HIVE - Nectar API Route Catalogue

Please note that this API Route catalogue relates to **Hive Software Version 1.2.39** or later.

This document is an **engineering-level reference** for every HTTP endpoint exposed by the Hive Nectar webserver present on each Hive player. It is meant for:

- **AV Integrators** who need to trigger system-level tasks (upload files, control system parameters and manage system operation) from external tools or scripts.
- **Front-end developers** creating or maintaining UI's that need to use these routes.
- **Ops teams** who need to hit the API directly via `curl`, Postman, or their favourite automation framework.

Service assumptions

Item	Value / Notes
Base URL	`http://<device-ip>/`
Transport	Plain HTTP on the local network. If you reverse-proxy behind Nginx/Traefik and enable TLS, all routes remain identical.
Authentication	If password protection is **off** (default out-of-box) every route is open. Once a user account exists, *all* `POST` routes and any `GET`s that read sensitive data require a valid session cookie (`Set-Cookie` issued by `/login`).
Content-Type expectations	`application/json` for every `POST` unless otherwise specified. Upload endpoints expect `multipart/form-data`.
Typical response envelope	`json { "success": true, "data": { … } }` — Some older handlers return raw strings or booleans; those are noted in the table.
Error handling	Non-happy paths usually return **HTTP 500** with a plain-text stack-trace. (Legacy behaviour—plan to migrate to structured errors in a future release.)
Versioning	The API is *not* versioned; backward-compatibility breaks are rare but may happen between major Hive releases. Please report any issues you encounter to support@hive.run

Security & production notes

1. **Unauthenticated network** – If your deployment is on a guest Wi-Fi or any environment you don't fully trust, *enable password protection* immediately (`/register`).

2. **No CSRF tokens** – Front-end and server run on the same origin, so CSRF protection is currently disabled. When exposing the API to third-party sites, wrap calls in an authenticated proxy.

3. **Rate limiting / throttling** – None implemented. Batch or scripted calls should insert delays where appropriate (e.g. cloning or transcode jobs).

4. **Binary uploads flash hardware** – Routes like `/uploadEdidFile` and `/api/runFactoryReset` can reboot or brick a device if mis-used. Always verify payloads first on a staging unit.

Patch end points

GET/POST	Path	Parameters (origin → type)	Purpose / Response
GET	`/api/watchPatchDouble`	`patchPath` → query :string	Watch double parameters at patchPath. Server streams `{value}` when the double changes.
GET	`/api/watchPatchJSON`	`patchPath` → query :string	SSE; pushes `{json}` on any JSON change at patchPath.
GET	`/api/watchPatchString`	`patchPath` → query :string	SSE pushes string values. On any change at patchPath
POST	`/api/getPatchDouble`	`patchPath` → body :string	Returns `{success, value:number}`.
POST	`/api/getPatchJSON`	`patchPath` → body :string	Returns `{success, json}`.
POST	`/api/getPatchString`	`patchPath` → body :string	Returns `{success, value:string}`.

POST	`/api/setPatchDouble`	`patchPath` → body :string `value` → body :number	Writes a numeric patch value.
POST	`/api/setPatchString`	`patchPath` → body :string `value` → body :string	Writes a string patch value.
POST	`/api/updatePatchJSON`	`patchPath` → body :string `patch` → body :array	Applies JSON-patch ops (RFC 6902) and returns `{success}`.

PatchPaths

A ‘PatchPath’ is a location within the Hive renderer which can be written to or read from dynamically. You can also monitor these locations and get callbacks when the data at these locations changes.

LAYER PARAMETERS

Function	Description	PatchPath	Value range
File Select	Selects current Media File	/LAYER 1/FILE SELECT/Value	0..255: File Select
Folder Select	Selects current Media Folder	/LAYER 1/FOLDER SELECT/Value	0..255: Folder Select
In Frame	Frame number From which Media Playback should start	/LAYER 1/IN FRAME/Value	0..4294967295 Frame Number to start media from
Out Frame	Frame number at which Media Playback should end or loop	/LAYER 1/OUT FRAME/Value	0..4294967295 Frame Number to end/loop media at

Play Mode	How the media should play	/LAYER 1/PLAY MODE/Value	0: In Frame 1: Out Frame 2: Loop Forward 3: Loop Reverse 4: Play Once Forward 5: Play Once Reverse 6: Stop 7: Pause 8: Bounce (Ping-Pong) 9: Take Over Frame 10: Loop Forward with pause on zero intensity 11: Loop Reverse with pause on zero intensity 12: Play Once Forward with pause on zero intensity 13: Play Once Reverse with pause on zero intensity 15: Bounce (Ping-Pong) with pause on zero intensity 20: Synchronise to Time code 40: Loop Forward with re-trigger on intensity 41: Loop Reverse with re-trigger on intensity 42: Play Once Forward with re-trigger on intensity 43: Play Once Reverse with re-trigger on intensity 45: Bounce (Ping-Pong) with re-trigger on intensity

Play Speed	Play speed of media	/LAYER 1/PLAY SPEED/Value	0.0: Stop 0.001..0.499: Slower 0.5: 100% 0.501..1.0: Faster (up to 10x)
Scale	Zoom into or out of the media	/LAYER 1/SCALE/Value	0.0..4999: Zoom Out 0.5: 100% 0.5001..1.0: Zoom In
Framing Mode	How media should fit into output rectangle	/LAYER 1/FRAMING MODE/Value	0: Letterbox 1: Crop 2: Stretch 3: Multi Letterbox 4: Centered
Aspect Ratio	Horizontal and Vertical adjustment of the rectangular shape of the media	/LAYER 1/ASPECT RATIO/Value	0.0: No Adjustment 0..4999: Horizontal Squeeze 0.5: Center 0.501..1.0: Vertical Squeeze
Position X	Horizontal position of media	/LAYER 1/POSITION X/Value	0.0..0.4999: Left 0.5: Center 0.5001..1.0: Right
Position Y	Vertical position of media	/LAYER 1/POSITION Y/Value	0.0..0.4999: Above 0.5: Center 0.5001..1.0: Below

Rotation X	Rotate the media around the horizontal axis	/LAYER 1/ROTATION X/Value	0.0..0.25: Auto Rotate CCW (0 FAST) 0.25..0.4999: Manual Rotate CCW 0.5: No Rotation 0.5001..0.75: Manual Rotate CW 0.75..1.0: Auto Rotate CW (Hi FAST)
Rotation Y	Rotate the media around the vertical axis	/LAYER 1/ROTATION Y/Value	0.0..0.25: Auto Rotate CCW (0 FAST) 0.25..0.4999: Manual Rotate CCW 0.5: No Rotation 0.5001..0.75: Manual Rotate CW 0.75..1.0: Auto Rotate CW (Hi FAST)
Rotation Z	Rotate the media around the Z axis	/LAYER 1/ROTATION Z/Value	0.0..0.25: Auto Rotate CCW (0 FAST) 0.25..0.4999: Manual Rotate CCW 0.5: No Rotation 0.5001..0.75: Manual Rotate CW 0.75..1.0: Auto Rotate CW (Hi FAST)
Movement Speed	Reserved for future use	/LAYER 1/MOVEMENT SPEED/Value	Reserved for future use

