (HTTP/Icecast). Players support shuffle, repeat, fade, and independent volume control.

Each player is assigned to one or more zones.

1.2.2 Inputs and DSO

An **Input** maps one or two physical input channels to an audio signal within the system. Inputs can be configured as microphone (`mic`) or line-level (`line`) sources. Each input has its own gate, compressor, and EQ processing.

Microphone inputs trigger **ducking** in zones — automatically lowering player and line levels when speech is detected.

DSO (*Dźwiękowy System Ostrzegawczy*) is a dedicated emergency input type used for fire alarm and public address emergency systems. When active, DSO can duck all other sources across all connected zones.

1.2.3 Zone mixing

A **Zone** is the core mixing stage. It receives audio from players, inputs, and announcements, and mixes them through separate buses (`player`, `lineInputs`, `micInputs`, `dso`, `announcement`). Each bus has its own volume, mute, and balance controls.

Zones also contain **Duckers** — automatic gain reduction stages that lower certain buses when a signal is detected on another. For example, line inputs can be ducked when a microphone becomes active.

1.2.4 Speaker outputs

A **Speaker** is a physical output channel. It receives the mixed audio from its connected zone and applies output processing (EQ, delay, limiter) before sending audio to a hardware output. Speakers can be paired as stereo left/right or used as mono center channels.

1.2.5 Announcement clips

An **Announcement** is a one-shot audio clip that plays at elevated priority. When triggered in a zone, it temporarily ducks all other audio sources, plays the clip through a dedicated announcement bus, then restores normal levels. Multiple announcements can queue up; priority determines playback order.

1.3 Signal processing

Processing is available at several points in the signal chain:

Stage	Where	Available processors
Input	Per input	EQ, Gate, Compressor
Output	Per speaker	EQ, Delay, Limiter
Zone	Per bus	Volume, Mute, Balance, Ducker

Processor settings can be saved as **Presets** and reused across multiple inputs or speakers.

1.4 Automation

1.4.1 Actions and Action Groups

An **Action** is a scripted sequence of steps — playing a sound, changing a volume, triggering a zone, or calling an external HTTP endpoint. Actions can be triggered manually through the API or wired up to external events.

Action Groups are named collections of actions that can be triggered together as a single unit.

1.4.2 External action triggers

External Actions allow third-party systems to trigger actions in SoundCoreHero over the network using TCP, UDP, HTTP, or Modbus. This is used for integration with building management systems, access control, scheduling systems, and similar automation platforms.

1.5 Projects and configuration

All objects — zones, players, inputs, speakers, announcements, actions, and so on — are stored in a **Project**. A project is a complete snapshot of the system configuration.

Projects can be:

- Saved and loaded to switch between full configurations
- Exported as a file for backup or transfer to another device
- Imported to restore a saved configuration

The **File Manager** provides access to the audio file storage on the device, where music, announcement clips, and other media are kept.

1.6 Control interfaces

SoundCoreHero exposes several interfaces for external control:

Interface	Description
REST API	Primary interface — normally exposed over HTTPS. See API Overview.
WebSockets	Real-time push events and audio streaming. See WebSockets.
TCP Server	Optional integration interface for external devices and legacy systems. See TCP Server.
Serial Port	Same JSON protocol, delivered over a serial connection. See Serial Port Server.

Please look into the API Overview for the details.

1.7 Security

Access to the API can be restricted by user account. Each user is assigned a set of permissions that control which resources and operations they can access. See Security and Users Manager for details.

1.8 TCP / UDP Ports

Port	Protocol	Description	Additional Info
123	UDP	NTP sync	If private NTP server is used
443	TCP		
502	TCP	Default Modbus external devices	Configurable
7623	TCP	DANTE configuration service	
7878	TCP	SCH HTTP API	Only if plain HTTP server is enabled
7879	TCP	SCH Websocket	Only if plain HTTP server is enabled
7979	TCP	SCH Audio/Mic Websocket	Only if plain HTTP server is enabled
9010	UDP	SCH Controller Scan	Only if Controllers are used
10055	TCP	SCH Controller Manager	Only if Controllers are used
39200	TCP	TCP Server	Only if plain TCP server is enabled
39300	TCP	TLS TCP Server	

2 API

This document describes the HTTP API conventions shared across all endpoints. For resource-specific endpoints, refer to the dedicated files below.

2.1 Resources

- Security — authentication and authorization
- Players — media sources (playlists, streams)
- Inputs — physical audio inputs and DSO
- Zones — mixing and routing stages
- Speakers — physical output channels
- Playlists — ordered track lists
- Announcements — priority one-shot clips
- Presets — reusable processor settings
- Actions — scripted sequences
- Action Groups — reusable sets of actions
- External Actions — HTTP and network triggers
- Scheduler — time-based automatic action triggers
- Triggers — hardware input monitoring and action automation
- WebSockets — real-time messaging and events
- WebSockets Audio — real-time audio streaming
- Subscriptions — subscribing for webSocket meters and audio streaming
- File Manager — file storage and retrieval
- Users Manager — user accounts and permissions
- Project Manager — project and workspace management
- System — system-level operations
- Network — network configuration
- Software Update — firmware and software updates
- Logger — logging and diagnostics
- TCP Server — TCP-based integrations
- Serial Port Server — serial device integration

2.2 Transport

Request and response bodies are **JSON**.

All HTTPS endpoints use **POST** unless noted otherwise.

Out of the box, SoundCoreHero only accepts encrypted connections. Plain HTTP, WebSockets, TCP, and Serial Port access may be enabled for legacy integrations from the System settings.

SoundCoreHero interfaces and their configuration fields:

Service	Address	Configuration field	Default state
Web Interface	`https://<device-ip>`		always on
REST API	`https://<device-ip>/api/<endpoint>`		always on
Control WebSocket	`wss://<device-ip>/ws/`		always on
Audio WebSocket	`wss://<device-ip>/ws-audio/`		always on
TLS TCP control	`<device-ip>:3936`	`tcpServerEnabled`	Disabled
Plain REST API	`http://<device-i`	`httpServerPlainHttpEnabled`	Disabled
Plain Control WebSocket	`ws://<device-ip>`	`httpServerPlainHttpEnabled`	Disabled
Plain Audio WebSocket	`ws://<device-ip>`	`httpServerPlainHttpEnabled`	Disabled
Plain TCP control	`<device-ip>:3926`	`tcpServerEnabled` and `tcpServerPlainTcpEnabled`	Disabled
Serial Port	Configured serial device path	`serialPortEnabled`	Disabled

2.3 Authentication

By default, HTTP, TCP, and Serial Port requests must be authenticated unless the endpoint is explicitly public.

Administrators can disable authentication entirely for trusted environments. They can also disable API key usage while keeping session-based login enabled.

Public unauthenticated endpoints:

- `/usersmanager-login`
- `/usersmanager-reset-password`

To authenticate request sender must provide either of these credentials:

- `sessionId` — intended for Web UI clients obtained after a successful login
- `apiKey` — intended for machine-to-machine integrations

If API key usage is disabled in the Security settings, only `sessionId` authentication is accepted.

Trigger-code (action) strings on TCP and Serial Port do not follow authentication rules. Trigger codes can be enabled or disabled in the Security settings.

For HTTP requests provide credentials in headers:

- `X-Session-Id`
- `X-Api-Key`

For TCP and Serial Port JSON requests, provide credentials as top-level fields next to `url`, `method`, and `params`.

Examples:

```
{
    "url": "zone-get-status-all",
    "method": "GET",
    "apiKey": "sch_..."
}
```

or

```
{
    "url": "zone-get-status-all",
    "method": "GET",
    "sessionId": "<sessionId>"
}
```

2.4 Identifying objects

Most endpoints accept either an `...Id` or an `...Name` field to target objects:

- Single item: `playerId` or `playerName`, `zoneId` or `zoneName`, etc.
- Multiple items: `playerIds` or `playerNames`, `zoneIds` or `zoneNames`, etc.

For `...Ids` / `...Names` fields you can pass:

- an **array** of UUIDs or name strings, OR
- the string `"all"` to target every object of that type

Examples:

```
{ "playerNames": ["Player A", "Player B"] }
```

```
{ "zoneIds": "all" }
```

2.5 Status codes

- **200** — success; body is either empty or contains a result (e.g. `{ "id": "..." }`)
- **400** — error; body contains `{ "error": "..." }`
- **401** — authentication failed; body contains `{ "error": "..." }`

Some endpoints return HTTP 200 even on failure, with the error embedded in the body. These cases are noted in the individual endpoint descriptions.

2.6 Sessions and receivers

WebSocket-based subscriptions (meters, audio frames) are scoped per `(sessionId, receiverId)` pair.

- `sessionId` identifies the logged-in user session and is supplied through request authentication for HTTP subscription calls.
- `receiverId` is the receiver identifier. Any client may generate a stable UUID and use it to distinguish one receiver from another within the same session (for example, different browser tabs or devices).

To receive subscription data, the client must first establish the matching WebSocket connection using the same `(sessionId, receiverId)`, then call the HTTP subscribe endpoint.

3 HTTPS & Secure Communication

SoundCoreHero uses TLS encryption to protect communication between clients and the device. By default, only HTTPS and WSS (WebSocket Secure) are enabled. An administrator may selectively enable additional, unencrypted interfaces when required.

Authentication is enforced separately from transport encryption:

- Web UI clients log in with `email` and `password`, then send the returned `sessionId` in HTTP headers.
- Machine-to-machine clients should use per-user API keys, sent in HTTP headers or as top-level fields in TCP or Serial JSON.
- Only the login and password-reset endpoints are unauthenticated.
- Plain trigger-code strings on TCP and Serial are unauthenticated. Administrators can enable or disable trigger codes from the Security settings.
- Authentication can be left enforced, or turned off for trusted environments.
- API key usage can be enabled or disabled independently.

When to enable unencrypted interfaces:

- **Plain HTTP / WS** — useful during development, or when integrating with legacy controllers that do not support TLS.
- **Plain TCP** — required by some third-party control systems (e.g. Crestron, Extron, AMX) that communicate over raw TCP.
- **Serial Port** — needed when the controlling device is connected via RS-232 cable.

Even when unencrypted interfaces are active, HTTPS and WSS remain available and should be preferred whenever the connecting client supports TLS.

3.1 SSL/TLS certificate

SoundCoreHero ships with a self-signed certificate so that HTTPS works immediately after installation. The certificate can be downloaded from the admin UI.

3.1.1 Downloading the certificate

1. Log in to the Web UI as an admin.
2. Navigate to **Settings** → **Security**.
3. Click **Download Certificate**.
4. Install the certificate in your operating system or browser trust store.

3.1.2 Browser warnings

Because the default certificate is self-signed, browsers will show a security warning on first connection. After installing the certificate into the system or browser trust store, the warning will disappear. In controlled environments (AV installations, kiosks) this is typically done once during commissioning.

3.2 Additional configuration options

Security configuration is managed by system api endpoints documented in System.

Setting	Key	Description
Api Key	apiKeyEnabled	When `true`, clients can authenticate with API keys. When `false`, only session-based authentication is accepted.
Authentication	authenticationEnabled	When `true`, all non-public endpoints require authentication. When `false`, authentication is not required for any endpoint.
Trigger Codes	triggerCodesEnabled	When `true`, plain trigger-code strings on TCP and Serial Port can execute actions without JSON authentication. When `false`, trigger codes are disabled and cannot be used.

3.3 Best practices

1. Keep the default HTTPS and WSS-only setup unless an integration explicitly requires a plain interface.
2. Keep authentication enabled unless the device is isolated on a trusted network segment.
3. Disable API keys if the deployment should accept session-based logins only.
4. Use strong passwords and rotate API keys when an integration changes ownership.
5. Disable plain interfaces again after commissioning or testing is complete.
6. Treat Serial access as privileged physical access.

4 Players

A **Player** is a media source — it can play back a local playlist or a network stream. Players are routed into one or more **Zones**, where their audio is mixed with inputs and announcements.

Each player can be independently controlled (play, pause, stop, next, etc.), and carries its own volume and fade settings. A player with `sourceType: "playlist"` reads tracks from an assigned **Playlist** object. A player with `sourceType: "stream"` connects to a live HTTP/Icecast stream.

Connections in the system:

```
Playlist ▶ Player ▶ Zone(s)
Stream   ▶
```

4.1 Player object

Every player in the system is identified by a UUID. The full player object returned by `/player-get-status-all` and pushed via WebSocket has the following structure:

```
{
  "name": "Player 1",
  "streamUrl": null,
  "sourceType": "playlist",
  "volume": 0.0,
  "progress": 122.62,
  "duration": 193.42,
  "remaining": 70.80,
  "shuffle": false,
  "muted": false,
  "repeatOne": false,
  "repeatAll": true,
  "playlist": "<playlistUuid>",
  "fade": 5.0,
  "track": {
    "file": {
      "position": 0,
      "relativePath": "filename.flac",
      "absolutePath": "/full/path/to/filename.flac",
      "parentPath": "/Folder/",
      "pathValid": true
    }
  },
  "state": "Playing",
  "connections": {
    "zones": ["<zoneUuid>"]
  },
  "order": 0,
  "signalActive": false
}
```

4.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the player.
`streamUrl`	string | null	Stream URL when `sourceType` is `"stream"`, otherwise `null`.
`sourceType`	string	`"playlist"` or `"stream"`.
`volume`	float	Output level in dBFS (0.0 = unity).
`progress`	float | null	Current playback position in seconds, or `null` when no track is loaded or source is a stream.
`duration`	float | null	Total track duration in seconds, or `null` when no track is loaded or source is a stream.
`remaining`	float | null	Remaining playback time in seconds, or `null` when no track is loaded or source is a stream.
`shuffle`	boolean	Whether shuffle mode is enabled.
`muted`	boolean	Whether the player output is muted.
`repeatOne`	boolean	Whether single-track repeat is enabled.
`repeatAll`	boolean	Whether playlist repeat is enabled.
`playlist`	string | null	UUID of the assigned playlist, or `null`.
`fade`	float	Crossfade/fade time in seconds.
`track`	object | null	Currently loaded track, or `null` when no track is loaded (see `track`).
`state`	string	Playback state: `"Stopped"`, `"Paused"`, or `"Playing"`.
`connections`	object	Zone routing connections (see `connections`).
`order`	integer	Position in the player list (0-based).
`signalActive`	boolean	`true` when the player output level exceeds the signal-active threshold.

4.1.2 `track`

The `track` object describes the currently loaded track. It is `null` when no track is loaded.

Field	Type	Description
`file`	object	File metadata for the current track (see below).

`track.file`

Field	Type	Description
`position`	integer	Zero-based index of the track in the playlist.
`relativePath`	string	Path relative to the playlist root.

Field	Type	Description
`absolutePath`	string	Full absolute filesystem path.
`parentPath`	string	Parent folder path.
`pathValid`	boolean	`true` when the file exists at the given path.

4.1.3 `connections`

Field	Type	Description
`zones`	string[]	UUIDs of connected zones.

4.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/player-get-status-all`
- `/player-create`
- `/player-remove`
- `/players-modify`
- `/players-action`
- `/player-set-zones`
- `/player-add-zones`
- `/player-remove-zones`
- `/player-set-playlist`
- `/player-remove-playlist`
- `/player-set-track`
- `/player-set-playback-time`

4.2.1 `/player-get-status-all`

Method: `GET`

Params:

```
None
```

Response:

```
{
  "players": {
    "<playerUuid>": { ... },
    "<playerUuid>": { ... }
  }
}
```

Each value is a full player object (see Player object above).

4.2.2 `/player-create`

Creates a new player.

Params:

```
{
  "name": "Player 1",
  "sourceType": "stream",
  "streamUrl": "http://example.com/radio.mp3",

  "volume": 0.0,
  "muted": false,
  "fadeTime": 5.0,
  "repeatOne": false,
  "repeatAll": false,
  "shuffle": false
}
```

Notes:

- Required: `name`, `sourceType`.
- `sourceType` must be either `"stream"` or `"playlist"`.
- If `sourceType` is `"stream"`, `streamUrl` is required.
- Optional: `volume` (defaults to `0.0`), `muted` (defaults to `false`), `fadeTime` (defaults to `5.0`), `repeatOne` (defaults to `false`), `repeatAll` (defaults to `false`), `shuffle` (defaults to `false`).
- `name` must be unique among all players.

Response:

```
{ "id": "<playerUuid>" }
```

4.2.3 `/player-remove`

Params (id or name):

```
{ "playerId": "<playerUuid>" }
```

Notes:

- The player is disconnected from all zones upon removal.
- Controllers referencing this player will have their binding cleared.
- Remaining players have their `order` values refreshed.

4.2.4 `/players-modify`

Modifies one or multiple players.

Params (all optional aside from selection):

```
{
  "playerIds": ["<playerUuid>"],
  "rename": "New name",
  "order": 0,

  "volume": 0.0,
  "muted": false,
  "fadeTime": 5.0,
  "repeatOne": false,
  "repeatAll": false,
  "shuffle": false,

  "sourceType": "stream",
  "streamUrl": "http://example.com/radio.mp3",

  "track": 0
}
```

Notes:

- `rename` can only be applied to a single player. The new name must be unique.
- `order` can only be applied to a single player. Value must be within `0..players.len()-1`.
- `track` sets the current track position (index) in the playlist.

Playlist assignment via `players-modify`:

- Set playlist for the selected players:

```
{ "playerIds": ["<playerUuid>"], "playlistId": "<playlistUuid>" }
```

- Unset playlist (must pass JSON null):

```
{ "playerIds": ["<playerUuid>"], "playlistId": null }
```

4.2.5 `/players-action`

Performs a transport action on one or multiple players.

Params:

```
{
  "playerIds": "all",
  "action": "play"
}
```

Allowed actions:

- `play`
- `pause`
- `stop`
- `next`
- `previous`
- `previousTrack`

4.2.6 `/player-set-zones`

Sets the player connections to the provided zones (removes other zone connections).

Params:

```
{
  "playerId": "<playerUuid>",
  "zoneIds": ["<zoneUuid1>", "<zoneUuid2>"]
}
```

4.2.7 `/player-add-zones`

Adds zones to the player, without disconnecting existing zones.

Params:

```
{
  "playerId": "<playerUuid>",
  "zoneIds": ["<zoneUuid>"]
}
```

4.2.8 `/player-remove-zones`

Removes zones from the player, without affecting other connections.

Params:

```
{
  "playerId": "<playerUuid>",
  "zoneIds": ["<zoneUuid>"]
}
```

4.2.9 `/player-set-playlist`

Convenience endpoint that assigns a playlist to a single player.

