# HIVE - Nectar API Route Catalogue

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:

- **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** maintaining the browser-based UI that already consumes these routes.
- **Ops teams** who occasionally 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 |

---

**Conventions used in the route table**

- **Route** – the exact string as registered with node js (case-sensitive).
- **Method(s)** – only **GET** and **POST** are present in this code-base.
- **Body / Query parameters** –
    - Required keys are **bold** (`name` • `value`).
    - Optional keys are *italicised* or `*(bool)*` for flags.
    - For file uploads, the table lists the **multipart field names** and accepted extensions.
- When a handler relies solely on `req.session`, URL params, or disk inspection, the parameter column shows **"—"**.

---

**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.

---

# Reading the table

| Column | Meaning |
|---|---|
| **Route** | The public URL path, listed alphabetically. Variables appear in `{braces}`. |
| **Method(s)** | Supported HTTP verb(s). |
| **Body / Query parameters** | JSON-body or query-string keys the handler reads, their type, whether *required*, and extra notes. |
| **Returns** | Shape of the JSON reply (or other payload) on success. Fields in **bold** are always present. |
| **Description** | Functional summary plus important side-effects (disk writes, process restarts, etc.). |

Unless noted otherwise every reply is `application/json; charset=utf-8` and **status 200** on success, non-200 on error (handlers typically send 400 or 500).

All timestamps are UNIX ms, sizes are bytes.

Session-based endpoints (/login, /logout, etc.) rely on the `express-session` cookie; every other route sits behind the `checkAuthentication` middleware when `pwProtect==true`.

For unattended or local scripts you may disable security via **/api/removeSecurity**.

# 1. Alphabetical route list

| # | Route | Method(s) | Body / Query Parameters (required • optional → description) |
|---|---|---|---|
| 1 | **/** | GET | — – Root request: redirects to `/index.html` or `/login.html` depending on session & password-protection state. |
| 2 | **/login** | POST | username • password → attempts to authenticate a user, writes session cookie. |
| 3 | **/logout** | GET | — – Destroys session and redirects. |
| 4 | **/protected** | GET | — – Example protected route (only served if logged-in). |
| 5 | **/register** | POST | username • password → creates a new local account (bcrypt-hashed, saved to `.users.json`). |
| 6 | **/upload** | POST | multipart-form-data field `myFiles[]` (up to 20) + optional fields:• **broadcastToWorkers** *bool/int* – propagate to workers from a Queen.• **lastFileIndex** *int* – last numeric index already used.• **replaceIndex** *int* – overwrite an existing media file slot.• **replaceFilename** *string* – name of the file being replaced. |
| 7 | **/uploadEdidFile** | POST | multipart field **edidFile** (binary *.bin*) • output *“hdmi”* \| *“golden-finger”* • restart *(bool)* – whether to reboot after flashing. |
| 8 | **/uploadLiveControlFile** | POST | multipart **myFiles[]** – JSON definitions for custom Live-Control widgets. |

| | | | |
|---|---|---|---|
| 9 | **/uploadLiveControlLayoutFile** | POST | multipart **myFiles[]** – JSON page-layout definitions. |
| 10 | **/uploadLUTFile** | POST | multipart **myFiles[]** – *.cube/.3dl* lookup-tables placed under *Distrib/modules/effects/LUTS*. |
| 11 | **/uploadScreenberryWBFile** | POST | multipart **myFiles[]** • calibrationName *(folder)* – archives Screenberry warp/blend sets under the given sub-folder. |
| 12 | **/uploadUpdateFromFile** | POST | multipart **myFiles[]** (one *.tar.gz*) • preserveUserData *(bool)* – whether the updater should keep user assets. |
| 13 | **/uploadViosoWBFile** | POST | multipart **myFiles[]** – *.vwf* calibration files for VIOSO warp/blend. |
| 14 | **/api/ResetBroadcastSend** | POST | — – Clears Bee-to-Bee broadcast file-transfer state. |
| 15 | **/api/ResetCloudFileTransferProgress** | POST | — – Zeroes the cloud-transfer progress counters. |
| 16 | **/api/StartBeeInteractiveImageDispatchThread** | POST | — – Spawns a 2 s polling loop that uploads visitor photos to BBC Earth Experience API. |
| 17 | **/api/StoreLocalVariable** | POST | name • value – persists small JSON blobs under */LocalVariables/*. |
| 18 | **/api/appendObjectToJSONFIle** | POST | pathToJSONFile • object – appends *object* to the array stored in *pathToJSONFile*. |
| 19 | **/api/assignNewDeviceStatus** | POST | newDevice *(bool)* – creates/removes `/home/nvidia/Hive/NEW_DEVICE` marker file. |
| 20 | **/api/backupDevice** | POST | sourceIPAddress • targetLocation (Linux SCP URI `IP:/path`) • username • password – pushes full |