Blend Mode	How this layer of media should be blended with any layers appearing below this layer	/LAYER 1/BLEND MODE/Value	0: ALPHA 1: ADDITIVE 2: MULTIPLY 3: DIFFERENCE 4: SCREEN 5: PRESERVE LUMA 6: RECTANGLE WIPE 7: TRIANGLE WIPE 8: MINIMUM 9: MAXIMUM 10: SUBTRACT 11: DARKEN 12: LIGHTEN 13: SOFT LIGHTEN 14: DARK LIGHTEN 15: EXCLUSION 16: RANDOM 17: RIPPLE 18: THRESHOLD 19: SINE 20: INVERT MASK 21: NOISE 22: SWIRL 23: GRADIENT 24: PIXEL SORT 25: CHECKERBOARD 26: PULSE

			27: HUE SHIFT 28: FRACTAL 29: WAVEFORM 30: RGB SPLIT 31: GLITCH
Intensity	Media Intensity / Opacity	/LAYER 1/INTENSITY/Value	0.0..1.0: Media Intensity / Opacity
Red	Red channel adjustment of media	/LAYER 1/RED/Value	0.0..0.4999: Remove Red Channel 0-99.9% 0.5: Red Channel at 100% 0.5001..1.0: Add to Red Channel 0-99.9%
Green	Green channel adjustment of media	/LAYER 1/GREEN/Value	0.0..0.4999: Remove Green Channel 0-99.9% 0.5: Green Channel at 100% 0.5001..1.0: Add to Green Channel 0-99.9%
Blue	Blue channel adjustment of media	/LAYER 1/BLUE/Value	0.0..0.4999: Remove Blue Channel 0-99.9% 0.5: Blue Channel at 100% 0.5001..1.0: Add to Blue Channel 0-99.9%
Hue	Hue adjustment of the colour of the media	/LAYER 1/HUE/Value	0.0..1.0: Hue adjust 0-360’
Saturation	Saturation adjustment of media	/LAYER 1/SATURATION/Value	0.0..0.4999: Desaturate 100-0% 0.5: Saturation not adjusted 0.5001..1.0: Over-saturate 0-100%

Contrast	Contrast adjustment of media	/LAYER 1/CONTRAST/Value	0.0..0.4999: Contrast adjust 0-100% 0.5: Contrast not adjusted 0.5001..1.0: Contrast adjust 100-200%
LUT	Select LUT from LUTS folder. See web UI Param Page for complete list available on device	/LAYER 1/LUT/Value	0..32767: Select LUT from LUT folder
Strobe	Strobe Media	/LAYER 1/STROBE/Value	0.0..0.5: On Off Strobe slow-fast 0.5..1.0: Punch Strobe slow-fast
TC Hour	Timecode Trigger point Hour. Play media on layer with respect to this start point. (Only active when TC Offsets on external clock page is set to 'Layer Param')	/LAYER 1/MTC HOUR/Value	0..24: HOUR
TC Minute	Timecode Trigger point Minute. Play media on layer with respect to this start point. (Only active when TC Offsets on external clock page is set to 'Layer Param')	/LAYER 1/MTC MINUTE/Value	0..60: MINUTE

TC Second	Timecode Trigger point Second. Play media on layer with respect to this start point. (Only active when TC Offsets on external clock page is set to 'Layer Param')	/LAYER 1/MTC SECOND/Value	0..60: SECOND
TC Frame	Timecode Trigger point Frame. Play media on layer with respect to this start point. (Only active when TC Offsets on external clock page is set to 'Layer Param')	/LAYER 1/MTC FRAME/Value	0..60: FRAME

FX1 Select	Select Effect 1. See Effects Page for parameters for each effect	/LAYER 1/FX1 SELECT/Value	0..32767: Select Effect 1 (FX1) 0 - NONE 1 - OLD TV 2 - SEPIA 3 - FEEDBACK 4 - BLUR 5 - CRYSTALISE 6 - FRACTAL SOUP 7 - RADAR 8 - PIXELISE 9 - SOFT EDGE OVAL 10 - TILE 11 - INFINITY ZOOM 12 - DOT GRID 13 - KALEIDOSCOPE 14 - MULTI MIRROR 15 - REBELLE DISTORT
FX1 Opacity	Effect 1 Opacity. Blends Effected media with original media	/LAYER 1/FX1 OPACITY/Value	0.0..1.0: FX1 Opacity 0-100%
FX1 Param 1	Effect 1 High resolution adjustment for parameter 1	/LAYER 1/FX1 PARAM 1/Value	0.0..1.0 FX1 Parameter 1
FX1 Param 2	Effect 1 High resolution adjustment for parameter 2	/LAYER 1/FX1 PARAM 2/Value	0.0..1.0 FX1 Parameter 2

FX1 Param 3	Effect 1 High resolution adjustment for parameter 3	/LAYER 1/FX1 PARAM 3/Value	0.0..1.0 FX1 Parameter 3
FX1 Param 4	Effect 1 High resolution adjustment for parameter 4	/LAYER 1/FX1 PARAM 4/Value	0.0..1.0 FX1 Parameter 4
FX1 Param 5	Effect 1 High resolution adjustment for parameter 5	/LAYER 1/FX1 PARAM 5/Value	0.0..1.0 FX1 Parameter 5
FX1 Param 6	Effect 1 High resolution adjustment for parameter 6	/LAYER 1/FX1 PARAM 6/Value	0.0..1.0 FX1 Parameter 6
FX1 Param 7	Effect 1 High resolution adjustment for parameter 7	/LAYER 1/FX1 PARAM 7/Value	0.0..1.0 FX1 Parameter 7
FX1 Param 8	Effect 1 High resolution adjustment for parameter 8	/LAYER 1/FX1 PARAM 8/Value	0.0..1.0 FX1 Parameter 8
FX1 Param 9	Effect 1 High resolution adjustment for parameter 9	/LAYER 1/FX1 PARAM 9/Value	0.0..1.0 FX1 Parameter 9
FX1 Param 10	Effect 1 High resolution adjustment for parameter 10	/LAYER 1/FX1 PARAM 10/Value	0.0..1.0 FX1 Parameter 10
FX1 Param 11	Effect 1 High resolution adjustment for parameter 11	/LAYER 1/FX1 PARAM 11/Value	0.0..1.0 FX1 Parameter 11
FX1 Param 12	Effect 1 High resolution	/LAYER 1/FX1 PARAM 12/Value	0.0..1.0 FX1 Parameter 12

	adjustment for parameter 12		
FX1 Param 13	Effect 1 High resolution adjustment for parameter 13	/LAYER 1/FX1 PARAM 13/Value	0.0..1.0 FX1 Parameter 13
FX1 Param 14	Effect 1 High resolution adjustment for parameter 14	/LAYER 1/FX1 PARAM 14/Value	0.0..1.0 FX1 Parameter 14
FX1 Param 15	Effect 1 High resolution adjustment for parameter 15	/LAYER 1/FX1 PARAM 15/Value	0.0..1.0 FX1 Parameter 15
FX1 Param 16	Effect 1 High resolution adjustment for parameter 16	/LAYER 1/FX1 PARAM 16/Value	0.0..1.0 FX1 Parameter 16