Notes:

- The implementation uses the *first* player from `playerIds` / `playerNames`.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "playerIds": ["<playerUuid>"]
}
```

4.2.10 `/player-remove-playlist`

Removes a playlist assignment from a single player (while preserving the playlist for other players that still use it).

Params:

```
{
  "playlistId": "<playlistUuid>",
  "playerId": "<playerUuid>"
}
```

4.2.11 `/player-set-track`

Sets the current track position (index) in the player.

Params:

```
{
  "playerId": "<playerUuid>",
  "position": 0
}
```

4.2.12 `/player-set-playback-time`

Sets the playback time position (seconds) for the player.

Params:

```
{
  "playerId": "<playerUuid>",
  "time": 12.5
}
```

5 Inputs

An **Input** represents a physical audio input on the device (e.g. a microphone or line-level source). Each input is assigned to one or more device channels via a `channelsFormat`, which describes whether the signal is mono or stereo and which hardware channels it reads from.

Inputs are routed into **Zones**, where they are mixed with players and announcements. Each input carries its own volume, mute, EQ, gate, and compressor settings.

Input modes:

Mode	Description
`mic`	Microphone-level input — contributes to the `micInputs` mixer bus in a zone.
`line`	Line-level input — contributes to the `lineInputs` mixer bus in a zone.

Connections in the system:

```
Device channels ▶ Input ▶ Zone(s)
```

5.1 Input object

Every input in the system is identified by a UUID. The full input object returned by `/input-get-status-all` and pushed via WebSocket has the following structure:

```
{
  "name": "Mic 1",
  "mode": "mic",
  "channelsFormat": { "mono": { "channel": 1 } },
  "order": 0,
  "volume": 0.0,
  "muted": false,
  "eq": null,
  "gate": null,
  "compressor": null,
  "signalActive": false,
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,
  "presetGateId": "<presetGateUuid>",
  "presetGateBypassed": true,
  "presetCompressorId": "<presetCompressorUuid>",
  "presetCompressorBypassed": true,
  "connections": {
    "zones": ["<zoneUuid>"]
  }
}
```

5.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the input.
`mode`	string	Input mode: `"mic"` or `"line"` (see Input modes above).
`channelsFormat`	object	Device channel assignment (see `channelsFormat`).
`order`	integer	Position in the input list (0-based).
`volume`	float	Output level in dBFS (0.0 = unity).
`muted`	boolean	Whether the input output is muted.
`eq`	object | null	Custom EQ configuration, or `null` when a preset EQ is assigned (see `eq`).
`gate`	object | null	Custom gate configuration, or `null` when a preset gate is assigned (see `gate`).
`compressor`	object | null	Custom compressor configuration, or `null` when a preset compressor is assigned (see `compressor`).
`signalActive`	boolean	`true` when the input output level exceeds the signal-active threshold.
`presetEqId`	string | null	UUID of the assigned EQ preset, or `null` when using custom EQ.
`presetEqBypassed`	boolean	Whether the EQ stage is bypassed.
`presetGateId`	string | null	UUID of the assigned gate preset, or `null` when using a custom gate.
`presetGateBypassed`	boolean	Whether the gate stage is bypassed.
`presetCompressorId`	string | null	UUID of the assigned compressor preset, or `null` when using a custom compressor.
`presetCompressorBypassed`	boolean	Whether the compressor stage is bypassed.
`connections`	object	Zone routing connections (see `connections`).

5.1.2 `channelsFormat`

Describes which hardware device channels the input reads from.

Variant	JSON value	Description
none	`{}`	No channels assigned.
mono	`{ "mono": { "channel": <number> } }`	Single device channel.
stereo	`{ "stereo": { "left": <number>, "right": <number> } }`	Stereo pair of device channels.

5.1.3 `eq`

`eq` is `null` when a preset EQ is assigned (`presetEqId` is set). Otherwise it contains:

Field	Type	Description
`enabled`	boolean	Whether EQ processing is active.
`bands`	array	Array of EQ band objects (same structure as in speakers.md).

Setting `eq` directly clears `presetEqId`. Setting `presetEqId` switches back to preset-based EQ.

5.1.4 `gate`

`gate` is `null` when a preset gate is assigned (`presetGateId` is set). Otherwise it contains the custom gate parameters. Setting `gate` directly clears `presetGateId`. Setting `presetGateId` switches back to preset-based gating.

5.1.5 `compressor`

`compressor` is `null` when a preset compressor is assigned (`presetCompressorId` is set). Otherwise it contains the custom compressor parameters. Setting `compressor` directly clears `presetCompressorId`. Setting `presetCompressorId` switches back to preset-based compression.

5.1.6 `connections`

Field	Type	Description
`zones`	string[]	UUIDs of connected zones.

5.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/input-get-status-all`
- `/input-create`
- `/input-remove`
- `/inputs-modify`
- `/input-set-zones`
- `/input-add-zones`
- `/input-remove-zones`
- `/input-set-channels`
- `/input-remove-channels`
- `/dso-get-status`
- `/dso-modify`

5.2.1 `/input-get-status-all`

Method: `GET`

Returns all inputs.

Params:

```
None
```

Response:

```
{
  "inputs": {
    "<inputUuid>": { ... },
    "<inputUuid>": { ... }
  }
}
```

Each value is a full input object (see Input object above).

5.2.2 `/input-create`

Creates a new input.

Params:

```
{
  "name": "Mic 1",
  "mode": "mic",

  "volume": 0.0,
  "muted": false,

  "channelsFormat": { "mono": { "channel": 1 } },

  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,

  "presetGateId": "<presetGateUuid>",
  "presetGateBypassed": true,

  "presetCompressorId": "<presetCompressorUuid>",
  "presetCompressorBypassed": true
}
```

Notes:

- Required: `name`, `mode`, `muted`, `channelsFormat`.
- Optional: `volume` (defaults to `0.0`), `presetEqId`, `presetEqBypassed` (defaults to `true`), `presetGateId`, `presetGateBypassed` (defaults to `true`), `presetCompressorId`, `presetCompressorBypassed` (defaults to `true`).
- `channelsFormat` must not overlap existing input channels (and DSO channels when enabled).
- The total number of used input channels (including DSO) must not exceed the device channel limit.
- `name` must be unique among all inputs.

Response:

```
{ "id": "<inputUuid>" }
```

5.2.3 `/input-remove`

Params:

```
{ "inputId": "<inputUuid>" }
```

Notes:

- The input is disconnected from all zones upon removal.
- Remaining inputs have their `order` values refreshed.

5.2.4 `/inputs-modify`

Modifies one or multiple inputs.

Common params:

```
{
  "inputIds": ["<inputUuid>"],
  "rename": "New name",
  "order": 0,

  "muted": false,
  "volume": 0.0,

  "mode": "mic",
  "channelsFormat": { "stereo": { "left": 1, "right": 2 } },

  "eq": { "enabled": true, "bands": [] },
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,

  "gate": { "threshold": -40.0 },
  "presetGateId": "<presetGateUuid>",
  "presetGateBypassed": true,

  "compressor": { "threshold": -20.0 },
  "presetCompressorId": "<presetCompressorUuid>",
  "presetCompressorBypassed": true
}
```

Notes:

- All params are optional (aside from the selection).
- `rename` can only be applied to a single input (`inputId` / `inputName`). The new name must be unique.
- `order` can only be applied to a single input. Value must be within `0..inputs.len()-1`.
- `channelsFormat` can be modified only for a single input at a time. Must not overlap other inputs or enabled DSO.
- `mode` changes will cause zones to re-connect this input using the new mode.

5.2.5 `/input-set-zones`

Sets the input connections to the provided zones (removes other zone connections).

Params:

```
{
  "inputId": "<inputUuid>",
  "zoneIds": ["<zoneUuid>"]
}
```

5.2.6 `/input-add-zones`

Adds the input to zones without altering existing connections.

Params:

```
{
  "inputId": "<inputUuid>",
  "zoneIds": ["<zoneUuid>"]
}
```

5.2.7 `/input-remove-zones`

Removes the input from the provided zones.

Params:

```
{
  "inputId": "<inputUuid>",
  "zoneIds": ["<zoneUuid>"]
}
```

5.2.8 `/input-set-channels`

Assigns device channels to a single input. Any other input currently using the requested channels will have its channels disconnected (set to `none`).

Params:

```
{
  "inputId": "<inputUuid>",
  "channelsFormat": { "mono": { "channel": 1 } }
}
```

Notes:

- Channels must be within the device range (`1..totalChannels`).
- The total number of used input channels must not exceed the device channel limit.

5.2.9 `/input-remove-channels`

Disconnects device channels from the input (sets `channelsFormat` to `none`).

Params:

```
{ "inputId": "<inputUuid>" }
```

5.3 DSO

DSO (Polish: *Dźwiękowy System Ostrzegawczy* — emergency microphone / public address emergency source) is a special, singleton input used for emergency announcements. It behaves similarly to a regular input but is limited to mono or no channel assignment. DSO has its own dedicated mixer bus within each zone (`dso` bus), separate from regular line and mic inputs.

DSO can be enabled or disabled globally. When enabled, its assigned device channel must not overlap any regular input.

Connections in the system:

```
Device channel ▶ DSO ▶ Zone(s)  (via the dso mixer bus)
```

5.4 DSO object

The DSO object returned by `/dso-get-status` has the following structure:

```
{
  "enabled": true,
  "channelsFormat": { "mono": { "channel": 1 } },
  "volume": 0.0,
  "muted": false,
  "eq": null,
  "gate": null,
  "compressor": null,
  "signalActive": false,
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,
  "presetGateId": "<presetGateUuid>",
  "presetGateBypassed": true,
  "presetCompressorId": "<presetCompressorUuid>",
  "presetCompressorBypassed": true
}
```

5.4.1 DSO Field reference

Field	Type	Description
`enabled`	boolean	Whether DSO is globally active.
`channelsFormat`	object	Device channel assignment — `none` (`{}`) or `mono` only (see `channelsFormat`).
`volume`	float	Output level in dBFS (0.0 = unity).
`muted`	boolean	Whether the DSO output is muted.
`eq`	object | null	Custom EQ configuration, or `null` when a preset EQ is assigned (see `eq`).
`gate`	object | null	Custom gate configuration, or `null` when a preset gate is assigned (see `gate`).

Field	Type	Description
`compressor`	object | null	Custom compressor configuration, or `null` when a preset compressor is assigned (see `compressor`).
`signalActive`	boolean	`true` when the DSO output level exceeds the signal-active threshold.
`presetEqId`	string | null	UUID of the assigned EQ preset, or `null` when using custom EQ.
`presetEqBypassed`	boolean	Whether the EQ stage is bypassed.
`presetGateId`	string | null	UUID of the assigned gate preset, or `null` when using a custom gate.
`presetGateBypassed`	boolean	Whether the gate stage is bypassed.
`presetCompressorId`	string | null	UUID of the assigned compressor preset, or `null` when using a custom compressor.
`presetCompressorBypassed`	boolean	Whether the compressor stage is bypassed.

5.4.2 `/dso-get-status`

Method: `GET`

Returns the DSO object.

Params:

```
None
```

Response:

```
{
  "dso": { ... }
}
```

The value is a full DSO object (see DSO object above).

5.4.3 `/dso-modify`

Params (all optional):

```
{
  "enabled": true,
  "channelsFormat": { "mono": { "channel": 1 } },

  "muted": false,
  "volume": 0.0,

  "eq": { "enabled": true, "bands": [] },
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,

  "gate": { "threshold": -40.0 },
  "presetGateId": "<presetGateUuid>",
  "presetGateBypassed": true,

  "compressor": { "threshold": -20.0 },
  "presetCompressorId": "<presetCompressorUuid>",
  "presetCompressorBypassed": true
}
```

Notes:

- DSO `channelsFormat` must be `none` (`{}`) or `mono` (stereo is rejected).
- If DSO is enabled, the chosen channel must not overlap other inputs.
- When `enabled` and `channelsFormat` are both provided, the channel validation uses the new format.

6 Zones

A **Zone** represents an independent audio mixing and routing stage. It receives audio from **Players**, **Inputs** (line and mic), and **DSO**, mixes them through separate mixer buses, and routes the combined output to one or more **Speakers**.

Each zone has three optional **Ducker** stages that automatically reduce player/line volume when a mic or DSO signal is detected:

- `lineInputs` ducker — ducks line inputs
- `micInputs` ducker — ducks mic inputs
- `dso` ducker — ducks DSO

Zones also support per-source **mixer** gain and mute controls, allowing fine-grained level balancing within the zone.

A zone can be assigned a **Controller** to enable hardware control surfaces.

Connections in the system:

```
Player(s) ▶
Input(s)  ▶ Zone ▶ Speaker(s)
DSO       ▶
```

6.1 Zone object

Every zone in the system is identified by a UUID. The full zone object returned by `/zone-get-status-all` and pushed via WebSocket has the following structure:

```
{
  "name": "Zone 1",
  "order": 0,
  "mixers": {
    "master":       { "muted": false, "balance": 0, "volume": -3.5 },
    "player":       { "muted": false, "balance": 0, "volume": -1.0 },
    "dso":          { "muted": false, "balance": 0, "volume": 1.5 },
    "announcement": { "muted": false, "balance": 0, "volume": 0.0 },
    "lineInputs": [
      { "inputId": "<inputUuid>", "muted": false, "balance": 0, "volume": 0.0 }
    ],
    "micInputs": [
      { "inputId": "<inputUuid>", "muted": false, "balance": 0, "volume": 0.0 }
    ]
  },
  "duckers": {
    "lineInputs": null,
    "micInputs": null,
    "dso": null
  },
  "linePresetDuckerId": "<presetDuckerUuid>",
  "micPresetDuckerId": "<presetDuckerUuid>",
  "dsoPresetDuckerId": "<presetDuckerUuid>",
  "linePresetDuckerBypassed": false,
  "micPresetDuckerBypassed": false,
  "dsoPresetDuckerBypassed": false,
  "announcementInfo": {
    "announcementState": "Stopped",
    "announcement": null,
    "announcementDuration": null,
    "announcementProgress": null
  },
  "connections": {
    "player": "<playerUuid>",
    "lineInputs": ["<inputUuid>"],
    "micInputs": [],
    "speakers": ["<speakerUuid>"]
  },
  "signalActive": false
}
```


6.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the zone.
`order`	integer	Position in the zone list (0-based).
`mixers`	object	Mixer buses for volume, balance, and mute control (see `mixers`).
`duckers`	object	Optional ducker stages for automatic volume reduction (see `duckers`).
`linePresetDuckerId`	string | null	UUID of the assigned line input ducker preset, or `null`.

Field	Type	Description
`micPresetDuckerId`	string | null	UUID of the assigned mic input ducker preset, or `null`.
`dsoPresetDuckerId`	string | null	UUID of the assigned DSO ducker preset, or `null`.
`linePresetDuckerBypassed`	boolean	Whether the line input ducker stage is bypassed.
`micPresetDuckerBypassed`	boolean	Whether the mic input ducker stage is bypassed.
`dsoPresetDuckerBypassed`	boolean	Whether the DSO ducker stage is bypassed.
`announcementInfo`	object	Current announcement playback state and details (see `announcementInfo`).
`connections`	object	Audio routing and device connections (see `connections`).
`signalActive`	boolean	`true` when the zone output level exceeds -60 dBFS threshold.

6.1.2 `mixers`

Each mixer bus controls the level of one audio source feeding into the zone.

Field	Type	Description
`volume`	float	Level in dBFS (0.0 = unity).
`balance`	integer	Stereo balance. 0 = centre.
`muted`	boolean	Whether this bus is muted.

Fixed buses: `master`, `player`, `dso`, `announcement`.

Per-input arrays: `lineInputs`, `micInputs` — each element adds an `inputId` field identifying the routed input.

6.1.3 `duckers`

Each ducker entry (`lineInputs`, `micInputs`, `dso`) is either `null` (when a preset ducker is assigned) or an object with custom ducker parameters:

Field	Type	Description
`threshold`	float	Sidechain level (dBFS) above which ducking engages.
`range`	float	Amount of gain reduction applied (dB, ≤ 0).
`attack`	integer	Attack time in ms.
`hold`	integer	Hold time in ms.
`release`	integer	Release time in ms.

The corresponding `linePresetDuckerId` / `micPresetDuckerId` / `dsoPresetDuckerId` fields hold the UUID of the assigned preset ducker (or `null` when using custom parameters). The `*Bypassed` fields control whether the ducker stage is skipped.

6.1.4 `announcementInfo`

Field	Type	Description
`announcementState`	string	`"Stopped"` or `"Playing"`.
`announcement`	object | null	Details of the currently playing announcement (see below). `null` when stopped.
`announcementDuration`	float | null	Total duration in seconds.
`announcementProgress`	float | null	Current playback position in seconds.

When an announcement is playing, the `announcement` object is not `null` and contains details about the current announcement. For a full description of the announcement object fields, refer to announcement.md.

6.1.5 `connections`

Field	Type	Description
`player`	string | null	UUID of the connected player, or `null`.
`lineInputs`	string[]	UUIDs of connected line inputs.
`micInputs`	string[]	UUIDs of connected mic inputs.
`speakers`	string[]	UUIDs of connected speakers.

6.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/zone-get-status-all`
- `/zone-create`
- `/zone-remove`
- `/zones-modify`
- `/zone-set-players`
- `/zone-add-players`
- `/zone-remove-players`
- `/zone-set-inputs`
- `/zone-add-inputs`
- `/zone-remove-inputs`
- `/zone-set-speakers`
- `/zone-add-speakers`
- `/zone-remove-speakers`
- `/zone-announcement-abort`

6.2.1 `/zone-get-status-all`

Method: `GET`

Returns all zones.

Params:

```
None
```

Response:

```
{
  "zones": {
    "<zoneUuid>": { ... },
    "<zoneUuid>": { ... }
  }
}
```

Each value is a full zone object (see Zone object above).

6.2.2 `/zone-create`

Creates a zone.

Params:

```
{
  "name": "Zone 1",

  "linePresetDuckerId": "<presetDuckerUuid>",
  "linePresetDuckerBypassed": false,

  "micPresetDuckerId": "<presetDuckerUuid>",
  "micPresetDuckerBypassed": false,

  "dsoPresetDuckerId": "<presetDuckerUuid>",
  "dsoPresetDuckerBypassed": false
}
```

All ducker fields are optional and default to the nil preset with bypass disabled.

Response:

```
{ "id": "<zoneUuid>" }
```

6.2.3 `/zone-remove`

Removes a zone. Connected players, speakers, and controllers are automatically disconnected.

Params:

```
{ "zoneId": "<zoneUuid>" }
```

6.2.4 `/zones-modify`

Modifies one or multiple zones. All fields except the zone selector are optional — only the provided fields are updated.

Params:

```
{
  "zoneIds": ["<zoneUuid>"],
  "rename": "New zone name",
  "order": 0,

  "linePresetDuckerId": "<presetDuckerUuid>",
  "linePresetDuckerBypassed": false,

  "micPresetDuckerId": "<presetDuckerUuid>",
  "micPresetDuckerBypassed": false,

  "dsoPresetDuckerId": "<presetDuckerUuid>",
  "dsoPresetDuckerBypassed": false,

  "duckers": {
    "lineInputs": { "threshold": -20.0, "range": -36.0, "attack": 500, "hold": 1000, "release": 500 },
    "micInputs":  { "threshold": -20.0, "range": -36.0, "attack": 500, "hold": 1000, "release": 500 },
    "dso":        { "threshold": -20.0, "range": -36.0, "attack": 500, "hold": 1000, "release": 500 }
  },

  "mixers": {
    "master":       { "volume": -3.5, "balance": 0, "muted": false },
    "player":       { "volume": -1.0, "balance": 0, "muted": false },
    "dso":          { "volume": 0.0,  "balance": 0, "muted": false },
    "announcement": { "volume": 0.0,  "balance": 0, "muted": false },

    "lineInputs": [
      { "inputId": "<inputUuid>", "volume": 0.0, "balance": 0, "muted": false }
    ],
    "micInputs": [
      { "inputId": "<inputUuid>", "volume": 0.0, "balance": 0, "muted": false }
    ]
  }
}
```

Notes:

- `rename` and `order` can only be used when targeting a single zone.
- If you provide a `duckers.*` object, it replaces the corresponding preset-based ducker with custom parameters (the preset ducker ID is cleared).
- Assigning a `*PresetDuckerId` switches back to preset-based ducking.
- `mixers.lineInputs[]` and `mixers.micInputs[]` items must refer to inputs currently routed to the zone.
- Each mixer field (`volume`, `balance`, `muted`) is independently optional — omitted fields keep their current value.

6.2.5 `/zone-set-players`

Connects a player to a zone.

Notes:

- The first player from `playerIds` / `playerNames` is used.
- Behaves the same as `/zone-add-players`.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "playerIds": ["<playerUuid>"]
}
```

6.2.6 `/zone-add-players`

Connects a player to a zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "playerIds": ["<playerUuid>"]
}
```

6.2.7 `/zone-remove-players`

Disconnects a player from a zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "playerIds": ["<playerUuid>"]
}
```

6.2.8 `/zone-set-inputs`

Sets the zone inputs (replaces all existing inputs).

