AD-DA.ca, Webhook

Plugin Description

The AD-DA.ca Webhook plugin embeds a lightweight HTTP server directly inside a Q-SYS Core and exposes user-defined webhook endpoints. When an external system sends an HTTP request to a configured Path, the plugin flashes an LED, captures the request payload (JSON body and/or query string), and surfaces matching key/value pairs as Q-SYS controls so that other plugins, scripts, or design logic can react to the incoming data.

The plugin is built on a custom Express-like HTTP server module with middleware support, route parameters, JSON body parsing, Bearer-token authentication, and Host-header interface filtering. Parameters from the query string and the JSON body are merged automatically (body values take precedence on collision), allowing the same endpoint to be triggered from either `GET` or `POST` requests.

In addition to user-defined webhooks, the plugin exposes a set of built-in endpoints to read the design status and inventory, and to remotely drive Q-SYS Named Components and Named Controls. This makes the plugin a single entry point for integrating Q-SYS with external automation systems, control panels, mobile apps, and cloud services.

Key features

- **Passive webhook listening** - The plugin exposes configurable HTTP endpoints that passively listen for incoming `GET` or `POST` requests
- **Immediate trigger execution** - Upon receiving a webhook, the plugin instantly updates control pins based on the value and status extracted from the payload, enabling real-time control of connected applications or devices
- **Flexible data input** - Supports data reception via query parameters (HTTP `GET`) or through JSON payloads (HTTP `POST`) with `Content-Type: application/json`
- **Trigger support without parameters** - Can respond to simple webhook triggers even when no parameters are provided with the `GET` request or no data payload is received with the `POST` request, allowing the path to act as a pure trigger
- **Multi-path listening** - Up to 6 independent webhook paths, each with configurable key/value extraction
- **Extended key monitoring** - Up to 32 key-value pairs can be monitored per webhook path
- **Hold or Pulse Mode** - Per-webhook toggle to either hold the last received value or clear it after each request
- **Bearer Token Authentication** - Optional `Authorization: Bearer <token>` enforcement on all routes

- **Interface Filtering** - Bind the server to a specific network interface or accept requests on any interface
- **Built-in Status and Inventory Endpoints** - `/status` and `/inventory` return `Design.GetStatus()` and `Design.GetInventory()` as JSON
- **Named Component Control Endpoint** - `/namedcomponent` drives any Named Component control (`Trigger`, `SetString`, `SetPosition`, `SetValue`, `SetBoolean`, `Toggle`) with optional ramp time
- **Named Controls Endpoint** - `/namedcontrols` drives Named Controls using the same action set
- **Automatic Path Sanitization** - Paths are normalized (allowed characters, collapsed slashes, leading slash forced)
- **Case-insensitive Routes** - Endpoints match regardless of URL casing

Use cases

- Integrate with a room booking system to automatically power on AV equipment via webhook
- React to smart-home or building-automation events (e.g. sensors, wireless buttons) to control Q-SYS audio
- Connect Q-SYS to cloud automation platforms such as Zapier, IFTTT, Home Assistant, Unifi, etc.
- Trigger scenes or presets remotely via webhook calls from third-party systems
- Integrating Q-SYS with external automation, BMS, or AV control systems
- Exposing read-only design status and inventory to monitoring tools
- Remotely driving Named Components and Named Controls without writing custom Lua

Configuration Overview

Plugin version

This document is written for the plugin :

- AD-DA.ca Webhook V1.2.0.0

Release note

[2026-06] : V1.2.0.0

- Migrated to Express-like HTTP server module with middleware support
- Added built-in `/status` and `/inventory` endpoints returning `Design.GetStatus()` and `Design.GetInventory()` as JSON
- Added Body and Code Response Controls
- Added Path sanitization and case-insensitive route matching

- New UI

Fixes

- Memory leak fixed under heavy load

[2025-07] : V1.1.0.0

- Added `Data Pair Count` property
- Added `Complete URL Path` controls
- Initial release to Asset Manager

[2025-04] : V1.0.0.0

- Initial release

QDS compatibility

This plugin has been developed and tested on :

- Q-SYS Designer versions `9.13.0` thru `10.3.0`
- Q-SYS Designer version `10.0.X` - known Emulation Mode issue with network interface address selection (works normally in Run Mode on a physical Core)

Properties

License

Type: `String`
Default value: `Empty`
Description: License key activated for the main core.

Licenses are issued on a "per core" basis, as in each core needs its own license to run as many instances of the plugin as needed on this particular core and licenses are not transferable from core to core.