FX2 Select	Select Effect 2. See web UI Effects Page for list	/LAYER 1/FX2 SELECT/Value	0..32767: Select Effect 2 (FX2) 0 - NONE 1 - OLD TV 2 - SEPIA 3 - FEEDBACK 4 - BLUR 5 - CRYSTALISE 6 - FRACTAL SOUP 7 - RADAR 8 - PIXELISE 9 - SOFT EDGE OVAL 10 - TILE 11 - INFINITY ZOOM 12 - DOT GRID 13 - KALEIDOSCOPE 14 - MULTI MIRROR 15 - REBELLE DISTORT
FX2 Opacity	Effect 2 Opacity. Blends Effected media with original media	/LAYER 1/FX2 OPACITY/Value	0.0..1.0: FX2 Opacity 0-100%
FX2 Param 1	Effect 2 High resolution adjustment for parameter 1	/LAYER 1/FX2 PARAM 1/Value	0.0..1.0 FX2 Parameter 1
FX2 Param 2	Effect 2 High resolution adjustment for parameter 2	/LAYER 1/FX2 PARAM 2/Value	0.0..1.0 FX2 Parameter 2

FX2 Param 3	Effect 2 High resolution adjustment for parameter 3	/LAYER 1/FX2 PARAM 3/Value	0.0..1.0 FX2 Parameter 3
FX2 Param 4	Effect 2 High resolution adjustment for parameter 4	/LAYER 1/FX2 PARAM 4/Value	0.0..1.0 FX2 Parameter 4
FX2 Param 5	Effect 2 High resolution adjustment for parameter 5	/LAYER 1/FX2 PARAM 5/Value	0.0..1.0 FX2 Parameter 5
FX2 Param 6	Effect 2 High resolution adjustment for parameter 6	/LAYER 1/FX2 PARAM 6/Value	0.0..1.0 FX2 Parameter 6
FX2 Param 7	Effect 2 High resolution adjustment for parameter 7	/LAYER 1/FX2 PARAM 7/Value	0.0..1.0 FX2 Parameter 7
FX2 Param 8	Effect 2 High resolution adjustment for parameter 8	/LAYER 1/FX2 PARAM 8/Value	0.0..1.0 FX2 Parameter 8
FX2 Param 9	Effect 2 High resolution adjustment for parameter 9	/LAYER 1/FX2 PARAM 9/Value	0.0..1.0 FX2 Parameter 9
FX2 Param 10	Effect 2 High resolution adjustment for parameter 10	/LAYER 1/FX2 PARAM 10/Value	0.0..1.0 FX2 Parameter 10
FX2 Param 11	Effect 2 High resolution adjustment for parameter 11	/LAYER 1/FX2 PARAM 11/Value	0.0..1.0 FX2 Parameter 11
FX2 Param 12	Effect 2 High resolution	/LAYER 1/FX2 PARAM 12/Value	0.0..1.0 FX2 Parameter 12

	adjustment for parameter 12		
FX2 Param 13	Effect 2 High resolution adjustment for parameter 13	/LAYER 1/FX2 PARAM 13/Value	0.0..1.0 FX2 Parameter 13
FX2 Param 14	Effect 2 High resolution adjustment for parameter 14	/LAYER 1/FX2 PARAM 14/Value	0.0..1.0 FX2 Parameter 14
FX2 Param 15	Effect 2 High resolution adjustment for parameter 15	/LAYER 1/FX2 PARAM 15/Value	0.0..1.0 FX2 Parameter 15
FX2 Param 16	Effect 2 High resolution adjustment for parameter 16	/LAYER 1/FX2 PARAM 16/Value	0.0..1.0 FX2 Parameter 16
Transition Duration	Set the duration of the transition (cross fade on a layer)	/LAYER 1/TRANSITION DURATION/Value	0..65535 Milliseconds - 1 second = 1000

Transition Mode	Set the transition mode/blend mode for the transition (cross fade on a layer)	/LAYER 1/TRANSITION MODE/Value	0: ALPHA 1: ADDITIVE 2: MULTIPLY 3: DIFFERENCE 4: SCREEN 5: PRESERVE LUMA 6: RECTANGLE WIPE 7: TRIANGLE WIPE 8: MINIMUM 9: MAXIMUM 10: SUBTRACT 11: DARKEN 12: LIGHTEN 13: SOFT LIGHTEN 14: DARK LIGHTEN 15: EXCLUSION 16: RANDOM 17: RIPPLE 18: THRESHOLD 19: SINE 20: INVERT MASK 21: NOISE 22: SWIRL 23: GRADIENT 24: PIXEL SORT 25: CHECKERBOARD

			26: PULSE 27: HUE SHIFT 28: FRACTAL 29: WAVEFORM 30: RGB SPLIT 31: GLITCH
Volume	Audio volume (if video file has uncompressed embedded audio stream in 16, 24 or 32bit)	/LAYER 1/VOLUME/Value	0..65535: Volume 0 – 100%

DEVICE SETTINGS PATCH PATHS

All of the device settings are stored as JSON objects. The entire JSON objects can be read or written to. Alternatively any parameter of the JSON object can be written to individually using UpdatePatchJSON. Many of the JSON objects are quite large, so writing to parameters individually is recommended whenever possible.

* PLEASE BACKUP YOUR JSON FILES BEFORE EDITING THEM AS YOU CAN BREAK YOUR DEVICE BY SETTING INVALID VALUES

For example to switch on timecode triggering mode:

```
UpdatePatchJSON("/Timecode Cue List", [{"op":"replace","path":"/layers/0/useCueList","value":1}])
```

and to switch it off:

```
UpdatePatchJSON("/Timecode Cue List", [{"op":"replace","path":"/layers/0/useCueList","value":0}])
```

JSON OBJECT	**Description**		**PatchPath**
Media List	List of all Media files and Meta data on device		/Media List
System Settings	All of the devices System Settings		/System Settings
Output Mapping	Devices Video Output Mapping		/Output Mapping
Play List	Device Play List		/Play List
Timecode Cue List	External Clock Cue List		/Timecode Cue List
Vioso WB Settings	Warp & Blend Settings for Vioso		/Vioso WB Settings
Screenberry WB Settings	Warp & Blend Settings for Screenberry		/Screenberry WB Settings
Timeline	Timeline settings		/Timeline
LUT Colour Modes	All of the LUT Colour modes are accessed through this object		/LUT Colour Modes
NDI OUTPUT SETTINGS	All of the settings for the devices NDI output		/NDI OUTPUT SETTINGS
Sensors	Sensors can be configured to control any part of the system		/Sensors
Cloud	Users Hive cloud settings for this device		/Cloud
Schedule	The Schedule for this hive player		/Schedule
Dataflow	All of the dataflows stored on this device		/Dataflow
CMS	Configuration for the content management system		/CMS