Params:

```
{
  "zoneId": "<zoneUuid>",
  "inputIds": ["<inputUuid>"]
}
```

6.2.9 `/zone-add-inputs`

Adds inputs to the zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "inputIds": ["<inputUuid>"]
}
```

6.2.10 `/zone-remove-inputs`

Removes inputs from the zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "inputIds": ["<inputUuid>"]
}
```

6.2.11 `/zone-set-speakers`

Sets zone speakers (replaces all existing speakers).

Params:

```
{
  "zoneId": "<zoneUuid>",
  "speakerIds": ["<speakerUuid>"]
}
```

6.2.12 `/zone-add-speakers`

Adds speakers to zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "speakerIds": ["<speakerUuid>"]
}
```

6.2.13 `/zone-remove-speakers`

Removes speakers from zone.

Params:

```
{
  "zoneId": "<zoneUuid>",
  "speakerIds": ["<speakerUuid>"]
}
```

6.2.14 `/zone-announcement-abort`

Aborts an announcement currently playing in a zone.

Params:

```
{ "zoneId": "<zoneUuid>" }
```

7 Speakers

A **Speaker** represents a physical output channel on the device. It is connected to exactly one **Zone** (receiving the zone's mixed audio) and exactly one device output channel (determining which hardware output it drives).

Each speaker carries its own volume, mute, delay, EQ, and limiter processing — applied after the zone mixer and before the hardware output.

The `type` field (`left`, `right`, or `center`) is metadata used for stereo pairing and routing hints.

Connections in the system:

```
Zone ▶ Speaker ▶ Device output channel
```

7.1 Speaker object

Every speaker in the system is identified by a UUID. The full speaker object returned by `/speaker-get-status-all` and pushed via WebSocket has the following structure:

```
{
  "name": "Speaker 1",
  "description": "",
  "volume": 0.0,
  "type": "left",
  "muted": false,
  "order": 0,
  "eq": {
    "bands": [
      {
        "idx": 0,
        "curve": "peak",
        "frequency": 500.0,
        "resonance": 1.0,
        "gain": 0.0,
        "bypassed": true
      }
    ]
  },
  "delay": {
    "time": 2
  },
  "delayBypassed": true,
  "limiter": {
    "threshold": 0.0,
    "release": 100.0,
    "mode": "rms"
  },
  "playingPinkNoise": false,
  "signalActive": false,
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": false,
  "presetLimiterId": "<presetLimiterUuid>",
  "presetLimiterBypassed": true,
  "connections": {
    "zone": "<zoneUuid>",
    "devices": [1]
  }
}
```


7.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the speaker.
`description`	string	Optional free-text description.
`volume`	float	Output level in dBFS (0.0 = unity).
`type`	string	Speaker role: `"left"`, `"right"`, or `"center"`.
`muted`	boolean	Whether the speaker output is muted.
`order`	integer	Position in the speaker list (0-based).

Field	Type	Description
`eq`	object | null	Custom EQ configuration, or `null` when a preset EQ is assigned (see `eq`).
`delay`	object | null	Delay configuration (see `delay`).
`delayBypassed`	boolean	Whether the delay stage is bypassed.
`limiter`	object | null	Custom limiter configuration, or `null` when a preset limiter is assigned (see `limiter`).
`playingPinkNoise`	boolean	Whether pink noise is currently playing on this speaker.
`signalActive`	boolean	`true` when the speaker output level exceeds the signal-active threshold.
`presetEqId`	string | null	UUID of the assigned EQ preset, or `null` when using custom EQ.
`presetEqBypassed`	boolean	Whether the EQ stage is bypassed.
`presetLimiterId`	string | null	UUID of the assigned limiter preset, or `null` when using a custom limiter.
`presetLimiterBypassed`	boolean	Whether the limiter stage is bypassed.
`connections`	object	Zone and device channel connections (see `connections`).

7.1.2 `eq`

`eq` is `null` when a preset EQ is assigned (`presetEqId` is set). Otherwise it contains a `bands` array of EQ band objects:

Field	Type	Description
`idx`	integer	Band index (0-based).
`curve`	string	Filter type (e.g. `"peak"`, `"lowShelf"`, `"highShelf"`, `"lowPass"`, `"highPass"`).
`frequency`	float	Centre/corner frequency in Hz.
`resonance`	float	Q / resonance factor.
`gain`	float	Band gain in dB.
`bypassed`	boolean	Whether this band is bypassed.

Setting `eq` directly clears `presetEqId`. Setting `presetEqId` switches back to preset-based EQ.

7.1.3 `delay`

Field	Type	Description
`time`	integer	Delay time in milliseconds.

7.1.4 `limiter`

`limiter` is `null` when a preset limiter is assigned (`presetLimiterId` is set). Otherwise it contains:

Field	Type	Description
`threshold`	float	Limiting threshold in dBFS.
`release`	float	Release time in milliseconds.
`mode`	string	Detection mode: `"rms"` or `"peak"`.

Setting `limiter` directly clears `presetLimiterId`. Setting `presetLimiterId` switches back to preset-based limiting.

7.1.5 `connections`

Field	Type	Description
`zone`	string | null	UUID of the connected zone, or `null`.
`devices`	integer[]	Device output channel numbers this speaker drives.

7.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/speaker-get-status-all`
- `/speaker-create`
- `/speaker-remove`
- `/speakers-modify`
- `/speaker-set-zone`
- `/speaker-remove-zone`
- `/speaker-set-channel`
- `/speaker-remove-channel`
- `/speaker-play-pink-noise`

7.2.1 `/speaker-get-status-all`

Method: `GET`

Returns all speakers.

Params:

```
None
```

Response:

```
{
  "speakers": {
    "<speakerUuid>": { ... },
    "<speakerUuid>": { ... }
  }
}
```

Each value is a full speaker object (see Speaker object above).

7.2.2 `/speaker-create`

Params:

```
{
  "name": "Speaker 1",
  "type": "left",

  "description": "",
  "volume": 0.0,
  "muted": false,

  "delay": { "time": 0 },
  "delayBypassed": true,

  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,

  "presetLimiterId": "<presetLimiterUuid>",
  "presetLimiterBypassed": true
}
```

Param	Required	Default	Description
`name`	yes	—	Must be unique across speakers
`type`	yes	—	`"left"`, `"right"`, or `"center"`
`description`	no	`""`	
`volume`	no	`0.0`	dBFS
`muted`	no	`false`	
`delay`	no	—	Delay config object
`delayBypassed`	no	`true`	
`presetEqId`	no	default preset	EQ preset UUID
`presetEqBypassed`	no	`true`	
`presetLimiterId`	no	default preset	Limiter preset UUID
`presetLimiterBypassed`	no	`true`	

Response:

```
{ "id": "<speakerUuid>" }
```

7.2.3 `/speaker-remove`

Removes a speaker. Fails if the speaker is referenced by any action/action group param.

If the speaker was connected to a zone, that zone is also updated.

Params:

```
{ "speakerId": "<speakerUuid>" }
```

`speakerId` or `speakerName` can be used.

7.2.4 `/speakers-modify`

Modifies one or multiple speakers.

Params:

```
{
  "speakerIds": ["<speakerUuid>"],
  "rename": "New speaker name",
  "order": 0,

  "description": "",
  "volume": 0.0,
  "muted": false,
  "type": "left",

  "delay": { "time": 0 },
  "delayBypassed": true,

  "eq": {
    "bands": [
      {
        "idx": 0,
        "curve": "peak",
        "frequency": 500.0,
        "resonance": 1.0,
        "gain": 0.0,
        "bypassed": true
      }
    ]
  },
  "presetEqId": "<presetEqUuid>",
  "presetEqBypassed": true,

  "limiter": {
    "threshold": 0.0,
    "release": 100.0,
    "mode": "rms"
  },
  "presetLimiterId": "<presetLimiterUuid>",
  "presetLimiterBypassed": true
}
```

All params except the selection are optional.

Notes:

- `rename` and `order` only work when a single speaker is selected.
- Setting `eq` directly clears `presetEqId`. Setting `presetEqId` overrides the custom EQ.
- Setting `limiter` directly clears `presetLimiterId`. Setting `presetLimiterId` overrides the custom limiter.

7.2.5 `/speaker-set-zone`

Connects a speaker to a zone (disconnecting it from any previous zone).

Params:

```
{
  "speakerId": "<speakerUuid>",
  "zoneId": "<zoneUuid>"
}
```

`speakerId` or `speakerName` can be used. `zoneId` or `zoneName` can be used.

7.2.6 `/speaker-remove-zone`

Disconnects a speaker from its zone.

Params:

```
{ "speakerId": "<speakerUuid>" }
```

`speakerId` or `speakerName` can be used.

7.2.7 `/speaker-set-channel`

Connects a speaker to a device output channel.

Notes:

- If another speaker uses the same `deviceChannel`, it will be disconnected first.
- `deviceChannel` must be between 1 and the device's total channel count.
- Fails if the output channels limit would be exceeded.

Params:

```
{
  "speakerId": "<speakerUuid>",
  "deviceChannel": 1
}
```

`speakerId` or `speakerName` can be used.

7.2.8 `/speaker-remove-channel`

Disconnects a speaker from its device output channel.

Params:

```
{ "speakerId": "<speakerUuid>" }
```

`speakerId` or `speakerName` can be used.

7.2.9 `/speaker-play-pink-noise`

Starts/stops pink noise playback on a speaker. Useful for level calibration.

Params:

```
{
  "speakerId": "<speakerUuid>",
  "play": true
}
```

`speakerId` or `speakerName` can be used.

8 Playlists

A **Playlist** is an ordered list of audio files and/or folder references that a **Player** (with `sourceType: "playlist"`) reads through. Multiple players can share a playlist, but each player tracks its own playback position independently.

Playlist items can be individual files or folder references. Folder items support autoscan — they automatically refresh their contents when the folder changes on disk.

Connections in the system:

```
Audio files / folders ▶ Playlist ▶ Player(s)
```

8.1 Playlist object

Every playlist in the system is identified by a UUID. The full playlist object returned by `/playlist-get-status-all` has the following structure:

```
{
  "name": "Playlist 1",
  "order": 0,
  "rootPath": "/Music",
  "contents": [
    {
      "file": {
        "position": 0,
        "relativePath": "song.flac",
        "absolutePath": "/Music/folder/song.flac",
        "parentPath": "/folder/",
        "pathValid": true
      }
    },
    {
      "folder": {
        "position": 1,
        "relativePath": "subfolder",
        "absolutePath": "/Music/subfolder",
        "contents": [ ... ],
        "rootPath": "/Music",
        "parentPath": "/",
        "pathValid": true,
        "autoscan": false
      }
    }
  ]
}
```

8.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the playlist.
`order`	integer	Position in the playlist list (0-based).
`rootPath`	string	Filesystem root used to resolve relative paths.
`contents`	array	Ordered list of file and folder items (see `contents`).

8.1.2 `contents`

Each item in `contents` is a wrapped object with either a `file` or `folder` key.

File item (`file`)

Field	Type	Description
`position`	integer	Flat position index across the entire playlist.
`relativePath`	string	File name.
`absolutePath`	string	Full path on disk.
`parentPath`	string	Parent directory relative to the root path.
`pathValid`	boolean	Whether the file currently exists on disk.

Folder item (`folder`)

Field	Type	Description
`position`	integer	Flat position index of the folder header.
`relativePath`	string	Folder name.
`absolutePath`	string	Full path on disk.
`contents`	array	Nested array of file/folder items.
`rootPath`	string	Music root path.
`parentPath`	string	Parent directory relative to the root path.
`pathValid`	boolean	Whether the folder currently exists on disk.
`autoscan`	boolean	Whether the folder automatically refreshes its contents on disk changes.

8.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/playlist-get-status-all`
- `/playlist-create`

- `/playlist-remove`
- `/playlist-modify`
- `/playlist-set-players`
- `/playlist-add-item`
- `/playlist-remove-item`
- `/playlist-move-item`
- `/playlist-scan-folder`
- `/playlist-set-folder-autoscan`

8.2.1 `/playlist-get-status-all`

Method: `GET`

Returns all playlists.

Params:

```
None
```

Response:

```
{
  "playlists": {
    "<playlistUuid>": { ... },
    "<playlistUuid>": { ... }
  }
}
```

Each value is a full playlist object (see Playlist object above).

8.2.2 `/playlist-create`

Params:

```
{ "name": "Playlist 1" }
```

Response:

```
{ "id": "<playlistUuid>" }
```

8.2.3 `/playlist-remove`

Params:

```
{ "playlistId": "<playlistUuid>" }
```

8.2.4 `/playlist-modify`

Params:

```
{
  "playlistId": "<playlistUuid>",
  "rename": "New playlist name",
  "order": 0
}
```

8.2.5 `/playlist-set-players`

Sets the playlist assignment:

- Players in the provided list will have this playlist set.
- Players previously using this playlist but not in the list will have it unset.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "playerIds": ["<playerUuid1>", "<playerUuid2>"]
}
```

8.2.6 `/playlist-add-item`

Adds a file to playlist.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "path": "/Music/some-file.mp3"
}
```

8.2.7 `/playlist-remove-item`

Removes playlist items by positions.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "position": [0, 3, 10]
}
```

8.2.8 `/playlist-move-item`

Moves a playlist item.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "source": 3,
  "destination": 0
}
```

8.2.9 `/playlist-scan-folder`

Scans a playlist folder item at given position (adds/refreshes its contents).

Params:

```
{
  "playlistId": "<playlistUuid>",
  "position": 0
}
```

8.2.10 `/playlist-set-folder-autoscan`

Toggles autoscan for a folder element in playlist.

Params:

```
{
  "playlistId": "<playlistUuid>",
  "position": 0,
  "autoscan": true
}
```

9 Announcements

An **Announcement** is a one-shot audio clip that interrupts normal playback in one or more **Zones**. When triggered, the zone lowers player and input levels (ducking) according to the announcement's `duckingVolume`, `attack`, and `release` settings, plays the clip through the dedicated announcement mixer bus, then restores normal levels.

Multiple announcements can queue up in a zone; playback order is determined by `priority` (higher value = higher priority).

Connections in the system:

```
Audio file ▶ Announcement ▶ Zone(s)  (via announcement mixer bus, one-shot)
```

9.1 Announcement object

Every announcement in the system is identified by a UUID. The full announcement object returned by `/announcement-get-status-all` has the following structure:

```
{
  "name": "Bell",
  "path": "Announcement/bell.wav",
  "pathValid": true,
  "volume": 0.0,
  "priority": 5,
  "attack": 500,
  "release": 500,
  "duckingVolume": -24.0,
  "order": 0,
  "duration": 3.5
}
```

9.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the announcement.
`path`	string	Relative path to the audio file.
`pathValid`	boolean	Whether the file currently exists on disk.
`volume`	float	Playback volume in dBFS.
`priority`	integer	Priority level — higher value means higher priority when multiple announcements queue in a zone.
`attack`	integer	Ducking fade-in time in milliseconds.
`release`	integer	Ducking fade-out time in milliseconds.

Field	Type	Description
`duckingVolume`	float	Level other sources are ducked to in dBFS during playback.
`order`	integer	Position in the announcement list (0-based).
`duration`	float	Duration of the audio file in seconds (`0` if file is invalid).

9.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/announcement-get-status-all`
- `/announcement-create`
- `/announcement-remove`
- `/announcements-modify`
- `/announcement-play-to-zones`
- `/announcements-abort`

9.2.1 `/announcement-get-status-all`

Method: `GET`

Returns all announcements.

Params:

```
None
```

Response:

```
{
  "announcements": {
    "<announcementUuid>": { ... },
    "<announcementUuid>": { ... }
  }
}
```

Each value is a full announcement object (see Announcement object above).

9.2.2 `/announcement-create`

Params:

```
{
  "name": "Bell",
  "path": "/announcements/bell.wav",

  "duckingVolume": -20.0,
  "volume": 0.0,

  "attack": 100,
  "release": 200,
  "priority": 5
}
```

Response:

```
{ "id": "<announcementUuid>" }
```

9.2.3 `/announcement-remove`

Params:

```
{ "announcementId": "<announcementUuid>" }
```

9.2.4 `/announcements-modify`

Params:

```
{
  "announcementId": "<announcementUuid>",
  "rename": "New name",
  "order": 0,

  "path": "/announcements/new.wav",
  "duckingVolume": -20.0,
  "volume": 0.0,
  "attack": 100,
  "release": 200,
  "priority": 5
}
```

9.2.5 `/announcement-play-to-zones`

Plays an announcement to one or multiple zones.

Params:

```
{
  "announcementId": "<announcementUuid>",
  "zoneIds": ["<zoneUuid1>", "<zoneUuid2>"]
}
```

9.2.6 `/announcements-abort`

Aborts currently-playing announcements by ids/names across all zones.

Params:

```
{
  "announcementIds": ["<announcementUuid>"]
}
```

10 Presets

Presets are used to save and reuse settings for audio processors: **EQ**, **Gate**, **Ducker**, **Limiter**, and **Compressor**.

Each preset type has its own set of endpoints for creating, listing, modifying, and removing presets. Presets are identified by a unique `name` (IDs are used internally).

10.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

Create:

- `/preset-eq-create`
- `/preset-gate-create`
- `/preset-ducker-create`
- `/preset-limiter-create`
- `/preset-compressor-create`

Get status:

- `/preset-eq-get-status-all`
- `/preset-gate-get-status-all`
- `/preset-ducker-get-status-all`
- `/preset-limiter-get-status-all`
- `/preset-compressor-get-status-all`

Modify:

- `/preset-eq-modify`
- `/preset-gate-modify`
- `/preset-ducker-modify`
- `/preset-limiter-modify`
- `/preset-compressor-modify`

Remove:

- `/preset-eq-remove`
- `/preset-gate-remove`
- `/preset-ducker-remove`
- `/preset-limiter-remove`
- `/preset-compressor-remove`

10.2 Usage

The API for each preset type is identical except for the endpoint and the preset parameters.

Below is an example for the **Ducker** preset. Replace `ducker` with `eq`, `gate`, `limiter`, or `compressor` for other types.

10.2.1 `/preset-ducker-create`

Creates a new ducker preset.

Params:

```
{
  "name": "preset2",
  "preset": {
    "threshold": -20.0,
    "attack": 500,
    "hold": 1000,
    "release": 500,
    "range": -10.0,
    "bypassed": false
  }
}
```

- `name` (string, required): Unique name for the preset.
- `preset` (object, required): Preset parameters for the processor.

Response:

```
{
  "id": "uuid-string"
}
```

10.2.2 `/preset-ducker-get-status-all`

Lists all ducker presets.

Response:

```
{
  "presetsDucker": [
    {
      "id": "uuid-string",
      "name": "preset2",
      "preset": {
        "threshold": -20.0,
        "attack": 500,
        "hold": 1000,
        "release": 500,
        "range": -10.0,
        "bypassed": false
      }
    }
    // ...more presets...
  ]
}
```

10.2.3 `/preset-ducker-modify`

Modifies an existing ducker preset. You can update the preset parameters and/or rename the preset.

Params:

```
{
  "presetDuckerName": "preset2",
  "rename": "newName",      // optional
  "preset": { /* ... */ }   // optional
}
```

- `presetDuckerName` or `presetDuckerId` (string, required): Name or id of the preset to modify.
- `rename` (string, optional): New name for the preset.
- `preset` (object, optional): New preset parameters.

10.2.4 `/preset-ducker-remove`

Removes a ducker preset.

Params:

```
{
  "presetDuckerName": "preset2"
}
```

- `presetDuckerName` or `presetDuckerId` (string, required): Name or id of the preset to remove.

  Note: Removing a preset that is currently in use or removing the default preset will return an error.

10.3 Other Preset Types

For **EQ**, **Gate**, **Limiter**, and **Compressor** presets, use the same API structure as above, replacing `ducker` with the desired type in the endpoint and parameter names.

Example for EQ:

- `/preset-eq-create`
- `/preset-eq-get-status-all`
- `/preset-eq-modify`
- `/preset-eq-remove`

Response fields:

- `presetsEq`, `presetsGate`, `presetsLimiter`, `presetsCompressor` for the respective get-status-all endpoints.

10.4 Notes

- Preset names must be unique per type.
- The default preset (with an empty or nil UUID) cannot be modified or removed.
- Removing a preset that is currently in use will result in an error.
- Modifying a preset updates all entities using that preset.

11 Actions

An **Action** is a named sequence of steps executed in order. Each step is either an API request or a time delay. Multiple actions can be triggered in parallel. Actions are identified by name or ID following the same conventions as other resources.

For example, the following action starts playback and then unmutes the speakers after a 500 ms delay:

```
Request /players-action  { action: "play" }
Delay 500 ms
Request /speakers-modify { mute: false }
```

Step types:

- `task` — an HTTP POST request to an API endpoint, with fields: `url`, `method`, `params`
- `delay` — a fixed wait, with field: `ms`
- `waitForAnnouncementEnd` — waits until the last triggered announcement finishes, then delays for `ms` milliseconds

Example action definition:

```
{
  "name": "playersNextWait",
  "actionGroupNames": ["group_name_1"],
  "sequence": [
    {
      "type": "task",
      "url": "/players-action",
      "method": "POST",
      "params": { "action": "play" }
    },
    { "type": "delay", "ms": 500 },
    {
      "type": "task",
      "url": "/speakers-modify",
      "method": "POST",
      "params": { "mute": false }
    }
  ],
  "description": "",
  "triggerCode": ""
}
```

11.1 Action object

Every action in the system is identified by a UUID. The full action object returned by `/action-get-status-all` has the following structure:

```
{
  "name": "play notif_action",
  "order": 0,
  "actionGroupIds": ["<actionGroupUuid>"],
  "sequence": [
    { "type": "delay", "ms": 2000 },
    { "type": "task", "url": "/announcement-play-to-zones", "method": "POST", "params": { "announcementId": "<uuid>", "
    { "type": "waitForAnnouncementEnd", "ms": 1000 },
    { "type": "task", "url": "/players-action", "method": "POST", "params": { "action": "play" } }
  ],
  "running": false,
  "description": "<p>optional HTML description</p>",
  "triggerCode": "PK4"
}
```

11.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the action.
`order`	integer	Position in the action list (0-based).
`actionGroupIds`	string[]	UUIDs of action groups this action belongs to.
`sequence`	array	Ordered list of steps to execute (see `sequence`).
`running`	boolean	Whether the action is currently executing.
`description`	string	Optional HTML description.
`triggerCode`	string	Optional trigger code (ASCII only, must be unique if non-empty).

11.1.2 `sequence`

Each item in the `sequence` array is one of the following step types:

Type	Fields	Description
`task`	`url`, `method`, `params`	An HTTP request to an API endpoint.
`delay`	`ms`	A fixed wait in milliseconds.
`waitForAnnouncementEnd`	`ms`	Waits for the last triggered announcement to finish, then delays for `ms` ms.

11.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/action-get-status-all`
- `/action-create`

- `/action-modify`
- `/action-play`
- `/action-stop`
- `/action-remove`

11.2.1 `/action-get-status-all`

Method: `GET`

Returns all actions.

Params:

```
None
```

Response:

```
{
  "actions": {
    "<actionUuid>": { ... },
    "<actionUuid>": { ... }
  }
}
```

Each value is a full action object (see Action object above).

11.2.2 `/action-create`

Creates a new action.

Params:

```
{
  "name": "playersNextWait",
  "actionGroupNames": ["group_name_1"],
  "description": "optional",
  "triggerCode": "",
  "sequence": [
    { "type": "task", "url": "/players-action", "method": "POST", "params": { "action": "play" } },
    { "type": "delay", "ms": 500 }
  ]
}
```

Notes:

- `triggerCode` is optional, ASCII only, and must be unique if provided.

Response:

```
{ "id": "<actionUuid>" }
```

11.2.3 `/action-modify`

Modifies an existing action. All fields except the selector are optional.

Params:

```
{
  "actionId": "<actionUuid>",
  "rename": "newName",
  "order": 0,
  "actionGroupNames": ["group_name_1"],
  "description": "optional",
  "triggerCode": "",
  "sequence": [
    { "type": "task", "url": "/players-action", "method": "POST", "params": { "action": "play" } }
  ]
}
```

Notes:

- `triggerCode` is optional, ASCII only, and must be unique if provided.

11.2.4 `/action-play`

Executes a list of actions simultaneously. Each step is performed regardless of whether previous steps succeed or fail.

Params:

```
{ "actionIds": ["<actionUuid>"] }
```

11.2.5 `/action-stop`

Stops a running action. If the action played any announcements, they will be stopped too.

Params:

```
{ "actionId": "<actionUuid>" }
```

11.2.6 `/action-remove`

Removes an action.

Params:

```
{ "actionId": "<actionUuid>" }
```

12 Action Groups

An **Action Group** is a named collection of actions used to organise and categorise them. Each action can belong to one or more groups, and groups are used for display and filtering purposes.

12.1 Action Group object

Every action group in the system is identified by a UUID. The full action group object returned by `/action-group-get-status-all` has the following structure:

```
{
  "name": "group_name_1",
  "order": 0,
  "actionIds": ["<actionUuid>"]
}
```

12.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the action group.
`order`	integer	Position in the action group list (0-based).
`actionIds`	string[]	UUIDs of actions belonging to this group.

12.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/action-group-get-status-all`
- `/action-group-create`
- `/action-group-modify`
- `/action-group-remove`

12.2.1 `/action-group-get-status-all`

Method: `GET`

Returns all action groups.

Params:

```
None
```

Response:

```
{
  "actionGroups": {
    "<actionGroupUuid>": { ... },
    "<actionGroupUuid>": { ... }
  }
}
```

Each value is a full action group object (see Action Group object above).

12.2.2 `/action-group-create`

Creates a new action group.

Params:

```
{
  "name": "group_name_1",
  "actionIds": ["<actionUuid>"]
}
```

Response:

```
{ "id": "<actionGroupUuid>" }
```

12.2.3 `/action-group-modify`

Modifies an existing action group. All fields except the selector are optional.

Params:

```
{
  "actionGroupId": "<actionGroupUuid>",
  "name": "newName",
  "order": 0,
  "actionIds": ["<actionUuid>"]
}
```

12.2.4 `/action-group-remove`

Removes an action group.

Params:

```
{ "actionGroupId": "<actionGroupUuid>" }
```

13 External Actions

External actions are used for communication to external entities, devices and others. By default they should be triggered by scheduler, but there are 3 URLs prepared for UI to allow user to test communication in advance.

13.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/external-send-tcp-message`
- `/external-send-udp-message`
- `/external-send-tcp-hex`
- `/external-send-udp-hex`
- `/external-send-http-request`
- `/external-modbus-read-coils`
- `/external-modbus-read-discrete-inputs`
- `/external-modbus-read-holding-registers`
- `/external-modbus-read-input-registers`
- `/external-modbus-write-single-coil`
- `/external-modbus-write-single-register`
- `/external-modbus-write-multiple-coils`
- `/external-modbus-write-multiple-registers`

13.1.1 `/external-send-tcp-message`

Sends TCP message to external entity. Collects its response (if present) and sends it back to UI.

Params:

```
{
    "targetUrl": "192.168.0.33:7200",
    "body": "Message that will be sent to the target",
    "timeoutMs": 2000
}
```

- `targetUrl` - ip:port or hostname:port or any other valid url.
- `body` - message that will be sent to targetUrl via TCP.
- `timeoutMs` (optional), u64 - timeout for TCP communication to/from the target. If not present, system default will be used (externalActionsTimeoutMs).

Responses:

- No response from external entity: `204`
- Response received:

```
{
    "responseBody": "Some response"
}
```

13.1.2 `/external-send-udp-message`

Sends UDP message to external entity. Collects its response (if present) and sends it back to UI.

Params:

```
{
    "targetUrl": "192.168.0.33:7200",
    "body": "Message that will be sent to the target",
    "timeoutMs": 2000
}
```

- `targetUrl` - ip:port or hostname:port or any other valid url.
- `body` - message that will be sent to targetUrl via UDP.
- `timeoutMs` (optional), u64 - timeout for UDP communication to/from the target. If not present, system default will be used (externalActionsTimeoutMs).

Responses:

- No response from external entity: `204`
- Response received:

```
{
    "responseBody": "Some response"
}
```

13.1.3 `/external-send-tcp-hex`

Sends raw bytes over TCP encoded as a hex string (whitespace is allowed and ignored). If a response is received, it is returned as an uppercase hex string.

Params:

```
{
    "targetUrl": "192.168.0.33:7200",
    "body": "AA 01 02 FF",
    "timeoutMs": 2000
}
```

- `targetUrl` - ip:port or hostname:port.
- `body` - hex string (two hex characters per byte; whitespace is ignored; length must be even after removing whitespace).
- `timeoutMs` (optional), u64 - timeout for TCP communication to/from the target. If not present, system default will be used (externalActionsTimeoutMs).

Responses:

- No response from external entity: `204`
- Response received:

```
{
    "responseBody": "DEADBEEF"
}
```

13.1.4 `/external-send-udp-hex`

Sends raw bytes over UDP encoded as a hex string (whitespace is allowed and ignored). If a response is received, it is returned as an uppercase hex string.

Params:

```
{
    "targetUrl": "192.168.0.33:7200",
    "body": "AA 01 02 FF",
    "timeoutMs": 2000
}
```

- `targetUrl` - ip:port or hostname:port.
- `body` - hex string (two hex characters per byte; whitespace is ignored; length must be even after removing whitespace).
- `timeoutMs` (optional), u64 - timeout for UDP communication to/from the target. If not present, system default will be used (externalActionsTimeoutMs).

Responses:

- No response from external entity: `204`
- Response received:

```
{
    "responseBody": "DEADBEEF"
}
```

13.1.5 `/external-send-http-request`

Sends HTTP request to external entity. Collects its response (if present) and sends it back to UI.

Params:

```
{
    "targetUrl": "https://somewhere.com/something",
    "method": "POST",
    "body": "Message that will be sent to the target",
    "timeoutMs": 2000
}
```

- `targetUrl` - any valid URL.
- `method` - **POST** | **PUT** | **PATCH** | **GET** | **DELETE** - request method that will be used.
- `body` (optional) - request body that will be sent to targetUrl. Only applicable for **POST**, **PUT**, **PATCH** methods.
- `timeoutMs` (optional), u64 - timeout for communication to/from the target. If not present, system default will be used (externalActionsTimeoutMs).

Responses:

- Response without body:

```
{
    "responseStatus": 200
}
```

- Response with body:

```
{
    "responseStatus": 200,
    "responseBody": "Some response"
}
```

13.1.6 `/external-modbus-read-coils`

Reads coils from a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "quantity": 10,
    "timeoutMs": 2000
}
```

- `targetUrl` - IP or hostname of the Modbus device.
- `port` (optional) - TCP port of the Modbus device. Default is 502.
- `uid` - Modbus unit ID.
- `address` - Starting address of the coils to read.
- `quantity` - Number of coils to read.
- `timeoutMs` (optional) - Timeout for the operation. Default is `externalActionsTimeoutMs`.

Responses:

- Success:

```
{
    "responseBody": [true, false, true]
}
```

13.1.7 `/external-modbus-read-discrete-inputs`

Reads discrete inputs from a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "quantity": 10,
    "timeoutMs": 2000
}
```

- Same parameters as `/external-modbus-read-coils`.

Responses:

- Success:

```
{
    "responseBody": [true, false, true]
}
```

13.1.8 `/external-modbus-read-holding-registers`

Reads holding registers from a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "quantity": 10,
    "timeoutMs": 2000
}
```

- Same parameters as `/external-modbus-read-coils`.

Responses:

- Success:

```
{
    "responseBody": [123, 456, 789]
}
```

13.1.9 `/external-modbus-read-input-registers`

Reads input registers from a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "quantity": 10,
    "timeoutMs": 2000
}
```

- Same parameters as `/external-modbus-read-coils`.

Responses:

- Success:

```
{
    "responseBody": [123, 456, 789]
}
```

13.1.10 `/external-modbus-write-single-coil`

Writes a single coil to a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "value": true,
    "timeoutMs": 2000
}
```

- `value` - Boolean value to write to the coil.

Responses:

Responds with `204` on success.

13.1.11 `/external-modbus-write-single-register`

Writes a single register to a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "value": 123,
    "timeoutMs": 2000
}
```

- `value` - Integer value to write to the register.

Responses:

Responds with `204` on success.

13.1.12 `/external-modbus-write-multiple-coils`

Writes multiple coils to a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "values": [true, false, true],
    "timeoutMs": 2000
}
```

- `values` - Array of boolean values to write to the coils.

Responses:

Responds with `204` on success.

13.1.13 `/external-modbus-write-multiple-registers`

Writes multiple registers to a Modbus device.

Params:

```
{
    "targetUrl": "192.168.0.33",
    "port": 502,
    "uid": 1,
    "address": 0,
    "values": [123, 456, 789],
    "timeoutMs": 2000
}
```

- `values` - Array of integer values to write to the registers.

Responses:

Responds with `204` on success.

14 Scheduler

The Scheduler triggers Actions automatically at configured times. It is built from four entity types that work together:

- **Intervals** — define recurring time windows (e.g. "every weekday 08:00–18:00"). Each interval is either *positive* (active window) or *negative* (blackout/exclusion window).
- **Interval Groups** — combine multiple intervals into a single logical window: the union of all positive intervals minus all negative intervals.
- **Event Groups** — visually group events under a shared label.
- **Events** — define when to fire actions. An event has a *type* that controls the firing pattern (e.g. daily at specific times, at the start of an interval). Events can optionally reference an interval or interval group so they only fire during active windows.

Interval resolution for events: when an event fires, the effective interval comes from the event's own `intervalId` or `intervalGroupId`.

Validity window: every interval and event has `firstOccurrence` and `lastOccurrence` fields that bound when it is active. Use `"-infinity"` and `"+infinity"` for open-ended ranges.

14.1 Interval object

An interval defines one recurring time window, used as building blocks for Interval Groups.

```
{
  "name": "Weekdays",
  "enabled": true,
  "positive": true,
  "intervalGroupId": "<intervalGroupUuid>",
  "firstOccurrence": "2026-01-01T00:00:00",
  "lastOccurrence": "+infinity",
  "userData": {},

  "intervalType": "weekly",
  "startTime": "08:00:00",
  "endTime": "18:00:00",
  "days": [false, true, true, true, true, true, false]
}
```

14.1.1 Field reference

Field	Type	Description
`name`	string	Display name of the interval.

Field	Type	Description
`enabled`	boolean	Whether this interval is active. Default: `true`. Disabled intervals stay stored and grouped, but the scheduler ignores them until they are re-enabled.
`positive`	boolean	`true` = opening/active window; `false` = blackout/exclusion window. Default: `true`.
`intervalGroupId`	string|null	UUID of the interval group this interval belongs to, or `null`. Read by the group automatically.
`firstOccurrence`	string	Start of the interval's validity window. `"-infinity"` for no lower bound. Format: `"YYYY-MM-DDTHH:MM:SS"` or `"-infinity"`.
`lastOccurrence`	string	End of the interval's validity window. `"+infinity"` for no upper bound.
`userData`	any	Arbitrary client data stored with the interval.
`intervalType`	string	Type of interval. See table below.

14.1.2 `intervalType` values

`intervalType`	Extra fields	Description
`simple`	*(none)*	Always active within the validity window.
`periodic`	`length` (integer, seconds), `period` (integer, seconds)	Repeating active windows of `length` seconds, starting at `firstOccurrence` and repeating every `period` seconds. If `length` is greater than `period`, the windows overlap.
`daily`	`startTime` (string `"HH:MM:SS"`), `endTime` (string `"HH:MM:SS"`), `anomalies`	Runs every day from `startTime` to `endTime`. If `endTime` is earlier than `startTime`, the window ends on the next day.
`weekly`	`startTime`, `endTime`, `days` (array of 7 booleans), `anomalies`	Runs on selected weekdays. Index 0 = Sunday, 6 = Saturday. Overnight windows belong to the day on which they start.
`dayOfMonth`	`startTime`, `endTime`, `day` (integer 1–31), `anomalies`	Runs on a specific day of each month.
`weekdayOfMonth`	`startTime`, `endTime`, `day` (integer 0–6, Sun=0), `week` (integer 1–5), `anomalies`	Runs on the *N*-th weekday of each month (e.g. 2nd Tuesday).

`intervalType`	Extra fields	Description
`selectedDates`	`startTime`, `length`, `dates` (array of `"YYYY-MM-DD"` strings), `anomalies`	Fires only on the listed dates.
`prayer`	optional `latitude`, optional `longitude`, `method`, optional `calendarFiles`, `length`, `prayers` (array of 5 booleans), `anomalies`	Uses Aladhan prayer times. For online use, provide `latitude` and `longitude`. For offline use, provide one or more uploaded `calendarFiles` by filename. When `calendarFiles` is set, the interval uses only those uploaded calendars and does not contact Aladhan.

14.1.3 Prayer calendar files

Prayer intervals support two practical sources:

- Online lookup: the scheduler can download and locally cache annual Aladhan calendars for the interval's coordinates.
- Uploaded calendars: you can upload one or more annual Aladhan JSON files in the Adhan interval form and select them by filename for one interval.

Uploaded calendar files must contain the standard Aladhan `data` array with per-day `timings` and Gregorian dates, the same shape returned by the annual calendar API.

The Aladhan `from` / `to` calendar API is limited to at most 11 months in one request. For uploaded offline files, use the annual calendar API instead.