License keys can be added while in Offline mode, in the Properties pane on the right. To purchase licenses, go to https://ad-da.ca.

The plugin can be emulated for free, allowing you to prepare everything without requiring you to have access to a core.

License Backup

Type: `String`
Default value: `Empty`
Description: License key activated for the backup core.

If your main Q-SYS Core fails, the backup Core automatically takes over without needing a license to function as a temporary replacement. However, if the design boots from the backup Core, a valid license is required. Additionally, if the backup Core is restarted (due to power loss or other reasons), the plugin instances cannot initialize without a valid license.

Webhook Count

Type: `integer`
Min: `1`
Max: `6`
Default Value: `1`
Description: Number of independent webhook endpoints exposed by the plugin. Each webhook generates its own page with Path, URL, Hold, LED, Data, and key/value controls.

Data Pair Count

Type: `integer`
Min: `4`
Max: `32`
Default Value: `4`
Description: Number of key/value rows generated per webhook. Each row extracts one named parameter from the incoming request and exposes its value as a control.

Dev

Type: `string`
Default Value: `""`
Description: Developer/debug field reserved for internal use.

Debug Print

Type: `enum`
Choices: `None`, `Tx/Rx`, `Tx`, `Rx`, `Function Calls`, `All`, `Debug`
Default Value: `None`
Description: Controls verbosity of debug messages printed to the Q-SYS Designer Console. Use `Tx/Rx` to see HTTP traffic, `Function Calls` to trace plugin execution, or `All`/`Debug` for full diagnostics.

Controls

Status

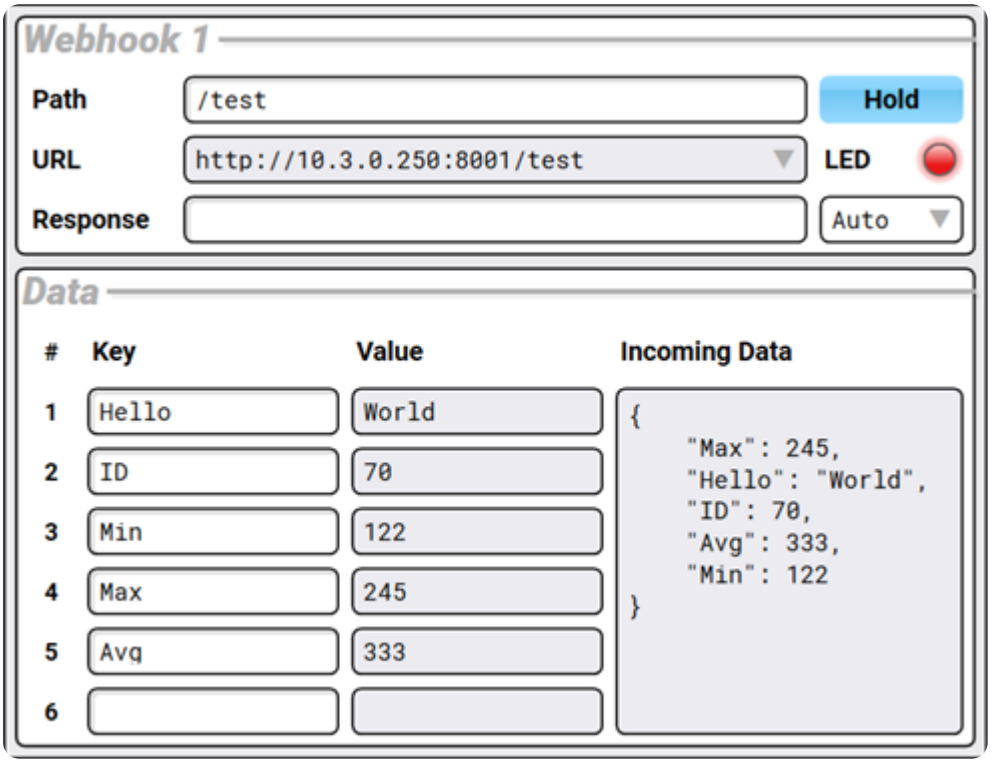
Status

Controls name: `Status`
Type: `Indicator-Text`
Description: Displays the verbose current operational status of the plugin:

- `OK` - the plugin is running correctly
- `Missing - [Description]` - the plugin lacks something essential to function properly
- `Fault - [Description]` - a fault has occurred during execution

Webhook Page

> One Webhook page is generated per configured webhook, named **Webhook 1**, **Webhook 2**, ... up to **Webhook N** where N is the **Webhook Count** property. The number of Key/Value rows on each page is determined by the **Data Pair Count**property.