PLAYLIST PATCH PATHS			
Function	**Description**	**PatchPath**	**Value range**
SEEK	Set play head position whilst paused or playing	/LAYER 1/Transport Control/Media Time/Value	0.0.. Duration of media in seconds
PLAYLIST PLAY SPECIFIED ROW	Play Row on Play list	/Playlist Control/Playlist Controller 1/Play List Next	1 = Row to play
PLAYLIST SEEK	Seek to specified time in play list	/Playlist Control/Playlist Controller 1/Play List Seek	s = time in floating point seconds to seek to

HEARTBEAT / UP TIME			
Function	**Description**	**PatchPath**	**Value range**
UP TIME	Get a floating point seconds value for how long the renderer has been running.	/UpTime/Up Time	0.0.. Duration of media in seconds

This is a non-exhaustive list of all of the possible patchPaths. If you wish to control something which is not shown in this document please don't hesitate to get in touch with the Hive dev team at support@hive.run

There are some further examples of how to use the Set/Get/WatchPatchDouble/String/JSON routes in a node JS environment at the end of this document in the section titled "Set/Get/WatchPatchDouble/String/JSON Examples"

File Uploads

GET/POST	Path	Parameters (origin → type)	Purpose / Result
POST	`/upload`	`myFiles` → form-data array (max 20) of files `broadcastToWorkers?` → body :number | boolean (O/req.) `lastFileIndex?` → body :number (O/req.) — next sequential index `replaceIndex?` → body :number (O/req.) — overwrite slot `replaceFilename?` → body :string (O/req.)	Main media ingest. Stores files in the current media folder, renames/indices them, updates thumbnails and `MediaCache.json`, and (optionally) broadcasts the file(s) to worker bees. Returns the uploaded-file array on success.
POST	`/uploadLiveControlFile`	`myFiles` → form-data array	Uploads control-png/SVG assets for Live-Control widgets into *Browser/Buzz/Source_Files/LiveControl/Controls/*.
POST	`/uploadLiveControlLayoutFile`	`myFiles` → form-data array	Saves full Live-Control layout JSON files into *Browser/Buzz/Source_Files/LiveControl/Layouts/*.
POST	`/uploadLUTFile`	`myFiles` → form-data array	Adds .cube / .3dl lookup-tables to *Distrib/modules/effects/LUTS/* and syncs them to workers.

POST	`/uploadScreenberryWBFile`	`myFiles` → form-data array`calibrationName` → body :string (folder name)	Places each calibration file into *Distrib/Screenberry/Calibrations/<calibrationName>/*.
POST	`/uploadUpdateFromFile`	`myFiles` → form-data array (expects hive_update.tar.gz)`preserveUserData?` → body :boolean (defaults true)	Drops the update package in */home/nvidia/Hive*, then executes hive_update.sh with `preserve` or `discard`; reboots on completion. Responds `{UpdateCompletedOK:true}` if the script launched.
POST	`/uploadViosoWBFile`	`myFiles` → form-data array	Adds .vwf warp-blend files to *Distrib/Vioso/Calibrations/*.

Media – file operations

GET/POST	Path	Parameters	Purpose / Response
POST	`/api/addSystemMedia`	varies by media type → body; always includes `replaceIndex:number`	Inserts *NDI*, *USB*, *WATCHFOLDER*, *APP*, *URL* or static system still; updates thumbnails and cache.
POST	`/api/addTimelineToMediaList`	timeline object → body	Saves timeline as new `.htl`; returns `{addedOK}`.
POST	`/api/clearMediaList`	–	Purges **Media/**, thumbnails, cache; seeds with `0000_BLACK.jpg`.
POST	`/api/deleteMediaFile`	`filename` → body :string	Removes media + thumbnail, updates cache.
POST	`/api/getAllMediaFilesWithFullPath`	–	Returns `filePaths:string[]` (recursive).
POST	`/api/getMediaCacheStatus`	–	`{mediaCacheReBuilding:boolean}`.
POST	`/api/getMediaFileMetaData`	`filename` → body :string	Returns `{metaData:{duration,fps}}`.

POST	`/api/getMediaFileSize`	`filename` → body :string	Returns `{size:number}` (bytes).
POST	`/api/getMediaFilenameAtIndex`	`index` → body :number	Looks up the filename that begins with that 4-digit index.
POST	`/api/getMediaFolder`	–	Returns `{mediaCache, mediaFolderDir}`; auto-rebuilds if needed.
POST	`/api/getMediaFolderNumFiles`	–	`{mediaFolderNumFiles, mediaFolderDir}`.
POST	`/api/getMediaThumbnailFilenames`	–	Array of `{filename}` for each cached item.
POST	`/api/getMediaPlaybackLog`	–	Returns `{mediaPlaybackLog}` (array).
POST	`/api/gettranscodeProgress`	`filename` → body :string	Returns `{transcodeProgress:number}` for that file.
POST	`/api/refreshMediaList`	–	Forces cache rebuild; returns `{listRefreshed:true}` when done.

POST	`/api/renameMediaFile`	`oldFilename` → body :string `newFilename` → body :string `postponeCacheUpdate?` → bool	Renames file + thumb, optionally defers cache rebuild.
POST	`/api/transcodeMediaFiles`	`files` → body :string `codec` → body :string	Kicks off background transcode; returns `{transcodeStartedOK}`.
POST	`/api/updatePlayListMetaData`	`thePlayList` → body :object	Fills in missing `duration/fps` on each playlist slot.
POST	`/api/updateTimelineInMediaList`	timeline object → body	Overwrites every existing `.htl` that matches `title`; returns `{updatedOK}`.

Media – folder management

GET/POST	Path	Parameters	Purpose
POST	`/api/getMediaFoldersList`	–	Alphabetised list of user sub-folders.
POST	`/api/mediaFoldersAddFolder`	`folderName` → body :string	Creates folder, seeds with **BLACK.jpg**.
POST	`/api/mediaFoldersDeleteFolder`	`folderName` → body :string	Removes folder + thumbs.
POST	`/api/mediaFoldersImportMedia`	`oldFolderName` `newFolderName` → string `mediaFilesToMove` → array `removeSourceFiles` → boolean	Copies / moves selected media.
POST	`/api/mediaFoldersRenameFolder`	`oldFolderName` `newFolderName` → string	Renames sub-folder.