See the official Aladhan annual prayer times calendar documentation:

- https://aladhan.com/prayer-times-api#tag/annual-prayer-times-calendar/GET/calendar/{year}

Examples:

```
curl 'https://api.aladhan.com/v1/calendar/2026?latitude=51.5074&longitude=-0.1278&method=3&iso8601=false' \
  -o london-2026.json

curl 'https://api.aladhan.com/v1/calendar/2026?latitude=25.2048&longitude=55.2708&method=4&iso8601=false' \
  -o dubai-2026.json

curl 'https://api.aladhan.com/v1/calendar/2026?latitude=52.2297&longitude=21.0122&method=13&iso8601=false' \
  -o warsaw-2026.json
```

14.1.4 `anomalies`

Anomalies override the regular interval pattern for specific calendar dates. Each entry in the array replaces the normal window for that date.

Exclude a day:

```
[
  { "date": "2026-12-25", "excluded": true }
]
```

Override a day with different hours:

```
[
  { "date": "2026-12-31", "startTime": "10:00:00", "endTime": "02:00:00" }
]
```

Field	Type	Description
`date`	string	The date to override, `"YYYY-MM-DD"`.
`excluded`	boolean	When `true`, the interval is skipped for that date.
`startTime`	string	Replacement start time for that date, `"HH:MM:SS"`.
`endTime`	string	Replacement end time for that date, `"HH:MM:SS"`. If earlier than `startTime`, the replacement ends on the next day.

Each date can have one exception. Use either `excluded: true` or a `startTime` / `endTime` pair.

14.2 Interval Group object

An interval group aggregates multiple intervals into one logical time window. The effective window is: union of all positive-member intervals minus all negative-member intervals.

```
{
  "name": "Opening Hours",
  "userData": {},
  "intervalIds": ["<intervalUuid>", "<intervalUuid>"]
}
```

14.2.1 Field reference

Field	Type	Description
`name`	string	Display name of the interval group.
`userData`	any	Arbitrary client data stored with the group.
`intervalIds`	string[]	UUIDs of intervals that belong to this group. **Read-only** — managed automatically when intervals set their `intervalGroupId`.

14.3 Event Group object

An event group is a visual grouping label for events.

```
{
  "name": "Morning Zone Events",
  "userData": {}
}
```

14.3.1 Field reference

Field	Type	Description
`name`	string	Display name of the event group.
`userData`	any	Arbitrary client data stored with the group.

14.4 Event object

An event defines when to fire one or more actions. The firing pattern is controlled by `eventType`.

```
{
  "name": "Morning announcement",
  "enabled": true,
  "userData": {},
  "actionIds": ["<actionUuid>"],
  "eventGroupId": "<eventGroupUuid>",
  "intervalId": null,
  "intervalGroupId": null,
  "firstOccurrence": "2026-01-01T00:00:00",
  "lastOccurrence": "+infinity",

  "eventType": "weeklyAtTimes",
  "times": ["09:00:00", "12:00:00"],
  "days": [false, true, true, true, true, true, false]
}
```

14.4.1 Field reference

Field	Type	Description
`name`	string	Display name of the event.
`enabled`	boolean	Whether this event is active. Default: `true`.
`userData`	any	Arbitrary client data stored with the event.
`actionIds`	string[]	UUIDs of actions to fire when the event triggers.

Field	Type	Description
`eventGroupId`	string|null	UUID of the event group this event belongs to, or `null`.
`intervalId`	string|null	UUID of the interval referenced by this event, or `null`.
`intervalGroupId`	string|null	UUID of the interval group referenced by this event, or `null`.
`firstOccurrence`	string	Start of the event's validity window.
`lastOccurrence`	string	End of the event's validity window.
`dateTimes`	string[]	Explicit fire times for `eventType: "multiple"`, each formatted as `"YYYY-MM-DDTHH:MM:SS"`.
`eventType`	string	Firing pattern. See table below.

14.4.2 `eventType` values

`eventType`	Extra fields	Description
`single`	*(none)*	Fires exactly once. The fire time is `firstOccurrence`.
`singleDuringSchedule`	`times` (array containing one `"HH:MM:SS"` value), `intervalId` or `intervalGroupId`	Fires at one time of day, but only when the selected interval or interval group is active. Negative intervals, breaks, and anomalies still apply because interval resolution is checked at the exact candidate time.
`multiple`	`dateTimes` (array of `"YYYY-MM-DDTHH:MM:SS"`)	Fires once at each listed datetime. A single-item list behaves the same as `single`.
`multipleDuringSchedule`	`times` (array of `"HH:MM:SS"`), `intervalId` or `intervalGroupId`	Fires at each listed time of day, but only when the selected interval or interval group is active. Negative intervals, breaks, and anomalies still apply because interval resolution is checked at the exact candidate time.
`intervalStart`	`startOffset` (integer, seconds)	Fires at the start of each interval window, offset by `startOffset` seconds.
`intervalEnd`	`endOffset` (integer, seconds)	Fires at the end of each interval window, offset by `endOffset` seconds (negative = before the end).

`eventType`	Extra fields	Description
`intervalPeriodic`	`startOffset`, `endOffset`, optional `startTime`, optional `endTime`, `period` (integer, seconds)	Fires repeatedly during each interval window. Start and end can be defined either as relative offsets or as absolute clock times. `startTime` means "use this daily clock-time anchor and fire only when that grid lands inside the active interval". `endTime` means "stop at this daily clock-time even if the interval continues later".
`intervalPeriodicNTimes`	`startOffset`, optional `startTime`, `period` (integer, seconds), `numberOfOccurrences` (integer), `intervalId` or `intervalGroupId`	For each interval window, starts either from a relative offset or an absolute daily clock-time anchor, then fires every `period` seconds, but no more than `numberOfOccurrences` times in that window. When `startTime` is used, the count starts from the first candidate that actually lands inside the active window.
`periodicIndependent`	`period` (integer, seconds), `firstOccurrence`, `lastOccurrence`	Fires every `period` seconds within the validity window, regardless of intervals.
`periodicNTimes`	`firstOccurrence`, `period` (integer, seconds), `numberOfOccurrences` (integer)	Fires at `firstOccurrence`, then every `period` seconds, for a total of `numberOfOccurrences` times. `lastOccurrence` is not used.
`dailyAtTimes`	`times` (array of `"HH:MM:SS"` strings)	Fires every day at each listed time.
`dailyPeriodic`	`startTime` (`"HH:MM:SS"`), `endTime` (`"HH:MM:SS"`), `period` (integer, seconds)	Fires repeatedly every day within the `startTime`-`endTime` window, every `period` seconds.
`weeklyAtTimes`	`times` (array of `"HH:MM:SS"` strings), `days` (array of 7 booleans)	Fires on selected weekdays at each listed time. Index 0 = Sunday, 6 = Saturday.
`weeklyPeriodic`	`startTime`, `endTime`, `period`, `days`	Fires repeatedly within a daily window on selected weekdays, every `period` seconds.

14.4.3 Interval-periodic start/end modes

For `intervalPeriodic` and `intervalPeriodicNTimes`, you can configure the start and end with these modes:

- `No` — no adjustment. The event starts at the raw interval begin, and for `intervalPeriodic` ends at the raw interval end.

- `Adjust` — keep the existing relative-offset behavior. Negative values allow firing before the interval begin or ending before the interval end; positive end offsets can extend past the raw interval end.
- `Time` — use an absolute daily clock-time such as `14:00:00`.

`Time` mode semantics:

- The chosen time stays unchanged even if the interval definition changes.
- Candidates are generated on that absolute clock-time grid, then filtered by the active interval or interval group.
- For `intervalPeriodic`, `endTime` is a hard daily cutoff inside the active interval window.
- For `intervalPeriodicNTimes`, only the start side supports `Time` mode.

14.5 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

Occurrence queries - `/scheduler-get-occurrences`

Intervals - `/scheduler-interval-get-all` - `/scheduler-interval-create` - `/scheduler-interval-modify` - `/scheduler-interval-remove` - `/scheduler-interval-anomaly-upsert` - `/scheduler-interval-anomaly-remove` - `/scheduler-prayer-calendar-get-all` - `/scheduler-prayer-calendar-upload` - `/scheduler-prayer-calendar-remove` - `/scheduler-prayer-calendar-clear-cache`

Interval Groups - `/scheduler-interval-group-get-all` - `/scheduler-interval-group-create` - `/scheduler-interval-gro` - `/scheduler-interval-group-remove`

Event Groups - `/scheduler-event-group-get-all` - `/scheduler-event-group-create` - `/scheduler-event-group-modify` - `/scheduler-event-group-remove`

Events - `/scheduler-event-get-all` - `/scheduler-event-create` - `/scheduler-event-modify` - `/scheduler-event-remove`

14.5.1 `/scheduler-get-occurrences`

Returns a flat, time-sorted list of all event fire occurrences within a requested window. Designed for the four common display queries: next N upcoming events, day view, week view, and month view.

Params:

```
{
  "from": "2026-04-22T00:00:00",
  "to":   "2026-04-22T23:59:59",
  "limit": 100
}
```

Notes:

- All params are optional.
- `from` defaults to the current local time.
- `to` defaults to `from + 1 year` (open-ended).
- `limit` caps the total number of occurrences returned globally. Defaults to `10` for open-ended queries and `50000` for range queries (when `to` is specified).

- Only enabled events with at least one action are included.
- Results are sorted ascending by `time`.

Common usage patterns:

```
// Next 10 upcoming events (across all events, from now)
{ "limit": 10 }

// All events occurring on a specific day
{ "from": "2026-04-22T00:00:00", "to": "2026-04-22T23:59:59" }

// All events in a week
{ "from": "2026-04-21T00:00:00", "to": "2026-04-27T23:59:59" }

// All events in a month
{ "from": "2026-04-01T00:00:00", "to": "2026-04-30T23:59:59" }
```

Response:

```
{
  "occurrences": [
    {
      "time": "2026-04-22T09:00:00",
      "eventId": "<eventUuid>",
      "eventName": "Morning announcement",
      "actionIds": ["<actionUuid>"]
    },
    {
      "time": "2026-04-22T12:00:00",
      "eventId": "<eventUuid>",
      "eventName": "Noon chime",
      "actionIds": ["<actionUuid>"]
    }
  ]
}
```

14.5.2 `/scheduler-interval-get-all`

Method: `GET`

Returns all intervals.

Params:

```
None
```

Response:

```
{
  "schedulerIntervals": {
    "<intervalUuid>": { ... },
    "<intervalUuid>": { ... }
  }
}
```

Each value is a full interval object (see Interval object above).

14.5.3 `/scheduler-interval-create`

Creates a new interval.

Params:

```
{
  "name": "Weekdays",
  "intervalType": "weekly",
  "positive": true,
  "intervalGroupId": "<intervalGroupUuid>",
  "firstOccurrence": "2026-01-01T00:00:00",
  "lastOccurrence": "+infinity",
  "userData": {},
  "startTime": "08:00:00",
  "endTime": "18:00:00",
  "days": [false, true, true, true, true, true, false]
}
```

Notes:

- `name` and `intervalType` are required. All other fields are optional.
- `enabled` defaults to `true`. When `false`, the interval remains in storage and in its interval group membership, but it contributes no active window until re-enabled.
- `positive` defaults to `true`.
- `firstOccurrence` defaults to `-infinity`, `lastOccurrence` defaults to `+infinity`.
- Type-specific fields (`startTime`, `endTime`, `length`, `days`, `period`, `day`, `week`, `dates`) depend on `intervalType`. See Interval object for details.
- Add or remove schedule exceptions with the dedicated exception endpoints below.
- Prayer intervals may also include `calendarFiles`, an array of uploaded calendar filenames selected in the Adhan interval form.
- The interval group can be referenced by `intervalGroupId` (UUID) or `intervalGroupName` (display name).

Response:

```
{ "id": "<intervalUuid>" }
```

14.5.4 `/scheduler-interval-modify`

Modifies an existing interval. All fields except the selector are optional.

Params:

```
{
  "intervalId": "<intervalUuid>",
  "name": "Weekdays Updated",
  "positive": false,
  "intervalGroupId": null,
  "firstOccurrence": "2026-06-01T00:00:00",
  "lastOccurrence": "+infinity",
  "userData": {},
  "intervalType": "daily",
  "startTime": "09:00:00",
  "endTime": "17:00:00"
}
```

Notes:

- Selector: `intervalId` (UUID) or `intervalName` (display name).
- `enabled` - Disabled intervals are ignored by interval groups and any events that reference them.
- Send `null` for `intervalGroupId` to remove the interval from its current group.
- If `intervalType` is provided, all type-specific fields for the new type must also be provided.
- Schedule exceptions are managed with the exception endpoints below, not by modifying the whole interval.

14.5.5 `/scheduler-prayer-calendar-get-all`

Method: `GET`

Lists uploaded prayer calendar JSON files available in the Adhan calendar manager and returns the current size of the stored server cache.

Params:

```
None
```

Response:

```
{
  "files": [
    {
      "name": "london-2026.json",
      "latitude": 51.5074,
      "longitude": -0.1278
    }
  ],
  "cacheSizeBytes": 182044
}
```

14.5.6 `/scheduler-prayer-calendar-upload`

Method: `POST`

Multipart upload of one Aladhan-compatible prayer calendar JSON file.

Form fields:

```
file=<json file>
```

Notes:

- Use this endpoint through the Adhan calendar manager in the interval form.
- The file body must match the Aladhan annual calendar response shape.
- The response returns the uploaded filename, which can be saved in `calendarFiles` on a `prayer` interval.

Response:

```
{
  "name": "london-2026.json"
}
```

14.5.7 `/scheduler-prayer-calendar-remove`

Method: `DELETE`

Removes one or more uploaded prayer calendar files from the Adhan calendar manager.

Params:

```
{
  "name": ["london-2026.json"]
}
```

Notes:

- This endpoint removes uploaded offline files only.

14.5.8 `/scheduler-prayer-calendar-clear-cache`

Method: `POST`

Clears the server-stored prayer calendar cache created from Aladhan downloads.

Params:

```
None
```

Notes:

- Use this from the Adhan calendar manager when you want to free disk space or force the server to rebuild cached downloads.

14.5.9 `/scheduler-interval-remove`

Removes an interval.

Params:

```
{ "intervalId": "<intervalUuid>" }
```

Notes:

- Selector: `intervalId` (UUID) or `intervalName` (display name).
- Fails if the interval is currently referenced by any event or event group.
- The interval is automatically removed from its interval group before deletion.

14.5.10 `/scheduler-interval-anomaly-upsert`

Adds or updates one schedule exception for an interval. If an exception already exists for the same date, it is replaced.

Exclude a date:

```
{
  "intervalId": "<intervalUuid>",
  "date": "2026-12-25",
  "excluded": true
}
```

Override hours for a date:

```
{
  "intervalId": "<intervalUuid>",
  "date": "2026-12-31",
  "startTime": "10:00:00",
  "endTime": "02:00:00"
}
```

Notes:

- Selector: `intervalId` (UUID) or `intervalName` (display name).
- Use `excluded: true` to skip the interval for the selected date.
- Use `startTime` and `endTime` to replace the interval hours for the selected date.
- When `endTime` is earlier than `startTime`, the replacement ends on the next day.

14.5.11 `/scheduler-interval-anomaly-remove`

Removes one schedule exception from an interval.

Params:

```
{
  "intervalId": "<intervalUuid>",
  "date": "2026-12-25"
}
```

Notes:

- Selector: `intervalId` (UUID) or `intervalName` (display name).
- Removing an exception restores the interval's normal schedule for that date.

14.5.12 `/scheduler-interval-group-get-all`

Method: `GET`

Returns all interval groups.

Params:

```
None
```

Response:

```
{
  "schedulerIntervalGroups": {
    "<intervalGroupUuid>": { ... },
    "<intervalGroupUuid>": { ... }
  }
}
```

Each value is a full interval group object (see Interval Group object above).

14.5.13 `/scheduler-interval-group-create`

Creates a new interval group.

Params:

```
{
  "name": "Opening Hours",
  "userData": {}
}
```

Notes:

- `name` is required. All other fields are optional.
- Intervals join a group by setting their `intervalGroupId`, not by passing `intervalIds` here.

Response:

```
{ "id": "<intervalGroupUuid>" }
```

14.5.14 `/scheduler-interval-group-modify`

Modifies an existing interval group. All fields except the selector are optional.

Params:

```
{
  "intervalGroupId": "<intervalGroupUuid>",
  "name": "Business Hours",
  "userData": {}
}
```

Notes:

- Selector: `intervalGroupId` (UUID) or `intervalGroupName` (display name).
- `intervalIds` cannot be modified directly; manage membership through the intervals themselves.

14.5.15 `/scheduler-interval-group-remove`

Removes an interval group.

Params:

```
{ "intervalGroupId": "<intervalGroupUuid>" }
```

Notes:

- Selector: `intervalGroupId` (UUID) or `intervalGroupName` (display name).
- Fails if the group is currently referenced by any event or event group.
- All member intervals have their `intervalGroupId` cleared automatically.

14.5.16 `/scheduler-event-group-get-all`

Method: `GET`

Returns all event groups.

Params:

```
None
```

Response:

```
{
  "schedulerEventGroups": {
    "<eventGroupUuid>": { ... },
    "<eventGroupUuid>": { ... }
  }
}
```

Each value is a full event group object (see Event Group object above).

14.5.17 `/scheduler-event-group-create`

Creates a new event group.

Params:

```
{
  "name": "Morning Zone Events",
  "userData": {}
}
```

Notes:

- `name` is required. All other fields are optional.

Response:

```
{ "id": "<eventGroupUuid>" }
```

14.5.18 `/scheduler-event-group-modify`

Modifies an existing event group. All fields except the selector are optional.

Params:

```
{
  "eventGroupId": "<eventGroupUuid>",
  "name": "Updated Group",
  "userData": {}
}
```

Notes:

- Selector: `eventGroupId` (UUID) or `eventGroupName` (display name).

14.5.19 `/scheduler-event-group-remove`

Removes an event group.

Params:

```
{ "eventGroupId": "<eventGroupUuid>" }
```

Notes:

- Selector: `eventGroupId` (UUID) or `eventGroupName` (display name).
- Events that belong to this group are not deleted; their `eventGroupId` is cleared.

14.5.20 `/scheduler-event-get-all`

Method: `GET`

Returns all events.

Params:

```
None
```

Response:

```
{
  "schedulerEvents": {
    "<eventUuid>": { ... },
    "<eventUuid>": { ... }
  }
}
```

Each value is a full event object (see Event object above).

14.5.21 `/scheduler-event-create`

Creates a new event.

Params:

```
{
  "name": "Morning announcement",
  "eventType": "weeklyAtTimes",
  "enabled": true,
  "actionIds": ["<actionUuid>"],
  "eventGroupId": "<eventGroupUuid>",
  "intervalId": null,
  "intervalGroupId": null,
  "firstOccurrence": "2026-01-01T00:00:00",
  "lastOccurrence": "+infinity",
  "userData": {},
  "times": ["09:00:00"],
  "days": [false, true, true, true, true, true, false]
}
```

Notes:

- `name` and `eventType` are required. All other fields are optional.
- `enabled` defaults to `true`.
- `firstOccurrence` defaults to `-infinity`, `lastOccurrence` defaults to `+infinity`.
- For `single` events, the fire time is taken from `firstOccurrence`.
- Type-specific fields depend on `eventType`. See Event object for details.
- References by name are supported: `actionNames`, `eventGroupName`, `intervalName`, `intervalGroupName`.
- `intervalId` and `intervalGroupId` are mutually exclusive.