| | | | |
|---|---|---|---|
| | | | configuration & media to a NAS/remote host for backup. |
| 21 | **/api/clearDirectoriesBeforeCloning** | POST | — – Purges media/LUT/thumb/live-control dirs in preparation for cloning. |
| 22 | **/api/clearMediaList** | POST | — – Deletes every file inside the active media folder (adds 0000_BLACK.jpg afterwards). |
| 23 | **/api/cloneDevice** | POST | sourceIPAddress • targetIPAddress – mirrors config/media from *source* onto *target* over SCP. |
| 24 | **/api/createCompositionTriggerFile** | POST | compositionName • (trigger fields…) – writes *.hcf* composition preset. |
| 25 | **/api/deleteLiveControlLayout** | POST | filename – removes a stored live-control page JSON. |
| 26 | **/api/deleteLUT** | POST | filename • deleteOnWorkers *(bool)* – removes a LUT and optionally propagates. |
| 27 | **/api/deleteMediaFile** | POST | filename • suppressCacheRefresh *(bool)* – deletes a media asset and thumbnail. |
| 28 | **/api/deleteScreenberryWBFile** | POST | filename – deletes a Screenberry calibration folder. |
| 29 | **/api/deleteViosoWBFile** | POST | filename • deleteOnWorkers *(bool)* – removes a *.vwf* file everywhere. |
| 30 | **/api/getAllMediaFilesWithFullPath** | POST | — – Returns `filePaths[]` (recursive list under current mediaDir). |
| 31 | **/api/getAudioDeviceList** | POST | — – Reads **AudioDeviceList.txt** (Linux) and lists ALSA/Pulse devices. |
| 32 | **/api/getBackupRegionJSON** | POST | backupIndex – fetches stored multi-screen backup-region geometry. |

| | | | |
|---|---|---|---|
| 33 | **/api/getBeeSyncLogSummary** | POST | — – Last 3 log lines of BeeSync / ptp4l / phc2sys units (Intel SDM only). |
| 34 | **/api/getBroadcastFileProgress** | POST | — – % complete for Queen->Worker file multicast. |
| 35 | **/api/getBroadcastFileSendIPAddressList** | POST | — – List of worker IPs currently targeted by a Queen broadcast. |
| 36 | **/api/getCloneDeviceProgress** | POST | — – Returns textual status & percentage while clone/backup is active. |
| 37 | **/api/getDeviceLog** | POST | — – On Linux runs `dmesg`; on Windows returns “Not supported”. |
| 38 | **/api/getDeviceSystemClock** | POST | — – Milliseconds since epoch on this device. |
| 39 | **/api/getDisplayEdid** | POST | output *(“hdmi” |
| 40 | **/api/getDisplayModes** | POST | — – Unified call: returns X-RandR modes (Xorg) **or** DRM/KMS modes. |
| 41 | **/api/getEdidProperties** | POST | — – Friendly name, forced-EDID flag and HPD status for HDMI & golden-finger. |
| 42 | **/api/getFileDownloadProgress** | POST | — – Server-side % of the current *downloadWithProgress* transfer. |
| 43 | **/api/getHiveCloudFileTransferProgress** | POST | — – % of the most-recent cloud file delivery to this player. |
| 44 | **/api/getHiveCloudPullBackFileUploadProgress** | POST | — – % while *pullBackFileUploadToHiveCloud* is uploading to Hive Cloud. |
| 45 | **/api/getHiveCloudUploadedFile** | POST | username • device • filenameToTransfer • server • broadcastToWorkers • lastFileIndex • replaceIndex • replaceFilename – downloads a file from Hive Cloud then ingests it locally. |
| 46 | **/api/getHiveCloudUpdateFileTransferProgress** | POST | — – % while downloading a tar-update from Hive Cloud. |