Live-control / templates

GET/POST	Path	Parameters	Purpose
POST	`/api/deleteLiveControlLayout`	`filename` → body :string	Removes a saved layout locally & on workers.
POST	`/api/getLiveControlLayouts`	–	Returns array of layout definitions with `layoutName`.
POST	`/api/getLiveControlTypes`	–	Returns array of control-type JSON objects.
POST	`/api/getTemplateFilenames`	–	Lists `*.html` template names (no extension).
POST	`/api/saveLiveControlLayout`	`filename` → body :string	Saves current live-control state to that file.

LUTs & calibration

GET/POST	Path	Parameters	Purpose
POST	`/api/deleteLUT`	`filename` → body :string	Deletes LUT and propagates to workers.
POST	`/api/deleteScreenberryWBFile`	`filename` → body :string	Recursively deletes Screenberry calibration folder.
POST	`/api/deleteViosoWBFile`	`filename` → body :string `deleteOnWorkers` → bool	Removes Vioso warp-blend file.
POST	`/api/getScreenberryWBFileList`	–	Lists Screenberry calibration dirs.
POST	`/api/getViosoWBFileList`	–	Lists Vioso `*.vwf` files.

Cloud / broadcast / transfer

GET/POST	Path	Parameters	Description
POST	`/api/ResetBroadcastSend`	–	Cancels any in-flight broadcast transfer.
POST	`/api/getBroadcastFileProgress`	–	`{progress:number, filename}` (0-100).
POST	`/api/getBroadcastFileSendIPAddressList`	–	Array of workers currently in broadcast send.
POST	`/api/getHiveCloudFileTransferProgress`	–	Progress of upload *to* Hive Cloud.
POST	`/api/getHiveCloudPullBackFileUploadProgress`	–	Progress of pull-back *from* Cloud.
POST	`/api/ResetCloudFileTransferProgress`	–	Zeros cloud-transfer progress meters.
POST	`/api/pullBackFileUploadToHiveCloud`	`filename, server, username,`	Uploads local media to Hive

`serialNumber` → string	Cloud “pull-back” stash.

Clone / backup / restore

GET/POST	Path	Parameters	Purpose
POST	`/api/backupDevice`	`sourceIPAddress`, `targetLocation`, `username`, `password`	SCP-pushes config+media to external Linux box.
POST	`/api/clearDirectoriesBeforeCloning`	–	Empties media, thumbs, LUTs, calibrations before clone/restore.
POST	`/api/cloneDevice`	`sourceIPAddress`, `targetIPAddress`	Copies config+media across two Hive players.
POST	`/api/getCloneDeviceProgress`	–	`{progress, completedPercent}`.
POST	`/api/initCloneDeviceProgress`	–	Resets progress tracker; returns “READY”.
POST	`/api/restoreBackupToDevice`	`targetIPAddress`, `sourceLocation`, `username`, `password`	Pulls stored backup into current device.

System / network / display

GET/POST	Path	Parameters	Description
POST	`/api/getNetworkStatus`	–	`{DHCPActive, validIPAddress, ipAddress, netmask}`.
POST	`/api/setNetworkConfiguration`	network JSON	Applies static/DHCP setup via BeeMonitoring.
POST	`/api/getStorageSpace`	`storageLocation` → body :string (`"media_folder"` or other)	Returns free bytes.
POST	`/api/runFactoryReset`	`method="FACTORY_RESET_FROM_SCRIPT"` `preserveMediaFolder` ?:bool `preserveNetworkSettings` ?:bool	Performs factory reset and reboots.
POST	`/api/runSystemCommand`	`cmd` → body :string	Executes shell command on all bees.
POST	`/api/runSystemCommandLocally`	`cmd` → body :string	Executes command only on this device.

POST	`/api/setOSBackgroundImage`	`imageIndex` → body :number (0-2)	Switches Linux desktop wallpaper set.
POST	`/api/setSDIDisplayMode`	`displayMode` → body :`"hdmi"`|`"sdi"`	Toggles SDI vs HDMI output (reboot).
POST	`/api/setSDIRefreshRate`	`refreshRate` → body :string (“23.98”, “60.00”, …)	Sets `HIVE_AJA_RES`, then reboots.
POST	`/api/getSDIDisplayMode`	–	Returns `{SDIActive,resX,resY,rate}`.
POST	`/api/GetDisplayModes`	–	Returns compositor type + parsed display modes.
POST	`/api/GetXrandrDisplayModes`	–	Raw `xrandr` + parsed modes (Xorg only).
POST	`/api/testDataflowNode`	`schemaIndex` → body :number `nodeIndex` → body :number	Triggers a self-test of a data-flow node; returns `{testDataflowNodeOK}`.
POST	`/api/RequestParametersReSendFromQueen`	–	Asks the queen to rebroadcast its current parameter set.

Preview / thumbnails

GET/POST	Path	Parameters	Description
POST	`/api/getOutputFrame`	`includeImageData?` → bool `dualFramesRequest?` → bool	Returns low-res mapped/output JPEG path(s) and optional base-64.
POST	`/api/getThumbnailImageData`	`thumbPath` → body :string (relative)	Returns `{imgData}` base-64 for that thumbnail.

Diagnostics & logs

GET/POST	Path	Parameters	Purpose
POST	`/api/getBeeSyncLogSummary`	–	Aggregates last 3 log lines from BeeSync/PTP services.
POST	`/api/getDeviceLog`	–	Raw `dmesg` output.
POST	`/api/getFileDownloadProgress`	–	Global HTTP-download progress (0-100).
POST	`/api/getHiveLog`	–	Latest Hive application log.
POST	`/api/getNectarLog`	–	Returns accumulated console output captured by Nectar.
POST	`/api/getMediaCacheStatus`	–	Indicates if media-cache rebuild is running.

Security & user management

GET/POST	Path	Parameters	Description
POST	`/api/getSecurityCredentials`	–	Returns first stored username (if auth enabled).
POST	`/api/getSecurityStatus`	–	`{pwProtect:boolean}`.
POST	`/api/removeSecurity`	–	Deletes `.users.json`, disables login wall.

Misc / device utilities

GET/POST	Path	Parameters	Purpose
POST	`/api/appendObjectToJSONFIle`	`pathToJSONFile` → string `object` → any	Atomically appends to array-JSON file.
POST	`/api/assignNewDeviceStatus`	`newDevice:boolean`	Creates/removes `/home/nvidia/Hive/NEW_DEVICE`.
POST	`/api/getAudioDeviceList`	–	Lists ALSA audio devices (Linux).
POST	`/api/getDeviceSystemClock`	–	`{systemClockTime}` epoch ms.
POST	`/api/getNDISources`	–	Returns discovered NDI streams.
POST	`/api/getTileList`	–	Returns array of worker/queen “tiles” with model/resolution.
POST	`/api/testOff`	–	Clears “test” flag in BeeMonitoring.
POST	`/api/testOn`	–	Sets test flag.