Response:

```
{ "id": "<eventUuid>" }
```

14.5.22 `/scheduler-event-modify`

Modifies an existing event. All fields except the selector are optional.

Params:

```
{
  "eventId": "<eventUuid>",
  "name": "Updated Event",
  "enabled": false,
  "actionIds": ["<actionUuid>"],
  "eventGroupId": null,
  "intervalId": "<intervalUuid>",
  "intervalGroupId": null,
  "firstOccurrence": "2026-06-01T00:00:00",
  "lastOccurrence": "+infinity",
  "userData": {},
  "eventType": "dailyAtTimes",
  "times": ["08:00:00", "16:00:00"]
}
```

Notes:

- Selector: `eventId` (UUID) or `eventName` (display name).
- Send `null` for `eventGroupId`, `intervalId`, or `intervalGroupId` to clear that reference.
- If `eventType` is provided, all type-specific fields for the new type must also be provided.
- `intervalId` and `intervalGroupId` cannot both be set at the same time.

14.5.23 `/scheduler-event-remove`

Removes an event.

Params:

```
{ "eventId": "<eventUuid>" }
```

Notes:

- Selector: `eventId` (UUID) or `eventName` (display name).

15 Triggers

A **Trigger** monitors a hardware input channel and fires one or more **Actions** when a defined condition is met. Triggers are the bridge between physical signals (button presses, relay contacts, alarm panel outputs) and software behavior in SoundCoreHero.

Each trigger specifies the input source, the condition to detect, debounce and retrigger timing, and the set of Actions to run when the condition fires. Triggers are stored per project and reloaded on project change.

Connections in the system:

```
Hardware input ▶ Trigger ▶ Action(s)
```

Trigger hardware is configured globally in the device `config.json` via the `gpioType` field:

`gpioType`	Description
`null`	GPIO disabled — no triggers
`default`	File-based GPIO; each trigger reads its own `sourcePath`
`wiktor`	2-channel GPIO via `/opt/SoundCoreHero/gpio/{channel}`
`gp80001p`	Numato Lab 8 Channel USB GPIO Module on `/dev/ttyACM0`

15.1 Trigger object

Every trigger is identified by a UUID. The full trigger object returned by `/trigger-get-status-all` and pushed via WebSocket has the following structure:

```
{
  "name": "Door Contact",
  "enabled": true,
  "direction": "input",
  "valueKind": "digital",
  "sourcePath": "",
  "channel": 1,
  "normalDigitalValue": false,
  "triggerCondition": "active",
  "analogActiveWhen": "below",
  "threshold": 0.0,
  "debounceMs": 100,
  "retriggerSeconds": 30,
  "actionIds": ["<actionUuid>"],
  "order": 0,
  "available": true,
  "active": false,
  "lastRawValue": "0",
  "lastNumericValue": 0.0,
  "lastError": null,
  "lastChangedAtMs": 1781114207292,
  "lastTriggeredAtMs": null
}
```


15.1.1 Field reference

Field	Type	Description
`name`	string	Display name. Written to logs when the trigger fires. Must be unique.
`enabled`	boolean	When `false`, the trigger is not polled and will not fire.
`direction`	string	Always `"input"` for hardware-monitored triggers.
`valueKind`	string	`"digital"` or `"analog"`. Analog is supported for `gpioType: default` and `gpioType: gp80001p`.
`sourcePath`	string	File path read for `gpioType: default`. Empty for hardware channel types.
`channel`	integer | null	Hardware channel number (1-indexed). Used by `wiktor` (1–2) and by `gp80001p` inputs (1–4, mapped internally to physical channels / ADC inputs `0..3`).
`normalDigitalValue`	boolean	Expected digital value while the connected system is idle (`false` = LOW, `true` = HIGH). For digital triggers, the input is considered **active** whenever the current value differs from `normalDigitalValue`.
`triggerCondition`	string	The condition that causes the trigger to fire (see Conditions).
`analogActiveWhen`	string	For analog triggers only: defines whether the signal is considered ON when the value is `"above"` or `"below"` `threshold`. Ignored for digital triggers. Default: `"below"`.

Field	Type	Description
`threshold`	float	Numeric threshold for analog conditions. `triggerCondition` is evaluated against the analog active/inactive state derived from `analogActiveWhen` and `threshold`. For `gp80001p`, API and project data use raw ADC values in the `0..1023` range. The UI converts these values to `0..5 V` for editing and display.
`debounceMs`	integer	Minimum time in ms the condition must remain stable before firing. `0` = no debounce.
`retriggerSeconds`	integer	Minimum time in seconds before the same trigger can fire again. For `active` and `inactive`, this is also the repeat interval while the condition remains true and must be greater than `0`.
`actionIds`	string[]	UUIDs of Actions to run when the trigger fires.
`order`	integer	Position in the trigger list (0-based).
`available`	boolean	`true` when the hardware read succeeded on the last poll cycle.
`active`	boolean	`true` when the input is currently in its **active** state. For digital inputs this means the current value differs from `normalDigitalValue`; for analog inputs it means the current value is on the active side of `threshold`.
`lastRawValue`	string | null	Last raw value read from the input source.
`lastNumericValue`	float | null	Last parsed numeric value (1.0 = HIGH/on, 0.0 = LOW/off for digital; `0..1023` raw ADC count for `gp80001p` analog).
`lastError`	string | null	Error message from the last failed hardware read, or `null`.
`lastChangedAtMs`	integer | null	Unix timestamp (ms) when the value last changed.
`lastTriggeredAtMs`	integer | null	Unix timestamp (ms) when the trigger last fired.

15.1.2 Conditions

`triggerCondition`	`valueKind`	Description
`active`	`digital`	Fires repeatedly every `retriggerSeconds` while the current value differs from `normalDigitalValue`.
`inactive`	`digital`	Fires repeatedly every `retriggerSeconds` while the current value equals `normalDigitalValue`.
`risingEdge`	`digital`	Fires once when the input changes from its normal value to its active value.

`triggerCondition`	`valueKind`	Description
`fallingEdge`	`digital`	Fires once when the input changes from its active value back to its normal value.
`anyChange`	`digital`	Fires once on every transition between active and inactive.
`active`	`analog`	Fires repeatedly every `retriggerSeconds` while the analog signal is ON.
`inactive`	`analog`	Fires repeatedly every `retriggerSeconds` while the analog signal is OFF.
`risingEdge`	`analog`	Fires once when the analog signal changes from OFF to ON.
`fallingEdge`	`analog`	Fires once when the analog signal changes from ON to OFF.
`anyChange`	`analog`	Fires once on every ON/OFF transition.

For analog triggers, the input is **active** when the measured value is either above or below `threshold`, depending on `analogActiveWhen`.

All trigger conditions use debounce. Edge conditions wait for the new state to remain stable for `debounceMs` before firing. Their `retriggerSeconds` value acts only as a cooldown between transitions; a transition that occurs during the cooldown is dropped rather than delayed.

On startup or project load, edge triggers treat the first stable input state as a transition from an unknown state. This means:

- `risingEdge` fires once if the input starts active
- `fallingEdge` fires once if the input starts inactive
- `anyChange` fires once when the initial stable state is established

15.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/trigger-get-status-all`
- `/trigger-create`
- `/trigger-modify`
- `/trigger-fire`
- `/trigger-remove`

15.2.1 `/trigger-get-status-all`

Method: `GET`

Returns all triggers.

Params:

```
None
```

Response:

```
{
  "triggers": {
    "<triggerUuid>": { ... },
    "<triggerUuid>": { ... }
  }
}
```

Each value is a full trigger object (see Trigger object above).

15.2.2 `/trigger-create`

Creates a new trigger.

Params:

```
{
  "name": "Door Contact",
  "enabled": true,
  "direction": "input",
  "valueKind": "digital",
  "sourcePath": "",
  "channel": 1,
  "normalDigitalValue": false,
  "triggerCondition": "active",
  "analogActiveWhen": "below",
  "threshold": 0.0,
  "debounceMs": 100,
  "retriggerSeconds": 30,
  "actionIds": ["<actionUuid>"]
}
```

Param	Required	Default	Description
`name`	yes	—	Must be unique across triggers
`enabled`	no	`true`	
`direction`	no	`"input"`	
`valueKind`	no	`"digital"`	
`sourcePath`	no	`""`	Required when `gpioType` is `default`

Param	Required	Default	Description
`channel`	no	—	Required when `gpioType` is `wiktor` or `gp80001p`; `gp80001p` channels exposed to users are `1..4`
`normalDigitalValue`	no	`false`	Used to determine which digital value counts as active
`triggerCondition`	no	`"active"`	
`analogActiveWhen`	no	`"below"`	Analog only: `"above"` or `"below"`
`threshold`	no	`0.0`	
`debounceMs`	no	`100`	
`retriggerSeconds`	no	`30`	Must be greater than `0` for `active` and `inactive`
`actionIds`	no	`[]`	Array of action UUIDs or names via `actionNames`

Response:

```
{ "id": "<triggerUuid>" }
```

15.2.3 `/trigger-modify`

Modifies an existing trigger. All fields except the selector are optional — only provided fields are updated.

Params:

```
{
  "triggerId": "<triggerUuid>",

  "name": "New name",
  "enabled": false,
  "channel": 2,
  "normalDigitalValue": true,
  "triggerCondition": "risingEdge",
  "analogActiveWhen": "below",
  "threshold": 0.0,
  "debounceMs": 200,
  "retriggerSeconds": 5,
  "actionIds": ["<actionUuid>"],
  "order": 1
}
```

Notes:

- `triggerId` or `triggerName` must be provided to identify the trigger.
- `actionNames` can be used instead of (or alongside) `actionIds`.
- `order` must be within `0..triggers.len()-1`.

15.2.4 `/trigger-fire`

Manually fires a trigger's actions immediately, bypassing condition and timing checks.

Params:

```
{ "triggerId": "<triggerUuid>" }
```

`triggerId` or `triggerName` can be used.

15.2.5 `/trigger-remove`

Removes a trigger.

Params:

```
{ "triggerId": "<triggerUuid>" }
```

`triggerId` or `triggerName` can be used.

15.3 WebSocket messages

Trigger state is broadcast over the control WebSocket. Refer to WebSockets for general message format.

Message key	Type	Description
`triggers`	`Record<string, TriggerObject>`	Full state snapshot sent on connection and project change.
`triggersUpdate`	`Record<string, JsonPatch>`	Incremental patch sent when any trigger field changes.

16 WebSockets

SoundCoreHero uses WebSockets to push real-time state changes and event notifications to connected clients. Clients do not need to poll — the server proactively sends updates whenever resources change.

For raw audio frame streaming (monitoring, visualization) see WebSockets — Audio.

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

16.1 Address

Default address: `wss://<device-ip>/ws/`

Plain WebSocket access can be enabled separately for legacy integrations. Plain WebSocket address: `ws://<device-ip>:7879`

16.2 Overview

The control WebSocket connection is session-scoped. Each client opens a connection, authenticates with a `(sessionId, receiverId)` pair, and then passively receives messages pushed by the server. There is no request/response pattern on this socket — all control operations are performed over HTTP.

A single session may have multiple concurrent receivers open simultaneously, including browser tabs and non-browser clients. Each receiver establishes its own independent WebSocket connection. The server routes messages either to all receivers in a session, or to a single specific receiver, depending on the message type.

16.3 Connection flow

1. Open a WebSocket connection to `wss://<device-ip>/ws/`.
2. The server immediately pushes an unauthorized status to indicate it is waiting for credentials:

```
{ "status": "unauthorized" }
```

3. The client sends a single JSON text frame with its session and receiver identity:

```
{
  "sessionId": "<sessionUuid>",
  "receiverId": "<receiverUuid>"
}
```

Both values must be RFC 4122 UUIDs. They should be generated by the client and remain stable for the lifetime of the session/receiver pair while connected — reuse the same values if reconnecting.

4. The server validates the session against the users manager.

- **Success** — the server replies with the list of pages the session is allowed to access, and begins sending push messages:

  ```
  { "allowedPages": { ... } }
  ```

- **Failure** — the server replies `{ "status": "failed" }` and closes the connection. This typically means the session is not logged in or has expired.

After the handshake the connection is receive-only. Any further text messages sent by the client will be ignored.

16.4 Sessions and receivers

- A **session** represents a logged-in user. All receivers sharing a `sessionId` belong to the same user.
- A **receiver** is a single independent client connection.
- Session-wide messages (e.g. resource snapshots on login) are delivered to **all receivers** in the session.
- Receiver-specific messages (e.g. meter data) are delivered only to the **receiver that subscribed** to them.

Use `crypto.randomUUID()` (browser) or an equivalent library to generate the UUIDs.

16.5 Received messages

All messages are JSON text frames. Each message is a JSON object with a single top-level key that identifies the message type; the value under that key is the payload.

16.5.1 Full state snapshots

Sent immediately after a successful connection handshake, and again whenever the project is reloaded or the resource collection is rebuilt. These messages allow the client to initialise its local state in one shot.

The payload is an object whose keys are resource UUIDs and whose values are the resource objects.

Key	Description
`announcements`	All announcements
`actions`	All actions
`actionGroups`	All action groups
`controllers`	All controllers
`inputs`	All inputs
`players`	All players
`playlists`	All playlists
`speakers`	All speakers
`zones`	All zones
`dso`	DSO processor state

Example:

```
{
  "players": {
    "3fa85f64-5717-4562-b3fc-2c963f66afa6": {
      "name": "Background Music",
      "state": "playing",
      ...
    },
    "7c9e6679-7425-40de-944b-e07fc1f90ae7": {
      "name": "Jingles",
      "state": "stopped",
      ...
    }
  }
}
```

16.5.2 Incremental updates

Sent whenever a resource changes (created, modified, or removed). The payload has the same shape as the full snapshot, but contains only the affected resource(s). The client should merge these into its local state rather than replacing it entirely.

Key	Description
`announcementsUpdate`	Changed announcement(s)
`actionsUpdate`	Changed action(s)
`actionGroupsUpdate`	Changed action group(s)
`controllersUpdate`	Changed controller(s)
`inputsUpdate`	Changed input(s)
`playersUpdate`	Changed player(s)
`playlistsUpdate`	Changed playlist(s)
`speakersUpdate`	Changed speaker(s)
`zonesUpdate`	Changed zone(s)
`dsoUpdate`	Updated DSO state

16.5.3 Meter messages

Sent periodically to receivers that have an active meters subscription. See Subscriptions for how to subscribe.

```
{ "meters": { ... } }
```

The payload contains level data for all subscribed nodes (players, zones, speakers, inputs, DSO). Meter messages are sent to the specific receiver that subscribed — they are not broadcast to all receivers in a session.

Receiving meter data requires both:

1. a valid logged-in session, because the WebSocket handshake validates `sessionId` against the users manager, and
2. an already-established control WebSocket connection using the same `(sessionId, receiverId)` as the HTTP subscription call.

Meter messages are **coalesced** server-side: only the most recent meter update is buffered per receiver. If the client cannot keep up with the send rate, intermediate meter frames are silently dropped. Only the latest reading matters.

16.5.4 System messages

These messages are broadcast to all connected sessions.

Key	Description
`cpuUsage`	Periodic CPU utilization report
`internetStatus`	Current internet connectivity status (`true` / `false`)
`softwareUpdateStatus`	Progress and result of a software update operation
`systemError`	A system-level error that the user should be aware of
`actionError`	An error produced while executing an action
`config`	The current device configuration
`projects`	The list of available projects on this device
`channelsLimit`	The configured and maximum allowed channel count

16.5.5 Server time

Sent to all receivers approximately once per minute. Clients can use this to synchronize display clocks without polling.

```
{ "serverTime": "2026-02-28T14:30:00Z" }
```

The value is an ISO 8601 UTC timestamp.

16.5.6 Session end

When a session is terminated server-side (logout or expiry), the server broadcasts a goodbye message to all receivers in that session and then drops the connections:

```
{ "message": "goodbye" }
```

Clients should treat receipt of this message as a signal to redirect to the login page.

16.6 Message delivery guarantees

The server maintains two separate queues per tab:

Queue	Used for	Guarantee
Events queue	All messages except meters	Every message is delivered in order; none are dropped
State queue	Meter messages only	Only the most recent message is delivered; outdated intermediates are discarded

If the events queue fills up (capacity: 1000 messages), the connection is considered unhealthy and may be closed.

16.7 Reconnection

The server does not automatically restore subscriptions after a reconnect. If the connection drops:

1. Re-establish the WebSocket connection and re-authenticate with the same `sessionId` and `receiverId`.
2. The server will send full state snapshots again.
3. Re-subscribe to any meters subscriptions via HTTP (see Subscriptions).

16.8 Related

- WebSockets — Audio — raw audio frame streaming
- Subscriptions — meters and audio subscription HTTP endpoints

17 WebSockets Audio

The Audio WebSocket server streams raw audio frames in real time to connected clients. It is designed for use cases such as level visualisation, waveform display, monitoring, and custom audio analysis.

For general state updates and event notifications see WebSockets.

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

17.1 Address

Default address: `wss://<device-ip>/ws-audio/`

Plain audio WebSocket access can be enabled separately for legacy integrations. Plain audio WebSocket address: `ws://<device-ip>:7979`

17.2 Overview

The Audio WebSocket is a **unidirectional, receiver-specific stream**. Each `(sessionId, receiverId)` pair can subscribe to exactly one audio source at a time. Frames for that source are delivered only to the receiver that subscribed — audio is never broadcast to multiple receivers simultaneously.

Subscribing to an audio source and unsubscribing from one are done over HTTP. This WebSocket connection only receives data; it carries no commands.

17.3 Connection flow

1. Open a WebSocket connection to `wss://<device-ip>/ws-audio/`.
2. The server immediately pushes an unauthorized status:

```
{ "status": "unauthorized" }
```

3. The client sends a single JSON text frame identifying itself:

```
{
  "sessionId": "<sessionUuid>",
  "receiverId": "<receiverUuid>"
}
```

Use the same UUIDs as the control WebSocket connection for the same receiver. The server does **not** perform a permission check on the audio socket itself, but audio delivery still depends on a valid logged-in session because HTTP subscription endpoints require session-based authentication.

4. The connection is now open and ready to receive audio frames.

At this point the connection is idle. No audio is delivered until the client also calls `/audio-subscribe` over HTTP using the same `receiverId` and an authenticated request carrying the same `sessionId` (see Subscriptions).

After sending the auth frame, no further messages should be sent by the client. Any additional text messages will be ignored.

17.4 Subscribing to an audio source

Audio delivery is controlled via HTTP, not over this WebSocket. The WebSocket only carries the resulting binary stream.

To start receiving audio:

```
POST /audio-subscribe
Content-Type: application/json

{
  "receiverId": "<receiverUuid>",
  "zoneId": "<zoneUuid>",
  "stage": { "name": "output" }
}
```

To stop receiving audio:

```
POST /audio-unsubscribe
Content-Type: application/json

{
  "receiverId": "<receiverUuid>"
}
```

Full details of available source types and zone stages are documented in Subscriptions.