| | | | |
|---|---|---|---|
| 47 | **/api/getHiveLog** | POST | — – On Linux streams `journalctl -u hive`; on Windows opens the freshest log under *Distrib/SDebug/*. |
| 48 | **/api/getLiveControlLayouts** | POST | — – Array of saved live-control page JSONs. |
| 49 | **/api/getLiveControlPageNames** | POST | — – Human-readable names for the 16 Live-Control slots. |
| 50 | **/api/getLiveControlTypes** | POST | — – Control-type taxonomy (button, slider, etc.) from *Source_Files/LiveControl/Controls/*. |
| 51 | **/api/getMacAddress** | POST | — – Detects first active NIC (eth0/enpXs0/…) and returns its MAC. |
| 52 | **/api/getMediaCacheStatus** | POST | — – Boolean flag indicating if a cache rebuild is in progress. |
| 53 | **/api/getMediaFileMetaData** | POST | filename – Returns duration & fps (via FFprobe) for videos/timelines. |
| 54 | **/api/getMediaFilenameAtIndex** | POST | index – File-name whose numeric prefix equals *index* (0000_*). |
| 55 | **/api/getMediaFileSize** | POST | filename – Size in bytes of a media asset. |
| 56 | **/api/getMediaFolder** | POST | — – Returns mediaCache JSON for the folder currently set in the patch. |
| 57 | **/api/getMediaFolderNumFiles** | POST | — – Counts playable files (excluding .json/thumbs) in selected folder. |
| 58 | **/api/getMediaFoldersList** | POST | — – Alphabetical list of sub-folders beneath *mediaDir* (excluding “Media”). |
| 59 | **/api/getMediaPlaybackLog** | POST | — – Reads *Browser/Buzz/Source_Files/mediaLog.json*. |
| 60 | **/api/getMediaThumbnailFilenames** | POST | — – List of `*.jpg` thumbs in current folder (one per media). |

| | | | |
|---|---|---|---|
| 61 | **/api/getNetworkStatus** | POST | — – DHCP presence, first IPv4, netmask – via *os.networkInterfaces()*. |
| 62 | **/api/getNDISources** | POST | — – Reads */Browser/Buzz/Source_Files/NDI_Sources.json*. |
| 63 | **/api/getOutputFrame** | POST | includeImageData *(bool)* • dualFramesRequest *(bool)* – Returns paths (and optionally Base64) to the latest Honey low-res output + mapped preview. |
| 64 | **/api/getSDIDisplayMode** | POST | — – Indicates if SDI is active and the fixed UHD/8K mode in use. |
| 65 | **/api/getSecurityCredentials** | POST | — – Returns first stored username if password-protection is enabled. |
| 66 | **/api/getSecurityStatus** | POST | — – pwProtect `true/false`. |
| 67 | **/api/getStorageSpace** | POST | storageLocation *(“media_folder” |
| 68 | **/api/getTemplateFilenames** | POST | — – HTML template names for advanced mapping. |
| 69 | **/api/getTileList** | POST | — – Returns logical tiles (workers) with resolution and role info. |
| 70 | **/api/getViosoWBFileList** | POST | — – List of *.vwf* warp/blend files. |
| 71 | **/api/getXrandrDisplayModes** | POST | — – Raw parsed `xrandr` modes plus ASCII dump. |
| 72 | **/api/gettranscodeProgress** | POST | filename – Progress (0-100) for the named file’s current Transcoder job. |
| 73 | **/api/getMediaCacheStatus** | POST | — – Boolean: is rebuild in progress. |
| 74 | **/api/HTTPCommsMessage** | POST | msg – Arbitrary command string routed to Bee Monitoring. |
| 75 | **/api/initCloneDeviceProgress** | POST | — – Resets clone/backup progress counters. |

| | | | |
|---|---|---|---|
| 76 | **/api/login** *(same as `/login`)* | POST | username • password – alias path under `/api` isn't present; only `/login` exists. |
| 77 | **/api/mediaFoldersAddFolder** | POST | folderName – Creates a sanitized sub-folder and seeds it with *0000_BLACK.jpg*. |
| 78 | **/api/mediaFoldersDeleteFolder** | POST | folderName – Deletes folder and matching thumbs. |
| 79 | **/api/mediaFoldersImportMedia** | POST | newFolderName • oldFolderName • mediaFilesToMove[] • removeSourceFiles *(bool)*. |
| 80 | **/api/mediaFoldersRenameFolder** | POST | oldFolderName • newFolderName. |
| 81 | **/api/pullBackFileUploadToHiveCloud** | POST | filename • server URL • username • serialNumber – Push local file **up** to Hive Cloud. |
| 82 | **/api/readCompositionFile** | POST | compositionName – Returns parsed *.hcf*. |
| 83 | **/api/refreshMediaList** | POST | — – Forces a full FFprobe-based cache rebuild. |
| 84 | **/api/removeSecurity** | POST | — – Deletes *.users.json*, disables password gate. |
| 85 | **/api/renameMediaFile** | POST | oldFilename • newFilename • postponeCacheUpdate *(bool)*. |
| 86 | **/api/RequestParametersReSendFromQueen** | POST | — – Asks Queen to resend system patch over Bee link. |
| 87 | **/api/resetMediaList** *(alias of clearMediaList)* | POST | — |
| 88 | **/api/restoreBackupToDevice** | POST | targetIPAddress • sourceLocation • username • password – pulls backup back onto player. |
| 89 | **/api/runFactoryReset** | POST | method = `"FACTORY_RESET_FROM_SCRIPT"` • preserveNetworkSettings *(bool)* • preserveMediaFolder *(bool)*. |