POST	`/api/verifyNewDeviceStatus`	–	Checks for `NEW_DEVICE` marker file.

3. Authentication & security

- **/register** (POST) – add user (bcrypt password); only accessible when no `.users.json` exists.
- **/login** / **/logout** – standard form posts.
- Set remove security by calling **/api/removeSecurity**.
- The authentication middleware also honours `hiveCloudAuthToken` automatically for Hive Cloud pull-backs.

---

4. Common success envelopes

```
// progress-style
{
  "description": "hive cloud file upload progress",
  "progress": 37.5,
  "filename": "0020_demo.mov"
}

// generic ack
{ "cmdExecutedOK": true }
```

Errors are not standardised; many handlers reply `…OK:false` or custom strings. For automation check HTTP code **and** parse body.

5. Side-effects worth noting

Behaviour	Trigger routes
System reboot	`/api/setSDIRefreshRate`, `/api/setSDIDisplayMode`, `/api/runFactoryReset` (always), `/api/uploadEdidFile` (when `restart=true`), `/api/updateDevice`, Hive Cloud update workflow.
Disk-writes inside firmware partitions	Update, Factory-reset, EDID routes.
Process restarts / kills	`/api/runSystemCommand(_Locally)`, broadcast reset, transcode operations; Honey preview auto-spawn by `/api/getOutputFrame`.

Always ensure you POST **once** and wait for completion progress endpoints before repeating heavy operations (e.g. Broadcast, Clone/Backup).

---

6. Development & testing tips

- **CORS** is enabled globally (`app.use(cors())`), so browser tools can call the API directly from `localhost`.
- Set `--enable-iframe-embed` CLI flag to relax CSP & frame-ancestors.
- Media and thumb folders are resolved dynamically via `SystemSettings.hiveModel` and may live on `/SSD` for Player-2/3.
- The API surfaces an internal *BeeMonitoring* message bus via **/api/HTTPCommsMessage** – handy for remote patch control.
- For examples of this API in use, open the browser console and search for the end point in the javascript source code of the Hive user interface.

7. Uploading Video Files to your Hive Player

One of the most common tasks users of our web server API wish to tackle is uploading files to their Hive players from a custom UI. To accomplish this we need to use the /upload end point:

#	Route	Method(s)	Body / Query Parameters	Description
1	**/upload**	`POST`	**multipart/form-data** **Required** • `myFiles` – the file to upload (binary) **Optional** • `lastFileIndex` *(int)* – highest index which is ***already present*** on the device, or **-1** if none. • `replaceIndex` *(int)* – index of an existing slot you want to overwrite. • `replaceFilename` *(string)* – filename currently occupying that slot (helps workers delete the right file). • `broadcastToWorkers` *(0	1)* – if **1** and the device is a "queen", automatically propagate the file to all connected "worker" devices.

However.. Before we can call the /upload API there is some house keeping to do to ensure that the file your are about to upload will fit on the device and that you format the accompanying data correctly.

Typical flow

1. **Check free space** – call `/api/getStorageSpace` first.
2. **Build a `FormData` payload** (or multipart form) with the fields above.
3. **POST it to `http://<device-ip>/upload`**.
4. On success you'll receive a JSON object like the example below.

Minimal example

```
<!-- index.html -->
<input type="file" id="filePick" />
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script>
/**
 * Upload a single file to the specified device.
 * @param {File}   file        – The File object chosen by the user.
 * @param {String} deviceIp    – IP address (or hostname) of the target
device.
 * @param {Number} lastIndex   – Highest index currently stored on the
device, or -1 if none.
 */
async function uploadFile(file, deviceIp, lastIndex = -1) {
    try {
        /* ------------------------------------------------------------
         * 1)  CHECK FREE SPACE FIRST
         * ------------------------------------------------------------ */
        const storageResp = await $.ajax({
            url: `http://${deviceIp}/api/getStorageSpace`,
            type: "POST",
            contentType: "application/json",
            data: JSON.stringify({ storageLocation: "media_folder" })
        });

        if (!storageResp.storageSpaceCalculated || storageResp.storageSpace
< file.size) {
            console.warn("Insufficient storage space – upload aborted.");
            return;
        }

        /* ------------------------------------------------------------
         * 2)  BUILD THE multipart/form-data BODY
         * ------------------------------------------------------------ */
        const fd = new FormData();
        fd.append("myFiles", file);          // REQUIRED
        fd.append("lastFileIndex", lastIndex); // optional but useful

        /* ------------------------------------------------------------
         * 3)  SEND TO /upload  (progress via custom XHR)
         * ------------------------------------------------------------ */
        await $.ajax({
```

```
            url: `http://${deviceIp}/upload`,
            type: "POST",
            data: fd,
            processData: false,
            contentType: false,
            xhr: function () {
                const xhr = new XMLHttpRequest();
                xhr.upload.addEventListener("progress", (evt) => {
                    if (evt.lengthComputable) {
                        const pct = (evt.loaded / evt.total) * 100;
                        console.log(`Uploading ${file.name}:
${pct.toFixed(1)}%`);
                    }
                });
                return xhr;
            },
            success: (json) => {
                console.log("Upload complete:", json);
            },
            error: (jqXHR, textStatus, err) => {
                console.error("Upload failed:", textStatus, err);
            }
        });

    } catch (err) {
        console.error("Unexpected error:", err);
    }
}

/* ------------------------------------------------------------------
 * 4)  WIREFRAME UI — choose a file and go
 * ------------------------------------------------------------------ */
document.getElementById("filePick").addEventListener("change", function ()
{
    const chosen = this.files[0];
    if (chosen) {
        const deviceIp = "192.168.1.50"; // <-- replace with your target
        uploadFile(chosen, deviceIp);
    }
});
</script>
```

The server stores the file, example **0004_Sunset.jpg** and replies:

```
{
  "ok": true,
  "filename": "0004_Sunset.jpg",
  "fileIndex": 4,
  "storageUsed": 1073741824,
  "storageRemaining": 536870912
}
```

Curl Example

Replace an existing file in slot 2 and broadcast to workers

```
curl -X POST http://192.168.1.50/upload \
     -F "myFiles=@NewLogo.png" \
     -F "replaceIndex=2" \
     -F "replaceFilename=0002_OldLogo.png" \
     -F "broadcastToWorkers=1"
```

If the device is a queen, it will first delete `0002_OldLogo.png` on every worker, then push the new file to them. The response mirrors the previous example.

Error handling

HTTP code	Meaning	Common causes
`200 OK` (with `"ok": true`)	Success	–
`400 Bad Request`	Missing `myFiles` or malformed field	–
`507 Insufficient Storage`	Not enough free space	Call `/api/getStorageSpace` first
`409 Conflict`	`replaceIndex` out of range or filename clash	Verify indexes/filenames

Quick checklist for file upload implementation

1. **Check space** → `/api/getStorageSpace`.
2. **Create multipart form** with at least `myFiles`.
3. (Optional) set `lastFileIndex`, `replaceIndex`, `replaceFilename`, `broadcastToWorkers`.
4. **POST to `/upload`**.
5. **Parse JSON** response; update your media list accordingly.