WEBHOOK SETTINGS

Webhook Path

Controls name: `WebhookiPath`
Type: `Text-Text Edit`
Description: URL path the HTTP server listens on for this webhook (for example `/door1`). The value is auto-sanitized: only `[a-zA-Z0-9-_.~/]` characters are kept, repeated slashes are collapsed, any trailing slash is removed, and a leading slash is forced.

Webhook URL

Controls name: `WebhookiUrl`
Type: `Text-ComboBox`
Description: Read-only combo box listing the full URL for this webhook on every active network interface. Use it as a quick reference for the address external systems should call.

Webhook Hold

Controls name: `WebhookiHold`
Type: `Button-Toggle`
Description: When `true` (default), the **Value** controls keep the last received value until the next matching request. When `false`, the **Value** controls are cleared after each request, producing a pulse-like behavior.

Webhook LED

Controls name: `WebhookiLed`
Type: `Indicator-Led`
Description: Flashes for `0.4 s` each time a request hits this webhook's Path, providing visual confirmation that the request was received.

Webhook Data

Controls name: `WebhookiData`
Type: `Indicator-Text`
Description: Pretty-printed JSON dump of the full parameter set from the most recent request (query string and JSON body merged).

Webhook Response

Controls name: `WebhookiResponse`
Type: `Text-Text Edit`
Description: Optional response body returned to the caller for every request that matches this webhook. The current value of the control at the moment the request is received is sent back synchronously - the plugin does not wait for the control to update. If the value parses as JSON, it is returned with `Content-Type: application/json`; otherwise it is returned as `text/plain; charset=utf-8`. Leave empty to return `200 OK` with no body.

Webhook Response Code

Controls name: `WebhookiCode`
Type: `Text-ComboBox`
Choices: `Auto`, `200`, `201`, `202`, `204`, `400`, `401`, `403`, `404`, `409`, `422`, `429`, `500`, `502`, `503`
Description: Per-webhook HTTP status code returned to the caller. `Auto` (default) uses the normal behavior (`200` for successful webhook response, or `200` with no body when **Webhook Response** is empty). Selecting any explicit code forces that status for this webhook. This applies whether **Webhook Response** is empty (status only) or contains a body (status + body).

KEY / VALUE PAIRS

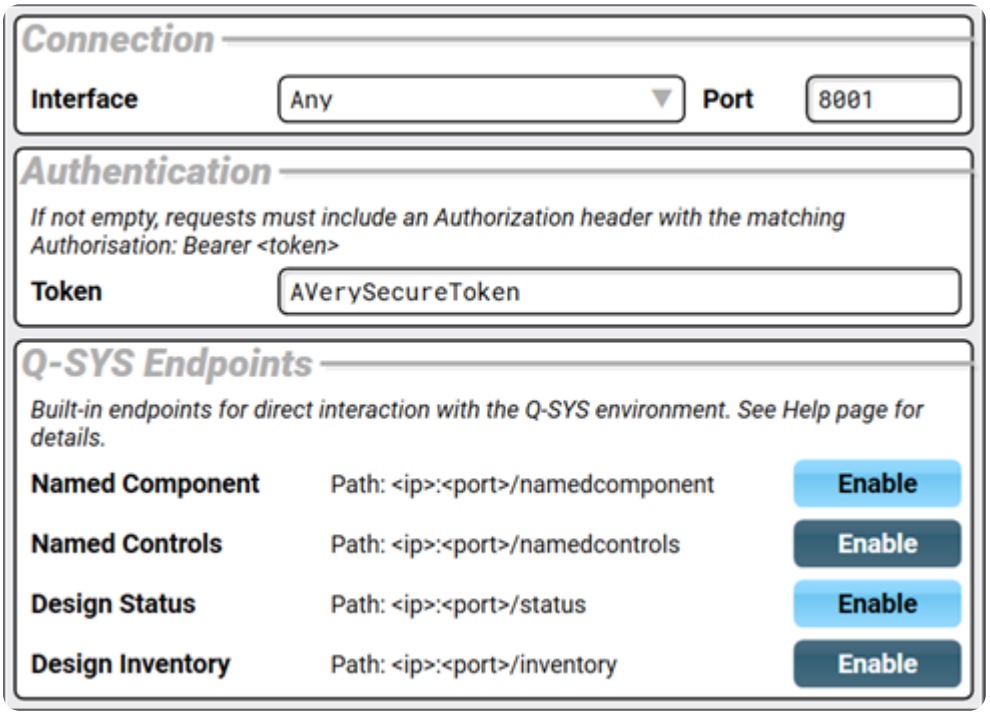
Webhook Key

Controls name: `WebhookiKeyj`
Type: `Text-Text Edit`
Description: Key name to extract from incoming request parameters. When the incoming payload contains a matching key, its value is forwarded to the corresponding **Value** control.

