AD-DA.ca, EzJSON

Plugin Description

The AD-DA.ca EzJSON plugin brings JSON processing capabilities to Q-SYS, enabling seamless encoding and decoding of JSON data structures.

JSON (JavaScript Object Notation), recognized as a standard for data interchange in modern applications, allows structured data exchange with web services, APIs, and third-party systems in a lightweight and human-readable format.

The plugin supports dual-mode operation. In **Encode** mode, key/value controls are transformed into a JSON string output. In **Decode** mode, an input JSON string is parsed and mapped back to key/value controls with automatic type detection.

Key features

- **Dual operation mode** - switch between `Encode` and `Decode`
- **Dynamic data pairs** - configure from `1` to `64` key/value rows per instance
- **Multi-type support** - `string`, `number`, `boolean`, `null`, `object`, `array`
- **Pretty JSON output** - optional formatted JSON with indentation
- **Decode key selection** - select parsed keys using per-row `Select` controls
- **Per-row hold behavior** - retain decoded values with `Hold` toggles
- **Status diagnostics** - detailed status and fault messaging
- **Debug print levels** - console verbosity control via `Debug Print` property

Use cases

- Build JSON payloads for external REST APIs from Q-SYS controls
- Parse incoming JSON-like text payloads into deterministic control rows
- Bridge Q-SYS logic with cloud services and control orchestrators
- Create layered/nested JSON using daisy-chained EzJSON instances

Configuration Overview

Plugin version

This document is written for the plugin :

- AD-DA.ca EzJSON V1.1.0.0

Release note

[2026-06] : V1.1.0.0

- Performance optimisation
- New UI

[2025-08] : V1.0.1.0

- Initial release to Asset Manager

[2025-08] : V1.0.0.0

- Initial release of EzJSON plugin
- Support for `String`, `Number`, `Boolean`, `Null`, `Object`, `Array`
- Dual mode operation (`Encode` / `Decode`)
- Support for up to `64` key-value pairs
- Error handling and validation

QDS compatibility

This plugin has been developed and tested on :

- Q-SYS Designer version `9.13.1`
- Q-SYS Designer version `10.0.1`

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.

Mode

Type: `enum`
Choices: `Encode`, `Decode`
Default Value: `Encode`
Description: Selects plugin operating mode.

Data Pair Count

Type: `integer`
Min: `1`
Max: `64`
Default Value: `4`
Description: Number of key/value rows available for JSON processing.

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 in Q-SYS Designer Console.

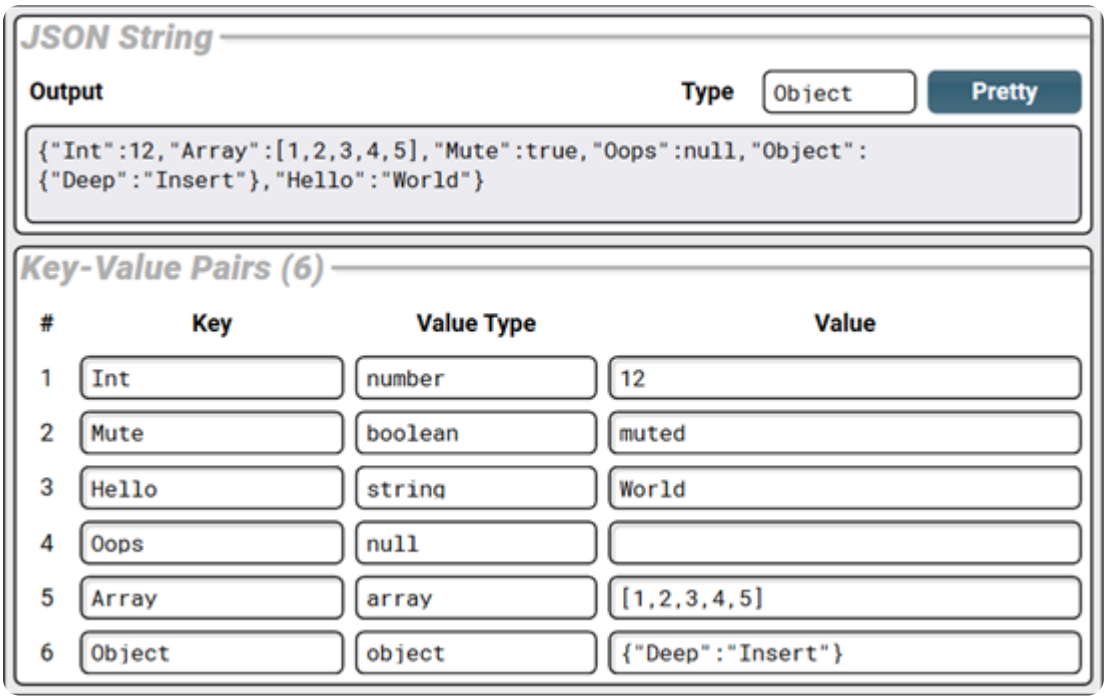
Controls

Status

Status

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

Encode Mode



JSON Type

Controls name: `JsonType`
Type: `Text-ComboBox`
Description: Selects JSON root structure generated by the encoder (`Object` or `Array`).