Set/Get/WatchPatchDouble/String/JSON Examples

The following example code was created using node js v16.2 but should also be compatible with more recent versions. They designed as simple functional tests for each of the following end points:

SetPatchDouble
GetPatchDouble
WatchPatchDouble

SetPatchString
GetPatchString
WatchPatchString

SetPatchJSON
GetPatchJSON
WatchPatchJSON
UpdatePatchJSON

Get/SetPatchDouble:

```
// testPatchDouble.js
const http = require('http');

// simple helper to POST JSON and parse the JSON response
function post(route, payload) {
  return new Promise((resolve, reject) => {
    const data = JSON.stringify(payload);
    const options = {
      hostname: 'localhost',  // ← your host
      port: 80,               // ← your port
      path: route,
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Content-Length': Buffer.byteLength(data),
      },
    };

    const req = http.request(options, res => {
      let body = '';
      res.on('data', chunk => (body += chunk));
      res.on('end', () => {
        try {
          resolve(JSON.parse(body));
        } catch (err) {
          reject(new Error('Invalid JSON response: ' + body));
        }
      });
    });

    req.on('error', reject);
    req.write(data);
    req.end();
  });
}

(async () => {
  try {
    // 1) Fetch the current double
    const getRes = await post('/api/getPatchDouble', {
      patchPath: '/LAYER 1/FILE SELECT/Value'
    });
    if (!getRes.success) {
      throw new Error('getPatchDouble failed');
    }
```

```
    const current = getRes.value;
    if (typeof current !== 'number') {
      throw new Error('Expected a numeric value, got: ' + typeof
current);
    }
    console.log('Current value:', current);

    // 2) Compute a new value (e.g. +1)
    const updated = current + 1;
    console.log('Updating value to:', updated);

    // 3) Write it back
    const setRes = await post('/api/setPatchDouble', {
      patchPath: '/LAYER 1/FILE SELECT/Value',
      value: updated
    });

    console.log('setPatchDouble response:', setRes);
    if (!setRes.success) {
      throw new Error('setPatchDouble reported failure');
    }

    console.log('✅ Round-trip OK!');
  }

  catch (err) {
    console.error('❌ Test failed:', err.message);
    process.exit(1);
  }
})();
```

Get/SetPatchString

```
// testPatchString.js
const http = require('http');

// helper to POST JSON and parse the JSON response
function post(route, payload) {
  return new Promise((resolve, reject) => {
    const data = JSON.stringify(payload);
    const options = {
      hostname: 'localhost',  // ← adjust if needed
      port: 80,               // ← adjust if needed
      path: route,
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Content-Length': Buffer.byteLength(data),
      },
    };

    const req = http.request(options, res => {
      let body = '';
      res.on('data', chunk => (body += chunk));
      res.on('end', () => {
        try {
          resolve(JSON.parse(body));
        } catch (err) {
          reject(new Error('Invalid JSON response: ' + body));
        }
      });
    });

    req.on('error', reject);
    req.write(data);
    req.end();
  });
}

(async () => {
  try {
    // 1) Fetch the current string
    const getRes = await post('/api/getPatchString', {
      patchPath: '/Status/Text'
    });
    if (!getRes.success) {
      throw new Error('getPatchString failed');
```

```
    }

    const current = getRes.value;
    if (typeof current !== 'string') {
      throw new Error('Expected a string value, got: ' + typeof
current);
    }
    console.log('Current string:', current);

    // 2) Compute a new string (append marker)
    const updated = current + ' [updated]';
    console.log('Updating string to:', updated);

    // 3) Write it back
    const setRes = await post('/api/setPatchString', {
      patchPath: '/Status/Text',
      value: updated
    });

    console.log('setPatchString response:', setRes);
    if (!setRes.success) {
      throw new Error('setPatchString reported failure');
    }

    console.log('✅ Round-trip OK!');
  }
  catch (err) {
    console.error('❌ Test failed:', err.message);
    process.exit(1);
  }
})();
```

Get/SetPatchJSON

```
// testTimecodeCueList.js
const http = require('http');

function post(route, payload) {
  return new Promise((resolve, reject) => {
    const data = JSON.stringify(payload);
    const options = {
      hostname: 'localhost', // ← your host
      port: 80,              // ← your port
      path: route,
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Content-Length': Buffer.byteLength(data),
      },
    };

    const req = http.request(options, res => {
      let body = '';
      res.on('data', chunk => (body += chunk));
      res.on('end', () => {
        try {
          resolve(JSON.parse(body));
        } catch (err) {
          reject(new Error('Invalid JSON response: ' + body));
        }
      });
    });

    req.on('error', reject);
    req.write(data);
    req.end();
  });
}

(async () => {
  try {
    // 1) Fetch
    const getRes = await post('/api/getPatchJSON', { patchPath:
'/Timecode Cue List' });
    if (!getRes.success) throw new Error('getPatchJSON failed');

    const obj = getRes.json;
```

```
    // 2) Check hiveVersion & description
    if (obj.hiveVersion == null || obj.description == null) {
      throw new Error('Response missing hiveVersion or
description');
    }
    console.log('hiveVersion:', obj.hiveVersion);
    console.log('description:', obj.description);

    // 3) Toggle layers[0].useCueList (0 or 1)
    if (!Array.isArray(obj.layers) || typeof
obj.layers[0]?.useCueList !== 'number') {
      throw new Error('Invalid layers structure
(layers[0].useCueList not a number)');
    }
    obj.layers[0].useCueList = obj.layers[0].useCueList === 1 ? 0
: 1;
    console.log('Toggled layers[0].useCueList →',
obj.layers[0].useCueList);

    // 4) Write it back
    const setRes = await post('/api/setPatchJSON', {
      patchPath: '/Timecode Cue List',
      json: obj,
    });
    console.log('setPatchJSON response:', setRes);
  } catch (err) {
    console.error('❌ Test failed:', err.message);
    process.exit(1);
  }
})();
```