17.5 Received messages

17.5.1 Audio frames

Audio frames are delivered as **binary WebSocket frames**. The binary payload is a packed array of interleaved stereo 32-bit float PCM samples:

```
[L0, R0, L1, R1, L2, R2, ..., Ln, Rn]
```

Property	Value
Format	IEEE 754 single-precision float (f32)
Byte order	Little-endian
Channels	2 (stereo, interleaved)
Frame size	Variable; depends on the audio engine buffer size

To decode a frame in JavaScript:

```
socket.onmessage = (event) => {
  const buffer = await event.data.arrayBuffer();
  const samples = new Float32Array(buffer);
  // samples[0] = L0, samples[1] = R0, samples[2] = L1, ...
};
```

To decode a frame in Python:

```
import struct

def decode_frame(data: bytes) -> list[tuple[float, float]]:
    count = len(data) // 4
    floats = struct.unpack(f"<{count}f", data)
    return list(zip(floats[0::2], floats[1::2]))  # [(L0, R0), (L1, R1), ...]
```

Frames arrive at the audio engine's processing rate. If the client cannot consume frames fast enough, newer frames may overwrite older ones in the server's buffer — only the latest is retained per receiver.

17.5.2 No text messages

The audio WebSocket does not send any JSON text messages. All data is binary.

17.6 Session end

When a session is terminated server-side (logout or expiry), the server immediately drops all audio connections belonging to that session. The WebSocket will be closed from the server side.

Clients should listen for the `close` event and attempt reconnection if appropriate:

```
socket.onclose = () => {
  // Session may have ended or connection dropped — reconnect if needed
};
```

17.7 Reconnection

The audio server does not retain subscription state across reconnects. After reconnecting:

1. Re-send the auth frame with `sessionId` and `receiverId`.
2. Re-call `/audio-subscribe` over HTTP to restart delivery.

17.8 Related

- WebSockets — control WebSocket for state updates and events
- Subscriptions — HTTP endpoints for subscribing to audio and meters

18 Subscriptions

SoundCoreHero streams live audio data to connected clients over WebSocket. There are three subscription systems:

- **Meters subscriptions** — deliver periodic level/metering data for players, zones, speakers, inputs, and DSO.
- **Audio frame subscriptions** — deliver raw audio frames from a single selected node for monitoring or visualization.
- **Microphone send subscriptions** — accept browser microphone audio and inject it into a selected input or DSO, replacing the physical capture source while active.

Both systems are scoped per `(sessionId, receiverId)`.

`sessionId` comes from the authenticated request context.

`receiverId` is the receiver identifier. Any client may generate a stable UUID and use it to distinguish one receiver from another within the same session (for example, different browser tabs or devices).

To actually receive subscription data, the client must already have an authenticated WebSocket connection open with the same `(sessionId, receiverId)`. A user therefore needs to be logged in first to obtain a valid `sessionId`.

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

18.1 Endpoints

- `/meters-subscribe`
- `/meters-unsubscribe`
- `/audio-subscribe`
- `/audio-unsubscribe`
- `/microphone-subscribe`
- `/microphone-unsubscribe`

18.2 Meters subscriptions

18.2.1 `/meters-subscribe`

Subscribes the given authenticated session and `receiverId` to metering data for the specified objects.

Params:

```
{
  "receiverId": "<receiverUuid>",

  "speakerIds": ["<speakerUuid>"],
  "zoneIds": ["<zoneUuid>"],
  "playerIds": ["<playerUuid>"],
  "inputIds": ["<inputUuid>"],

  "dso": true
}
```

Notes:

- Any of `speakerIds/Names`, `zoneIds/Names`, `playerIds/Names`, `inputIds/Names` may be provided.
- If no subscriptions are included, it behaves as unsubscribe for that `(sessionId, receiverId)`.
- The request must be authenticated with a session-based credential. API-key authentication is not sufficient for subscription endpoints.

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

18.2.2 `/meters-unsubscribe`

Unsubscribes the given authenticated session and `receiverId` from all metering data.

Params:

```
{
  "receiverId": "<receiverUuid>"
}
```

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

18.2.3 WebSocket audio transport

The dedicated audio WebSocket is bidirectional after the initial JSON authorization payload.

Authorization payload:

```
{
  "sessionId": "<sessionUuid>",
  "receiverId": "<receiverUuid>"
}
```

Binary messages from server to client carry monitored audio as interleaved stereo `Float32` PCM.

Binary messages from client to server carry microphone audio as mono `Float32` PCM. The backend reads these samples only when an active microphone send subscription exists for the same `(sessionId, receiverId)`.

18.3 Audio frame subscriptions

Each `(sessionId, receiverId)` can subscribe to exactly one audio source at a time. Subscribing to a new source replaces any existing subscription for that receiver.

18.3.1 `/audio-subscribe`

Params (subscribe to a zone output):

```
{
  "receiverId": "<receiverUuid>",

  "zoneId": "<zoneUuid>",
  "stage": { "name": "output" }
}
```

Params (subscribe to a zone stage requiring inputId):

```
{
  "receiverId": "<receiverUuid>",

  "zoneId": "<zoneUuid>",
  "stage": { "name": "postInputMixer", "inputId": "<inputUuid>" }
}
```

Stage options for zones:

- `output` (default)
- `postInputMixer` (requires `stage.inputId`)
- `postPlayerMixer`
- `postAnnouncementMixer`
- `postDsoMixer`

Other node types (pick exactly one):

```
{ "receiverId": "<receiverUuid>", "playerId": "<playerUuid>" }
```

```
{ "receiverId": "<receiverUuid>", "inputId": "<inputUuid>" }
```

```
{ "receiverId": "<receiverUuid>", "speakerId": "<speakerUuid>" }
```

```
{ "receiverId": "<receiverUuid>", "dso": true }
```

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

18.4 Microphone send subscriptions

Each `(sessionId, receiverId)` can send microphone audio to exactly one target at a time.

Only one active sender may target a given input or DSO at once.

While active, the selected target receives browser microphone audio instead of the physical capture signal. When no microphone samples are available for a render block, silence is injected for that target.

18.4.1 `/microphone-subscribe`

Params (send microphone to an input):

```
{
  "receiverId": "<receiverUuid>",
  "inputId": "<inputUuid>"
}
```

Params (send microphone to DSO):

```
{
  "receiverId": "<receiverUuid>",
  "dso": true
}
```

Notes:

- Only `inputId` or `dso` is supported.
- A second sender targeting the same input or DSO is rejected.
- The request must be authenticated with a session-based credential. API-key authentication is not sufficient for subscription endpoints.

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

18.4.2 `/microphone-unsubscribe`

Stops browser microphone injection for the given authenticated session and `receiverId`.

Params:

```
{
  "receiverId": "<receiverUuid>"
}
```

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

18.4.3 `/audio-unsubscribe`

Unsubscribes the given authenticated session and `receiverId` from audio frame streaming.

Params:

```
{
  "receiverId": "<receiverUuid>"
}
```

Error behavior note:

- On failure returns HTTP 200 with `{ "error": "..." }`.

19 File Manager

File Manager provides browsing, upload/download and filesystem operations under the configured `rootFolder`.

Notes:

- Paths are interpreted relative to `rootFolder`. Attempts to access outside `rootFolder` are denied.
- The `.text_to_speech_tmp` folder (ElevenLabs previews) is not accessible via File Manager.

19.1 Safety checks

Some operations verify that files are not currently in use by playlists or announcements before proceeding:

- **Remove** — checks that the path is not used in any playlist or announcement. Skipped when `force: true`.
- **Move** — checks that single files can be safely moved (folders are not checked). Skipped when `force: true`.
- **Rename** — no safety check; paths in playlists, players, and announcements are automatically updated.

19.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

Method	Endpoint
`GET`	`/filemanager-list-folder`
`GET`	`/filemanager-download`
`GET`	`/filemanager-get-disk-space`
`POST`	`/filemanager-upload-file`
`POST`	`/filemanager-create-folder`
`POST`	`/filemanager-copy-file-folder`
`POST`	`/filemanager-move-file-folder`
`PUT`	`/filemanager-rename-file-folder`
`DELETE`	`/filemanager-remove-file-folder`

19.2.1 `/filemanager-list-folder`

Lists a folder.

Method: `GET`

Query params:

- `path` (optional) - relative path to list. If omitted, lists the `rootFolder`.
- `search` (optional) - substring filter (case-insensitive) applied to entry names.
- `recursive` (optional) - `true|false`, default `false`. When `true`, lists recursively.

Examples:

- List root:

`GET /filemanager-list-folder?path=`

- List recursively with search:

`GET /filemanager-list-folder?path=Music&recursive=true&search=demo`

Responses:

- Success

```
{
  "files": [
    { "path": "Music/Demo 2/track.mp3", "duration": 123.45 },
    { "path": "Music/empty/file.wav", "duration": null }
  ],
  "folders": [
    "Music/Demo 2",
    "Music/empty"
  ]
}
```

19.2.2 `/filemanager-download`

Downloads a file or a folder.

- If `path` is a file, returns the file as `application/octet-stream`.
- If `path` is a directory, returns a zip as `application/zip`.

Method: `GET`

Query params:

- `path` (required) - relative path to a file or directory.

Example:

`GET /filemanager-download?path=Music/Demo%202/track.mp3`

Responses:

- Success: binary file or zip with `Content-Disposition: attachment`.

19.2.3 `/filemanager-get-disk-space`

Returns filesystem disk space statistics for the filesystem containing `rootFolder`.

Method: `GET`

Params:

```
None
```

Responses:

- Success

```
{
  "diskSpace": {
    "total": 123456789,
    "used": 23456789,
    "available": 100000000,
    "unit": "kB"
  }
}
```

19.2.4 `/filemanager-upload-file`

Uploads and validates an audio file into a target directory.

Missing target subdirectories are created automatically, so clients can preserve dropped or selected folder structure by sending each file with its destination folder in `path`.

Content-Type: `multipart/form-data`

Form fields:

- `file` (required) - the uploaded file.
- `path` (required) - target directory (relative to `rootFolder`).

Query params:

- `tryFix` (optional) - `true|false`, default `false`. If validation fails and `tryFix=true`, server attempts to repair the audio file.

19.2.5 `/filemanager-create-folder`

Creates a folder.

Params:

```
{
  "path": "Music/New Folder"
}
```

19.2.6 `/filemanager-copy-file-folder`

Copies a single file. Note: directory copying is not supported — only individual files can be copied.

Params:

```
{
  "pathFrom": "Music/Demo 2/track.mp3",
  "pathTo": "Music/Backup/track.mp3"
}
```

19.2.7 `/filemanager-move-file-folder`

Moves one or more files/folders into a destination folder. The destination folder must already exist.

Params:

```
{
  "pathTo": "Music/Archive",
  "pathFrom": [
    "Music/Demo 2/track.mp3",
    "Music/Demo 2/track2.mp3"
  ],
  "force": false
}
```

- `pathTo` (string, required) — destination folder.
- `pathFrom` (string[], required) — list of source paths to move.
- `force` (boolean, optional, default `false`) — if `true`, skips the safety check for files used in playlists.

19.2.8 `/filemanager-rename-file-folder`

Renames or moves a file/folder from one path to another.

Method: `PUT`

Params:

```
{
  "pathFrom": "Music/Demo 2/track.mp3",
  "pathTo": "Music/Demo 2/track-renamed.mp3"
}
```

19.2.9 `/filemanager-remove-file-folder`

Removes one or more files/folders.

Method: `DELETE`

Params:

```
{
  "path": [
    "Music/Demo 2/track.mp3",
    "Music/Old Folder"
  ],
  "force": false
}
```

- `path` (string[], required) — list of paths to remove.
- `force` (boolean, optional, default `false`) — if `true`, skips the safety check for files/folders used in playlists or announcements.

20 Users Manager

Users Manager handles user accounts, permissions, login sessions, and password setup/reset.

Where an endpoint needs to select a specific user, you can provide either `userId` or `userName` unless stated otherwise.

20.1 User object

Users are returned as an object keyed by `userId`.

Example:

```
{
  "users": {
    "d4745f1f-a808-4da5-90ca-c8f1b762043c": {
      "email": "admin@soundcorehero.com",
      "name": "Admin",
      "userType": "admin",
      "allowedPages": {
        "dashboard": "all",
        "players": "all",
        "mixer": "all",
        "controlActions": "all",
        "announcements": "all",
        "fileManager": "all",
        "scheduleCalendar": "all",
        "scheduleTimeIntervals": "all",
        "scheduleEvents": "all",
        "setupPlayers": "all",
        "setupPlaylists": "all",
        "setupInputs": "all",
        "setupZones": "all",
        "setupSpeakers": "all",
        "setupActions": "all",
        "setupControllers": "all",
        "systemSettings": "all",
        "systemNetwork": "all",
        "systemUsers": "all",
        "systemLogs": "all"
      },
      "autoLogout": true,
      "order": 0,
      "since": 1773937424
    }
  }
}
```

20.1.1 Field reference

Field	Type	Description
`email`	string	User login email address. Stored and matched case-insensitively.
`name`	string	Display name shown in the UI.
`userType`	string	User role: `admin`, `user`, or `panel`.
`allowedPages`	object	Per-page permission settings (see `allowedPages`).
`autoLogout`	boolean	Whether the session should auto-logout after inactivity.
`order`	integer	Position in the user list (0-based).
`hasApiKey`	boolean	Whether the user currently has an API key configured. The raw API key is never returned by list endpoints.
`since`	integer | null	Unix timestamp when the user logged in, or `null` if the user is not currently active. Present in list endpoints only.

20.1.2 `allowedPages`

The `allowedPages` object defines what areas of the system a user can access.

Each permission entry can be one of these values:

Value	Meaning
`"all"`	Full access to that page or feature.
`"none"`	No access.
`{ "given": ["<uuid>", ...] }`	Access only to selected objects. This is mainly used for object-based pages such as players, mixers, or playlists.

Supported permission keys:

- `dashboard`
- `players`
- `mixer`
- `controlActions`
- `announcements`
- `fileManager`
- `scheduleCalendar`
- `scheduleTimeIntervals`
- `scheduleEvents`
- `setupPlayers`
- `setupPlaylists`
- `setupInputs`
- `setupZones`
- `setupSpeakers`
- `setupActions`

- `setupControllers`
- `systemSettings`
- `systemNetwork`
- `systemUsers`
- `systemLogs`

Example with restricted access:

```
{
  "dashboard": "none",
  "players": {
    "given": [
      "6fdc39f8-d35f-4461-aa89-f17fb00db998",
      "b026e48d-95f6-49e2-a39f-55b14656a375"
    ]
  },
  "mixer": {
    "given": [
      "17e1565d-d4d2-403f-97c0-96e19796719a"
    ]
  },
  "controlActions": "all",
  "announcements": "none",
  "fileManager": "none",
  "scheduleCalendar": "none",
  "scheduleTimeIntervals": "none",
  "scheduleEvents": "none",
  "setupPlayers": "none",
  "setupPlaylists": {
    "given": [
      "ca31ceb2-d8a5-4b99-a9cc-ced886c56c7b",
      "bbce3b20-da71-483c-b584-8b22100b8428"
    ]
  },
  "setupInputs": "none",
  "setupZones": "none",
  "setupSpeakers": "none",
  "setupActions": "none",
  "setupControllers": "none",
  "systemSettings": "none",
  "systemNetwork": "none",
  "systemUsers": "none",
  "systemLogs": "none"
}
```

20.2 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/usersmanager-get-users`
- `/usersmanager-get-active-users`
- `/usersmanager-create`
- `/usersmanager-modify`

- `/usersmanager-delete`
- `/usersmanager-login`
- `/usersmanager-logout`
- `/usersmanager-create-api-key`
- `/usersmanager-remove-api-key`
- `/usersmanager-generate-token`
- `/usersmanager-reset-password`

20.2.1 `/usersmanager-get-users`

Method: `GET`

Returns all users. The response includes the `since` field for each user: a Unix timestamp when logged in, or `null` when not currently active.

Params:

```
None
```

Response:

```
{
  "users": {
    "d4745f1f-a808-4da5-90ca-c8f1b762043c": {
      "email": "admin@soundcorehero.com",
      "name": "Admin",
      "userType": "admin",
      "allowedPages": {
        "dashboard": "all",
        "players": "all",
        "mixer": "all",
        "controlActions": "all",
        "announcements": "all",
        "fileManager": "all",
        "scheduleCalendar": "all",
        "scheduleTimeIntervals": "all",
        "scheduleEvents": "all",
        "setupPlayers": "all",
        "setupPlaylists": "all",
        "setupInputs": "all",
        "setupZones": "all",
        "setupSpeakers": "all",
        "setupActions": "all",
        "setupControllers": "all",
        "systemSettings": "all",
        "systemNetwork": "all",
        "systemUsers": "all",
        "systemLogs": "all"
      },
      "autoLogout": true,
      "hasApiKey": false,
      "order": 0,
      "since": 1773937424
    },
    "8097c866-79bb-4433-a7da-d0f810b211c9": {
      "email": "panel@panel.com",
      "name": "panel",
      "userType": "panel",
      "allowedPages": {
        "dashboard": "none",
        "players": {
          "given": [
            "6fdc39f8-d35f-4461-aa89-f17fb00db998",
            "b026e48d-95f6-49e2-a39f-55b14656a375"
          ]
        },
        "mixer": {
          "given": [
            "17e1565d-d4d2-403f-97c0-96e19796719a"
          ]
        },
        "controlActions": "all",
        "announcements": "none",
        "fileManager": "none",
        "scheduleCalendar": "none",
        "scheduleTimeIntervals": "none",
        "scheduleEvents": "none",
        "setupPlayers": "none",
        "setupPlaylists": {
          "given": [
            "ca31ceb2-d8a5-4b99-a9cc-ced886c56c7b",
            "bbce3b20-da71-483c-b584-8b22100b8428"
          ]
        },
        "setupInputs": "none",
```

20.2.2 `/usersmanager-get-active-users`

Method: `GET`

Returns only users who currently have an active session.

Params:

```
None
```

Response:


```
{
  "users": {
    "d4745f1f-a808-4da5-90ca-c8f1b762043c": {
      "email": "admin@soundcorehero.com",
      "name": "Admin",
      "userType": "admin",
      "allowedPages": {
        "dashboard": "all",
        "players": "all",
        "mixer": "all",
        "controlActions": "all",
        "announcements": "all",
        "fileManager": "all",
        "scheduleCalendar": "all",
        "scheduleTimeIntervals": "all",
        "scheduleEvents": "all",
        "setupPlayers": "all",
        "setupPlaylists": "all",
        "setupInputs": "all",
        "setupZones": "all",
        "setupSpeakers": "all",
        "setupActions": "all",
        "setupControllers": "all",
        "systemSettings": "all",
        "systemNetwork": "all",
        "systemUsers": "all",
        "systemLogs": "all"
      },
      "autoLogout": true,
      "order": 0,
      "since": 1773937424
    }
  }
}
```


20.2.3 `/usersmanager-create`

Creates a new user account.

New users are created without a password. To allow login, first call `/usersmanager-generate-token`, then use the returned token with `/usersmanager-reset-password`.

Params:

```
{
  "name": "Operations Panel",
  "email": "panel@example.com",
  "userType": "panel",
  "autoLogout": false
}
```

Parameter notes:

- User names must be unique.
- `userType` must be one of `admin`, `user`, or `panel`.
- Email addresses must be unique.
- Initial permissions are assigned automatically from `userType`.

Response:

```
None
```

20.2.4 `/usersmanager-modify`

Updates a user. Only the supplied fields are changed.