Webhook Value

Controls name: `WebhookiValuej`
Type: `Indicator-Text`
Description: Value extracted from the most recent request for the matching Key. Behavior depends on the **Hold** toggle: held until next request when `true`, cleared after each request when `false`.

Setup Page



CONNECTION

Interface

Controls name: `Interface`
Type: `Text-ComboBox`
Description: Selects which network interface (IP address) the server accepts requests on (Host-header based filtering). `Any` accepts requests on all interfaces. The list refreshes from `Network.Interfaces()`.

Port

Controls name: `Port`
Type: `Knob-Integer`
Description: TCP port the HTTP server listens on. Allowed range is `8000` to `9999`. Default `8001`.

Token

Controls name: `Token`
Type: `Text-Text Edit`
Description: Bearer token required in the `Authorization: Bearer <token>` header of every incoming request. Leave blank to disable authentication.

BUILT-IN ENDPOINTS

Design Status Endpoint

Controls name: `DesignStatus`
Type: `Button-Toggle`
Description: Enables the built-in `/status` endpoint. When `true` (default), `GET` or `POST` requests to `<ip>:<port>/status` return the result of `Design.GetStatus()` encoded as JSON. When `false`, the endpoint returns `404 Not Found`.

Design Inventory Endpoint

Controls name: `DesignInventory`
Type: `Button-Toggle`
Description: Enables the built-in `/inventory` endpoint. When `true` (default), `GET` or `POST` requests to `<ip>:<port>/inventory` return the result of `Design.GetInventory()` encoded as JSON. When `false`, the endpoint returns `404 Not Found`.

Named Component Endpoint

Controls name: `NamedComponent`
Type: `Button-Toggle`
Description: Enables the built-in `/namedcomponent` action endpoint. When `true`, external systems can drive any Q-SYS Named Component control by posting `Component`, `Controls`, `Action`, `Value`, and optional `RampTime` parameters. This bypasses complicated schematic connections and long path addresses by interacting with another component directly through the name assigned to it in the Q-SYS Named Component library. The endpoint path is `<ip>:<port>/namedcomponent`.

Named Controls Endpoint

Controls name: `NamedControls`
Type: `Button-Toggle`
Description: Enables the built-in `/namedcontrols` action endpoint. When `true`, external systems can drive any Q-SYS Named Control by posting `Controls`, `Action`, `Value`, and

optional `RampTime` parameters. This bypasses complicated schematic connections and long path-address patterns by interacting with the built-in Q-SYS Named Controls library. The endpoint path is `<ip>:<port>/namedcontrols`.

Appendix

User-defined Webhook Request Examples

`POST http://<core_ip>:8001/door1` Body: `{"state":"open","by":"alice"}`

`GET http://<core_ip>:8001/door1?state=open&by=alice`

Both forms produce the same effect. If the JSON body and the query string define the same key, the body value wins.

Built-in Endpoints Reference

Statuses endpoints

- `GET|POST /status` - returns `Design.GetStatus()` as JSON. Gated by the `DesignStatus` toggle (enabled by default).
- `GET|POST /inventory` - returns `Design.GetInventory()` as JSON. Gated by the `DesignInventory` toggle (enabled by default).

Action endpoints

- `/namedcomponent` - drives any control of a Q-SYS Named Component
- `/namedcontrols` - drives a Q-SYS Named Control (top-level)

Both endpoints accept the same parameter set (except `Component`, which is only used by `/namedcomponent`) and can be called with `GET` (query string), `POST` (JSON body), or a mix of both. When the same key appears in both, the JSON body wins.