UpdatePatchJSON

```
// testUpdatePatchJSON.js
const http = require('http');

function post(route, payload) {
  return new Promise((resolve, reject) => {
    const data = JSON.stringify(payload);
    const options = {
      hostname: 'localhost',  // ← change if your API lives
elsewhere
      port: 80,               // ← change to your server's port
      path: route,
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Content-Length': Buffer.byteLength(data),
      },
    };

    const req = http.request(options, res => {
      let body = '';
      res.on('data', chunk => body += chunk);
      res.on('end', () => {
        try {
          resolve(JSON.parse(body));
        } catch (err) {
          reject(new Error('Invalid JSON response: ' + body));
        }
      });
    });

    req.on('error', reject);
    req.write(data);
    req.end();
  });
}

(async () => {
  try {
    // 1) Get the patch JSON
    const getRes = await post('/api/getPatchJSON', {
      patchPath: '/Timecode Cue List'
    });
    if (!getRes.success) {
      throw new Error('getPatchJSON failed');
```

```
    }
    const obj = getRes.json;

    // 2) Validate & toggle useCueList
    if (!Array.isArray(obj.layers) || typeof
obj.layers[0]?.useCueList !== 'number') {
      throw new Error('Invalid layers structure or useCueList not
a number');
    }
    const oldVal = obj.layers[0].useCueList;
    const newVal = oldVal === 1 ? 0 : 1;
    console.log(`Current useCueList: ${oldVal}, toggling to:
${newVal}`);

    // 3) Prepare JSON-Patch operation
    const patchOps = [
      { op: 'replace', path: '/layers/0/useCueList', value: newVal
}
    ];

    // 4) Send updatePatchJSON
    const setRes = await post('/api/updatePatchJSON', {
      patchPath: '/Timecode Cue List',
      patch: patchOps
    });

    console.log('updatePatchJSON response:', setRes);
    if (!setRes.success) {
      throw new Error('updatePatchJSON reported failure');
    }

    console.log('✅ Toggle operation completed successfully!');
  }
  catch (err) {
    console.error('❌ Test failed:', err.message);
    process.exit(1);
  }
})();
```

WatchPatchDouble

```
// testWatchPatchDouble.js
const http = require('http');

const patchPath = '/LAYER 1/FILE SELECT/Value';
const url =
`http://localhost:80/api/watchPatchDouble?patchPath=${encodeURICom
ponent(patchPath)}`;

// Open the SSE stream
const req = http.get(url, { headers: { Accept: 'text/event-stream'
} }, res => {
  console.log(`Subscribed to changes on "${patchPath}"`);
  res.setEncoding('utf8');

  let buffer = '';
  res.on('data', chunk => {
    buffer += chunk;
    // SSE events are separated by double newlines
    const events = buffer.split('\n\n');
    buffer = events.pop(); // leftover

    for (const event of events) {
      // Each event may have multiple lines (we only care about
"data: ...")
      for (const line of event.split('\n')) {
        if (line.startsWith('data: ')) {
          const payload = line.slice(6);
          try {
            const { value } = JSON.parse(payload);
            console.log(
              `[${new Date().toISOString()}] Path: ${patchPath} →
New value:`,
              value
            );
          } catch (e) {
            console.error('Malformed JSON in SSE data:', payload);
          }
        }
      }
    }
  });

  res.on('end', () => {
    console.log('SSE connection closed by server');
```

```
  });
});

req.on('error', err => {
  console.error('Connection error:', err);
});

process.on('SIGINT', () => {
  console.log('Closing subscription');
  req.abort();
  process.exit();
});
```

WatchPatchString

```
// testWatchPatchString.js
const http = require('http');

const patchPath = '/Status/Text';  // ← adjust to whatever path
you're watching
const url =
`http://localhost:80/api/watchPatchString?patchPath=${encodeURICom
ponent(patchPath)}`;

// Open the SSE connection
const req = http.get(url, { headers: { Accept: 'text/event-stream'
} }, res => {
  console.log(`Subscribed to changes on "${patchPath}"`);
  res.setEncoding('utf8');

  let buffer = '';
  res.on('data', chunk => {
    buffer += chunk;
    // split on SSE event delimiter
    const events = buffer.split('\n\n');
    buffer = events.pop(); // leftover fragment

    for (const event of events) {
      for (const line of event.split('\n')) {
        if (line.startsWith('data: ')) {
          const payload = line.slice(6);
          try {
            const { value } = JSON.parse(payload);
            console.log(
              `[${new Date().toISOString()}] Path: ${patchPath} →
New value: "${value}"`
            );
          } catch (err) {
            console.error('Malformed SSE data:', payload);
          }
        }
      }
    }
  });

  res.on('end', () => {
    console.log('SSE connection closed by server');
  });
});
```

```
req.on('error', err => {
  console.error('Connection error:', err);
});

process.on('SIGINT', () => {
  console.log('Closing subscription');
  req.abort();
  process.exit();
});
```


```
// testWatchPatchJSON.js
const http = require('http');

const patchPath = '/Timecode Cue List';
const url =
`http://localhost:80/api/watchPatchJSON?patchPath=${encodeURICompo
nent(patchPath)}`;

// Utility: recursively diff two values, returning an array of
change descriptions.
function diffValues(oldVal, newVal, path = '') {
  const changes = [];
  if (oldVal === newVal) {
    return changes;
  }
  const oldType = Object.prototype.toString.call(oldVal);
  const newType = Object.prototype.toString.call(newVal);

  if (oldType !== newType) {
    changes.push(`${path}: type changed from ${oldType} to
${newType}`);
    return changes;
  }

  // Handle objects (including arrays)
  if (oldType === '[object Object]' || oldType === '[object
Array]') {
    const keys = new Set([
      ...Object.keys(oldVal || {}),
      ...Object.keys(newVal || {})
    ]);
    for (const key of keys) {
      const subPath = path ? `${path}.${key}` : key;
      changes.push(
        ...diffValues(
          oldVal ? oldVal[key] : undefined,
          newVal ? newVal[key] : undefined,
          subPath
        )
      );
    }
  } else {
    // Primitive changed
```

```
    changes.push(`${path}: ${JSON.stringify(oldVal)} →
${JSON.stringify(newVal)}`);
  }
  return changes;
}

let lastJson = null;

// Open SSE connection
const req = http.get(
  url,
  { headers: { Accept: 'text/event-stream' } },
  res => {
    console.log(`Subscribed to JSON changes on "${patchPath}"`);
    res.setEncoding('utf8');

    let buffer = '';
    res.on('data', chunk => {
      buffer += chunk;
      const parts = buffer.split('\n\n');
      buffer = parts.pop(); // leftover

      for (const part of parts) {
        for (const line of part.split('\n')) {
          if (line.startsWith('data: ')) {
            const payload = line.slice(6);
            let parsed;
            try {
              parsed = JSON.parse(payload);
            } catch (e) {
              console.error('Invalid JSON payload:', payload);
              continue;
            }
            const json = parsed.json;
            if (lastJson === null) {
              console.log('Initial JSON:\n', JSON.stringify(json,
null, 2));
            } else {
              const changes = diffValues(lastJson, json);
              if (changes.length) {
                console.log('Changed parameters:');
                changes.forEach(c => console.log('  ' + c));
              } else {
                console.log('No changes detected');
              }
            }
```

```
            lastJson = json;
          }
        }
      }
    });

    res.on('end', () => {
      console.log('SSE connection closed by server');
    });
  }
);

req.on('error', err => {
  console.error('Connection error:', err);
});

process.on('SIGINT', () => {
  console.log('Closing subscription');
  req.abort();
  process.exit();
});
```