Params:

```
{
  "userName": "panel",
  "name": "Panel A",
  "autoLogout": false,
  "userType": "panel",
  "allowedPages": {
    "dashboard": "none",
    "players": {
      "given": [
        "6fdc39f8-d35f-4461-aa89-f17fb00db998",
        "b026e48d-95f6-49e2-a39f-55b14656a375"
      ]
    },
    "mixer": {
      "given": [
        "17e1565d-d4d2-403f-97c0-96e19796719a"
      ]
    },
    "controlActions": "all",
    "announcements": "none",
    "fileManager": "none",
    "scheduleCalendar": "none",
    "scheduleTimeIntervals": "none",
    "scheduleEvents": "none",
    "setupPlayers": "none",
    "setupPlaylists": "none",
    "setupInputs": "none",
    "setupZones": "none",
    "setupSpeakers": "none",
    "setupActions": "none",
    "setupControllers": "none",
    "systemSettings": "none",
    "systemNetwork": "none",
    "systemUsers": "none",
    "systemLogs": "none"
  }
}
```

Notes:

- `name`, `autoLogout`, `userType`, and `allowedPages` are optional.
- If `name` is provided, it must remain unique.
- Changing a user from `user` to `admin` resets permissions to full access.
- Changing a user from `admin` to `user` resets permissions to the default non-admin set.
- The last remaining admin cannot be downgraded to `user`.

Response:

```
None
```

20.2.5 `/usersmanager-delete`

Deletes a user.

Params:

```
{ "userName": "panel" }
```

Notes:

- At least one admin user must remain.

Response:

```
None
```

20.2.6 `/usersmanager-login`

Authenticates a user and creates a new session.

If the same user is already logged in, the previous session is closed automatically and replaced with the new one.

Params:

```
{
  "email": "admin@soundcorehero.com",
  "password": "secret"
}
```

Response:

```
{
  "sessionId": "43813b30-1a17-416a-b255-0dc5abf422a6",
  "email": "admin@soundcorehero.com",
  "userId": "d4745f1f-a808-4da5-90ca-c8f1b762043c",
  "name": "Admin",
  "userType": "admin",
  "allowedPages": {
    "dashboard": "all",
    "players": "all",
    "mixer": "all",
    "controlActions": "all",
    "announcements": "all",
    "fileManager": "all",
    "scheduleCalendar": "all",
    "scheduleTimeIntervals": "all",
    "scheduleEvents": "all",
    "setupPlayers": "all",
    "setupPlaylists": "all",
    "setupInputs": "all",
    "setupZones": "all",
    "setupSpeakers": "all",
    "setupActions": "all",
    "setupControllers": "all",
    "systemSettings": "all",
    "systemNetwork": "all",
    "systemUsers": "all",
    "systemLogs": "all"
  },
  "autoLogout": true,
  "hasApiKey": false
}
```


20.2.7 `/usersmanager-create-api-key`

Creates or rotates the API key for a selected user. If the user already has an API key, it is replaced immediately.

Params:


```
{ "userName": "panel" }
```


Response:


```
{
  "apiKey": "sch_...",
  "hasApiKey": true
}
```


Notes:

- The raw API key is shown only once in this response.
- Store it securely on the client side.

20.2.8 `/usersmanager-remove-api-key`

Removes the API key for a selected user.

Params:

```
{ "userName": "panel" }
```

Response:

```
None
```

20.2.9 `/usersmanager-logout`

Logs out the currently active session for a user.

Params:

```
{ "userName": "Admin" }
```

Response:

```
None
```

20.2.10 `/usersmanager-generate-token`

Generates a password setup/reset token for a user.

Use this when creating a new account without a password, or when an existing user needs to set a new password.

Params:

```
{ "userName": "panel" }
```

Response:

```
{ "passwordResetToken": "TOKEN" }
```

20.2.11 `/usersmanager-reset-password`

Sets a new password using a token from `/usersmanager-generate-token`.

The token is single-use. After a successful password reset, that token becomes invalid.

Params:

```
{
  "passwordResetToken": "TOKEN",
  "password": "newPassword",
  "confirmPassword": "newPassword"
}
```

Response:

```
None
```

21 Project Manager

Project Manager provides CRUD for PlayerProjects (load/save/import/export).

Projects are stored as `.graph` files (JSON) inside `PlayerProjects/<name>/` directories. Each project contains the full state of all players, zones, speakers, inputs, playlists, announcements, actions, action groups, controllers, and presets.

Notes:

- `project-import` expects a `.zip` file upload.
- When autosave is enabled, the current project is saved automatically every 5 minutes.
- Loading a project sets it as the default project (auto-loaded on restart).
- If the scheduler is enabled, loading a project also restarts the scheduler service with the project's `scheduler.db`.

21.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

Method	Endpoint
`GET`	`/project-get-status-all`
`GET`	`/project-export`
`POST`	`/project-new`
`POST`	`/project-load`
`POST`	`/project-save`
`POST`	`/project-import`
`DELETE`	`/project-remove`

21.1.1 `/project-get-status-all`

Returns current and available projects.

Method: `GET`

Params:

```
None
```

Responses:

```
{
  "currentProject": "default",
  "availableProjects": ["default", "energy", "test"]
}
```

21.1.2 `/project-new`

Creates a new project folder and seeds it with the default graph.

Params:

```
{ "projectName": "newProject" }
```

21.1.3 `/project-load`

Loads a project.

Params:

```
{ "projectName": "default" }
```

21.1.4 `/project-save`

Saves the current project in-place, or saves to a new project when `projectName` is provided (save-as). Save-as fails if the target project already exists. When saving as a new project, the scheduler database (`scheduler.db`) is also copied from the current project.

Params:

- Save current:

```
{}
```

- Save as:

```
{ "projectName": "backupProject" }
```

21.1.5 `/project-remove`

Removes a project.

Method: `DELETE`

Params:

```
{ "projectName": "oldProject" }
```

21.1.6 `/project-export`

Exports a project as a zip.

Method: `GET`

Query params:

- `name` (required) - project name.

Example:

`GET /project-export?name=default`

Responses:

- Success: `200` with `application/zip` body and `Content-Disposition: attachment; filename="<name>.zip"`.

21.1.7 `/project-import`

Imports a project from a zip.

Content-Type: `multipart/form-data`

Form fields:

- `file` (required) - `.zip` file.

22 System

System endpoints provide device-level operations for managing and configuring the system and its security settings.

22.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/system-get-status-all`
- `/system-modify`
- `/system-get-timezones`
- `/system-set-timezone`
- `/system-get-ntp`
- `/system-set-ntp`

22.1.1 `/system-get-status-all`

Returns the editable device settings together with security and certificate state.

Method: `GET`

Params:

```
None
```

Response:

```
{
  "system": {
    "autosave": true,
    "defaultProject": "default",
    "productModel": "professional",
    "locationName": "Main Hall",
    "apiKeyEnabled": true,
    "authenticationEnabled": true,
    "httpServerPlainHttpEnabled": false,
    "serialPortEnabled": false,
    "tcpServerEnabled": false,
    "tcpServerPlainTcpEnabled": false,
    "triggerCodesEnabled": true,
    "restartRequired": false
  }
}
```

Notes:

- `autosave` enables or disables automatic saving of the current project state to disk.

- `defaultProject` is the name of the project loaded on startup.
- `locationName` is an optional user-friendly name of the device location.
- `apiKeyEnabled`, `authenticationEnabled`, `triggerCodesEnabled` - please refer to the Security documentation for details.
- `serialPortEnabled`, `tcpServerEnabled`, `httpServerPlainHttpEnabled`, and `tcpServerPlainTcpEnabled` - please refer to the System documentation for details.
- `restartRequired` is `true` if any of the changes require a backend restart to take effect. Restart can be triggered by the `/restart` endpoint or by rebooting the device.

22.1.2 `/system-modify`

Applies a partial update to editable system settings.

Method: `PUT`

Params:

```
{
    "autosave": true,
    "defaultProject": "default",
    "locationName": "Main Hall",
    "apiKeyEnabled": true,
    "authenticationEnabled": true,
    "httpServerPlainHttpEnabled": false,
    "serialPortEnabled": false,
    "tcpServerEnabled": false,
    "tcpServerPlainTcpEnabled": false,
    "triggerCodesEnabled": true
}
```

22.1.3 `/system-get-certificate`

Returns the currently active TLS certificate.

Method: `GET`

Params:

```
None
```

22.1.4 `/system-get-timezones`

Lists available timezones and returns the current timezone.

Method: `GET`

Params:

```
None
```

Responses:

```
{
  "timezone": "Europe/Warsaw",
  "timezones": [
    "Europe/Warsaw",
    "UTC"
  ]
}
```

Notes:

- `timezone` is the backend system timezone currently used by the device.
- Frontend clients should use this value when building timezone-sensitive scheduler occurrence queries, instead of relying on the browser or network-provided local timezone.

22.1.5 `/system-set-timezone`

Sets the system timezone.

Params:

```
{ "timezone": "Europe/Warsaw" }
```

22.1.6 `/system-get-ntp`

Returns the configured NTP server and the live time-sync state reported by the system.

Method: `GET`

Params:

```
None
```

Response:

```
{
  "ntpServer": "10.0.0.5",
  "enabled": true,
  "synchronized": true,
  "serverName": "10.0.0.5",
  "serverAddress": "10.0.0.5"
}
```

Notes:

- `ntpServer` is the configured time server (IP address or hostname). An empty string means NTP is disabled.
- `enabled` reflects whether network time synchronization is currently running on the device.
- `synchronized` is `true` once the system clock has successfully synced to the server. It can be `false` for a short period after (re)configuration while the first sync completes.
- `serverName` is the configured server; `serverAddress` is the time source the device is currently locked onto (which may resolve to a specific address when a hostname is configured).

22.1.7 `/system-set-ntp`

Sets the NTP server the device synchronizes its clock from. Intended for network-restricted deployments where the device can only reach a local (e.g. GPS-backed) time server. The setting is stored in the device configuration and persists across reboots.

While a custom server is set, the device uses **only** the configured server and does not reach out to any default/public time servers. Setting an empty server removes the custom server and restores the device's default time synchronization.

Method: `POST`

Params:

```
{ "ntpServer": "10.0.0.5" }
```

Notes:

- `ntpServer` accepts an IPv4/IPv6 address or a hostname/FQDN (no URL scheme).
- An empty string disables network time synchronization.
- The response body matches `/system-get-ntp`.

22.1.8 `/channels-limit`

Returns a snapshot of audio channel usage, showing how many input and output channels are currently in use and what the maximum allowed count is.

Params:

```
None
```

Response:

```
{
  "channelsLimit": {
    "usedInputs": 2,
    "usedOutputs": 4,
    "maxAllowed": 8
  }
}
```

22.1.9 `/restart`

Restarts the device audio service. All active playback and connections are interrupted and restored on startup.

Params:

```
None
```

23 Network

Network endpoints manage the device's network configuration.

Endpoints forward the upstream status code and response body from the network service.

23.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/network-config`
- `/network-validate`
- `/network-revert`

23.1.1 `/network-config`

Gets or sets network configuration.

Methods:

- `GET /network-config` - returns current config.
- `POST /network-config` - posts a new config (must later be validated).

23.1.1.1 GET

Params:

```
None
```

Responses:

```
{
  "mode": "direct",
  "primary": {
    "dhcp4": true,
    "ip": "192.168.0.10",
    "mask": "255.255.255.0",
    "gateway": "192.168.0.1",
    "dns1": "1.1.1.1",
    "dns2": "8.8.8.8"
  },
  "secondary": {
    "dhcp4": true
  },
  "waitingForValidation": false
}
```

23.1.1.2 POST

Params:

```
{
  "mode": "direct",
  "primary": {
    "dhcp4": true
  },
  "secondary": {
    "dhcp4": true
  }
}
```

For static IP, use:

```
{
  "mode": "direct",
  "primary": {
    "dhcp4": false,
    "ip": "192.168.0.10",
    "mask": "255.255.255.0",
    "gateway": "192.168.0.1",
    "dns1": "1.1.1.1",
    "dns2": "8.8.8.8"
  }
}
```

Responses:

- Success (upstream-dependent; may be `200` with JSON or an empty response)

23.1.2 `/network-validate`

Validates the last posted network configuration.

Params:

```
None
```

Responses:

- Success (upstream-dependent)

23.1.3 `/network-revert`

Reverts to the last known-good configuration.

Params:

```
None
```

Responses:

- Success (upstream-dependent)

24 Software Update

Software Update module allows admin user to check, download and install updates.

24.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/software-update-check-update`
- `/software-update-download-update`
- `/software-update-remove-downloaded-update`
- `/software-update-install-update`

24.1.1 `/software-update-check-update`

Responds with json data of expected client version, channels count, update size and downloaded state.

Params:

```
None
```

Responses:

```
{
    "isUpdateAvailable": true,
    "isUpdateServerReachable": true,
    "updateVersion": "3.1.6",
    "updateChannels": "16",
    "updateSize": 5269224,
    "currentVersion": "3.1.5",
    "currentChannels": "16",
    "downloaded": false
}
```

If the update server is unreachable, this endpoint still responds with `200` and sets `isUpdateServerReachable=false`.

24.1.2 `/software-update-download-update`

Downloads an update. Periodically reports current download progress by websocket. Reports `204` when download completes.

Params:

```
None
```

Responses:

`204` on completion.

Websocket:

```
{
"softwareUpdate": {
    "downloadProgress": 69224,
    "updateSize": 5269224
    }
}
```

24.1.3 `/software-update-remove-downloaded-update`

Removes the currently downloaded update (if any).

Params:

```
None
```

Responses:

`204` on completion.

24.1.4 `/software-update-install-update`

Installs an update (If an update is not pre-downloaded, it will download it first). On success SoundCoreHero exits quickly to restart.

Params:

```
None
```

Responses:

The server process exits to restart, so the HTTP connection may close without a response.

25 Logger

Logger stores recent request logs in memory and appends them to `logs/requests.jsonl`.

25.1 Endpoints

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

- `/logger-get-users-paged-filtered`

25.1.1 `/logger-get-users-paged-filtered`

Returns logs paged and optionally filtered by `originName` and/or `action` (URL).

Method: `GET`

Query params:

- `page` (required) - 1-based page number.
- `size` (required) - page size.
- `originName` (optional) - filter by origin name (user email or scheduler event name or action name).
- `action` (optional) - filter by request URL (exact match).
- `sort` (optional) - `newest` (default) or `oldest`.

Example:

```
GET /logger-get-users-paged-filtered?page=1&size=50&sort=newest
```

Responses:

- Success

```
{
  "logs": [
    {
      "id": "random_id",
      "date": "2026-01-01",
      "time": "12:34:56",
      "url": "/player-get-status-all",
      "origin": "HTTP",
      "originName": "john@example.com",
      "params": { "some": "json" }
    }
  ],
  "meta": {
    "currentPage": 1,
    "itemsPerPage": 50,
    "totalPages": 10,
    "totalItems": 500,
    "originName": null,
    "action": null
  }
}
```


Note: On error, this endpoint returns `200` with `{ "error": "error_text" }` instead of `400`.

26 TCP Server

TCP Server allows to interact with SoundCoreHero directly without using UI or HTTP requests.

By default, the TCP server is disabled.

An administrator can enable the TCP server from the Security settings. If enabled, TLS TCP is exposed by default. Administrator can enable plain TCP for legacy integrations that cannot use TLS. If so, both TLS and plain TCP are available simultaneously on different ports.

Once connected, a server expects requests and to each request it provides a response.

It is possible to send multiple requests (and to receive multiple responses) inside a single TCP stream. Multiple clients can connect simultaneously.

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

26.1 Configuration

Field	Default	Description
`tcpServerEnabled`	`false`	Enables or disables the TCP server. When enabled, TLS TCP on port 39300 becomes available
`tcpServerPlainTcpEnabl`	`false`	When `true`, the backend also accepts direct plain TCP on port 39200
`tcpServerTimeoutMs`	`0`	Read/write timeout in milliseconds. `0` means no timeout — connections stay open indefinitely

Changes to `tcpServerEnabled` and `tcpServerPlainTcpEnabled` require a backend restart. The system status response exposes this via `restartRequired`.

26.2 Request

Each request is supposed to be a proper JSON message terminated with `<CR>` character (`0x0D` or `\r`).

Every JSON request must include either top-level `apiKey` or top-level `sessionId`. In practice, TCP integrations should use `apiKey`.

```
{
    "url": "zone-get-status-all",
    "apiKey": "sch_..."
}
}\r
```

A request **must** contain a `url` field with a proper value. The leading `/` is optional — it will be prepended automatically if missing.

Some urls may provide additional params. In such case they all should be inside `params` field:

```
{
    "url": "zone-create",
    "params": {
        "name": "zone_stereo",
        "volume": 0,
        "mono": false,
        "muted": false,
        "mutedAnnouncement": false,
        "mutedPlayer": false,
        "mutedMic": false,
        "mutedDso": false,
        "mutedInputs": false,
        "inputs": null
    }
}
```

A request may contain `method` field with `"GET"`, `"POST"`, `"PUT"`, `"DELETE"` or `"OPTIONS"` value. If omitted, `"PUT"` is used as the default.

```
{
    "url": "zone-get-status-all",
    "method": "GET",
    "apiKey": "sch_..."
}
```

The maximum request size is **100 KB**.

Multiple requests can be sent back-to-back, each terminated with `<CR>`. They will be parsed and handled together.

26.3 Trigger codes

Trigger codes are meant for integrations with very simple devices that cannot send JSON messages.

If the incoming message is **not valid JSON**, the server treats it as a **trigger code** for an action. Trigger codes are plain strings (still terminated with `<CR>`).

If a matching action is found, the server executes it and responds with confirmation. Otherwise, an error response is returned.

Trigger codes have to be enabled from the Security settings.

Trigger-code strings do not require authentication.

```
PK4\r
```

26.4 Response

To each processed request the server provides a JSON response terminated with `<CR>` character (`0x0D` or `\r`).

```
{
    "statusCode": 200
}\r
```

A response **always** contains `statusCode` field with an integer value. The field imitates a HTTP response code. `200` and `204` means success, while `400` and `404` means error.

The server **may** provide an error reason in `error` field:

```
{
    "error": "Request URL must be specified!",
    "statusCode": 400
}
```

A response may contain other fields depending on the request.

```
{
    "statusCode": 200,
    "zones": []
}
```

27 Serial Port Server

Serial Port Server allows to interact with SoundCoreHero directly without using UI or HTTP requests.

The server expects requests and to each request it provides a response.

The request and response format is the same as in the TCP Server — see tcp_server.md for details on JSON structure, trigger codes, and response format.

Serial Port JSON requests must also be authenticated. Provide top-level `apiKey` for integrations, or top-level `sessionId` if a session-based client is using the serial protocol.

Plain trigger-code strings remain supported on Serial Port and do not require authentication while `triggerCodesEnabled` is enabled.

For general API conventions — transport, object identification, and standard HTTP status codes — refer to the API Overview.

27.1 Configuration

Serial Port Server is disabled by default. If needed it may be enabled by `serialPortEnabled`.

Field	Default	Description
`serialPortEnabled`	`false`	Enable or disable the serial port server
`serialPortName`	`"/dev/ttyS0"`	Serial port device path
`serialPortBaudRate`	`115200`	Baud rate for the serial connection

The maximum request size is **100 KB**, same as the TCP Server.