Parameters

Parameter	Required	Notes
`Component`	`/namedcomponent` only	The Code Name of the target Named Component
`Controls`	yes	The name of the control inside the component, or the Named Control identifier
`Action`	yes	One of `Trigger`, `SetString`, `SetPosition`, `SetValue`, `SetBoolean`,

		`Toggle`
`Value`	required except for `Trigger` and `Toggle`	The new value to assign. Type is coerced from string based on `Action` (see below)
`RampTime`	optional, only meaningful with `SetPosition`	Seconds. Sets the control's `RampTime` for the transition and resets it back to `0` after the ramp expires

Action semantics

- `Trigger` - fires the control's trigger event. No `Value` required.
- `SetString` - assigns `Value` as a string to the control.
- `SetPosition` - assigns `Value` (number 0.0 - 1.0) as the control's normalized position. Uses `RampTime` if provided.
- `SetValue` - assigns `Value` (number) as the control's raw value.
- `SetBoolean` - assigns `Value` (`true` or `false`, as string or boolean) to the control.
- `Toggle` - inverts the control's current boolean state. No `Value` required.

Request format examples

`POST` with JSON body (recommended):

```
POST http://<core_ip>:8001/namedcomponent
Content-Type: application/json

{
  "Component": "Main_Mixer",
  "Controls":  "input.1.gain",
  "Action":    "SetPosition",
  "Value":     "0.75",
  "RampTime":  "2"
}
```

`GET` with query string (all parameters URL-encoded):

```
GET http://<core_ip>:8001/namedcomponent?Component=Main_Mixer&Controls=input.1.ga
```

More examples

Trigger a snapshot recall on a Named Component:

```
POST http://<core_ip>:8001/namedcomponent
{ "Component": "Snapshots", "Controls": "load.1", "Action": "Trigger" }
```

Mute a Named Control:

```
POST http://<core_ip>:8001/namedcontrols
{ "Controls": "Room_Mute", "Action": "SetBoolean", "Value": "true" }
```

Toggle a Named Control:

```
GET http://<core_ip>:8001/namedcontrols?Controls=Room_Mute&Action=Toggle
```

Set a string on a Named Component control:

```
GET http://<core_ip>:8001/namedcomponent?Component=Signage&Controls=message&Actio
```

Responses

- `200 OK` - the action was dispatched successfully
- `404 Not Found` - the endpoint is disabled, the component/control does not exist, or the action failed (e.g. `Action` is unknown, `Value` is missing or of the wrong type)
- `401 Unauthorized` - the `Token` property is set and the request is missing or has the wrong `Authorization: Bearer <token>` header

No JSON payload is returned by these action endpoints; only the HTTP status code matters.

How to Name a Component

- Q-SYS Designer `9.4` and lower: while in Offline mode, simply rename the component block.
- Q-SYS Designer `9.5` and higher: while in Offline mode, set the `Code Name` property in the Properties pane to the desired name, and make sure `Script Access` is set to `Script` or `All`.
- The `Code Name` property is the identifier the plugin uses to reach the component.

Control Pins

PIN NAME	VALUE	STRING	POSITION	PINS AVAILABLE
Status	-	-	-	-

Status	0	(Status String)	0	Output
Setup	-	-	-	-
Interface	-	(String)	-	Input / Output
Port	8000-9999	(Integer)	0-1	Input / Output
Token	-	(String)	-	Input / Output
Design Status Endpoint	0 1	false true	0 1	Input / Output
Design Inventory Endpoint	0 1	false true	0 1	Input / Output
Named Component Endpoint	0 1	false true	0 1	Input / Output
Named Controls Endpoint	0 1	false true	0 1	Input / Output
Webhook 1	-	-	-	-
Webhook 1 Path	-	(String)	-	Input / Output
Webhook 1 URL	-	(String)	-	Output
Webhook 1 Hold	0 1	false true	0 1	Input / Output
Webhook 1 LED	0 1	false true	0 1	Output
Webhook 1 Data	-	(JSON String)	-	Output
Webhook 1 Response	-	(String)	-	Input / Output
Webhook 1 Key 1	-	(String)	-	Input / Output
Webhook 1 Value 1	-	(String)	-	Output
Webhook 1 Key 2	-	(String)	-	Input / Output
Webhook 1 Value 2	-	(String)	-	Output

Webhook 1 Key 3	-	(String)	-	Input / Output
Webhook 1 Value 3	-	(String)	-	Output
Webhook 1 Key 4	-	(String)	-	Input / Output
Webhook 1 Value 4	-	(String)	-	Output

Additional Key/Value pin pairs (5 to N) follow the same pattern based on the **Data Pair Count** property (4 to 32). Additional Webhook blocks (Webhook 2 to Webhook 6) follow the same Path / URL / Hold / LED / Data / Key / Value pattern and are generated based on the **Webhook Count** property.

support@ad-da.ca | *Copyright © 2026 AD-DA.CA*