| | | | |
|---|---|---|---|
| 90 | **/api/runScheduleEventNow** | POST | event – executes a Timeline/Scheduler entry immediately. |
| 91 | **/api/runSystemCommand** | POST | method ="Nectar_run_command" • cmd – executes shell on **all** workers. |
| 92 | **/api/runSystemCommandLocally** | POST | method ="Nectar_run_command" • cmd – executes shell **only on this node**. |
| 93 | **/api/saveLiveControlLayout** | POST | filename – Saves current page (from *Distrib/System/liveControlLayout.json*) under the given name. |
| 94 | **/api/setForcingEdid** | POST | output *(hdmi |
| 95 | **/api/setNetworkConfiguration** | POST | ipAddress • netMask • gateway • dns etc. – passed through to beeMonitoring.SetNetworkConfiguration(). |
| 96 | **/api/setOSBackgroundImage** | POST | imageIndex *(0 Hive |
| 97 | **/api/setSDIDisplayMode** | POST | displayMode *(“hdmi” |
| 98 | **/api/setSDIRefreshRate** | POST | refreshRate (“23.98”…”60.00”) – updates HIVE_AJA_RES env and reboots. |
| 99 | **/api/storeImageAtLocation** | POST | filenameIncPath • imageData (Base64) – saves/overwrites arbitrary image on disk. |
| 100 | **/api/switchGrassValleyMatrix** | POST | backups[] (0 primary/1 backup) • devices[] (matrix destination indices) – crafts UDP TAKE-packet to GVG Mars 5184. |
| 101 | **/api/testDataflowNode** | POST | schemaIndex • nodeIndex – instructs SV patch to run self-test on a node. |

| | | | |
|---|---|---|---|
| 102 | **/api/testOff** | POST | — – Clears test mode on OLED front-panel (BeeMonitoring). |
| 103 | **/api/testOn** | POST | — – Enables perpetual test mode. |
| 104 | **/api/transcodeMediaFiles** | POST | files[] • codec *(“h264” |
| 105 | **/api/updateDevice** | POST | preserveUserData *(bool)* – downloads updater script + tarball from hive.run then executes. |
| 106 | **/api/updateHiveTimeline** | POST | timelineFilename (*.htl*) – Copies it to *Distrib/Timeline/Timeline.json*. |
| 107 | **/api/updatePlayListMetaData** | POST | thePlayList – fills missing *duration/fps* for every entry via FFprobe. |
| 108 | **/api/updateTimelineInMediaList** | POST | description = “hive buzz timeline” • title + timeline body – overwrites all matching *.htl* media files. |
| 109 | **/patch/*** | GET (static) | — – Serves files from *../SVBrowserUI*. |
| 110 | **/api/registerBee** | GET | — – Placeholder “hello world” test route. |

## 2. Upload endpoints

| Route | Verb | Field name(s) | Notes |
|---|---|---|---|
| `/upload` | POST | **myFiles** (max 20) | Multipart; stores to *current* media folder, then renames/indexes, thumb-gens, optional broadcast. |
| `/uploadLiveControlFile` | POST | **myFiles** | Saves to `Browser/Buzz/Source_Files/LiveControl/Controls/`. |
| `/uploadLiveControlLayoutFile` | POST | **myFiles** | Saves to `…/LiveControl/Layouts/`. |
| `/uploadLUTFile` | POST | **myFiles** | Saves to `/Distrib/modules/effects/LUTS/`. |
| `/uploadViosoWBFile` | POST | **myFiles** | Saves to `/Distrib/Vioso/Calibrations/`. |
| `/uploadScreenberryWBFile` | POST | **myFiles** + `calibrationName` | Creates per-calibration sub-folder. |
| `/uploadEdidFile` | POST | **edidFile** + `output` + `restart` | Stores EDID binary into `/Distrib/EDIDs`, optionally reboots. |
| `/uploadUpdateFromFile` | POST | **myFiles** + `preserveUserData` | Place `hive_update.tar.gz` & run updater on device. |

All uploads expect `multipart/form-data`.

## 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**<br>**Required**<br>• `myFiles` – the file to upload (binary)<br><br>**Optional**<br>• `lastFileIndex` *(int)* – highest index already present on the device, or **-1** if none.<br>• `replaceIndex` *(int)* – index of an existing slot you want to overwrite.<br>• `replaceFilename` *(string)* – filename currently occupying that slot (helps workers delete the right file).<br>• `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.