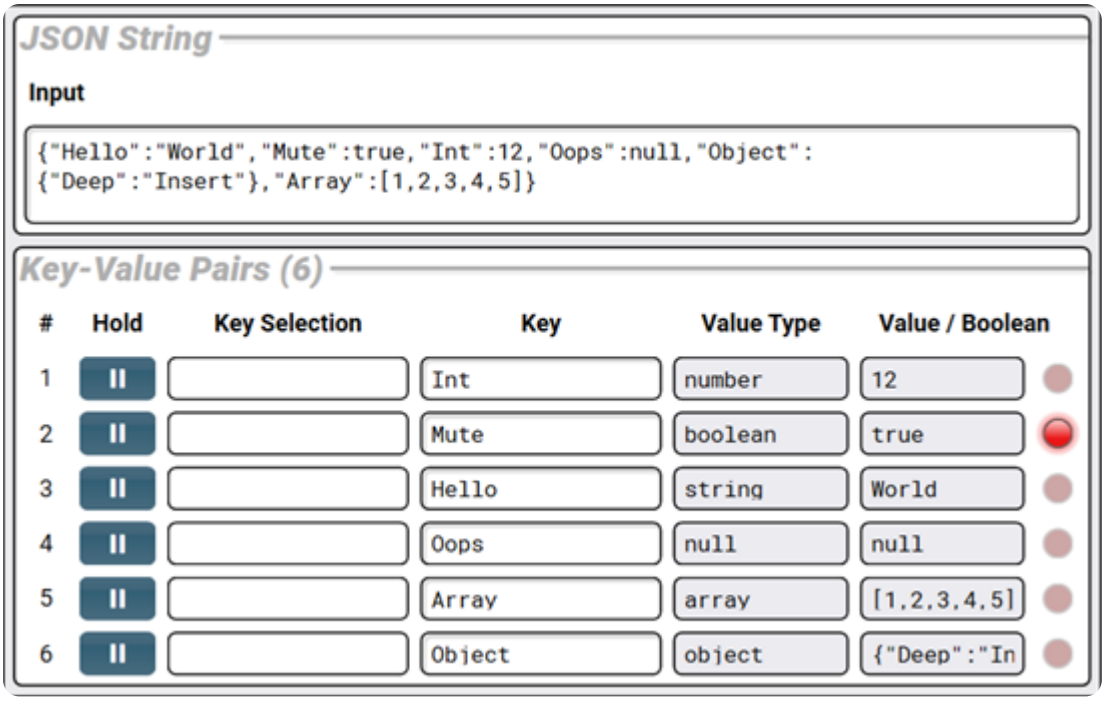
JSON Pretty

Controls name: `JsonPretty`
Type: `Button-Toggle`
Description: Enables pretty-formatted JSON output with indentation.

JSON Output

Controls name: `JsonString`
Type: `Indicator-Text`
Description: Resulting JSON string after encoding configured rows.

Decode Mode



JSON Input

Controls name: `JsonString`
Type: `Text-Text Edit`
Description: Input JSON string parsed by the decoder.

Data Pair Controls

For each row `x` from `1` to `Data Pair Count`:

Data/x/Key

Controls name: `Keyx`
Type: `Text`
Description: Key name used for JSON mapping in encode/decode flows.

Data/x/Value Type

Controls name: `ValueTypex`
Type: `Text` (Encode) / `Indicator-Text` (Decode)
Description: In Encode mode, selects the outgoing value type. In Decode mode, displays detected type.

Supported values:

- `string`
- `number`
- `boolean`
- `null`

- `object`
- `array`

Data/x/Value

Controls name: `Valuex`
Type: `Text` (Encode) / `Indicator-Text` (Decode)
Description: Input/output value associated with `Keyx`.

Data/x/Value Boolean (Decode)

Controls name: `ValueBooleanx`
Type: `Indicator-Led`
Description: Read-only boolean indicator enabled when decoded type is boolean.

Data/x/Select (Decode)

Controls name: `KeySelectx`
Type: `Text-ComboBox`
Description: Selects available decoded keys and copies selection into `Keyx`.

Data/x/Hold (Decode)

Controls name: `Holdx`
Type: `Button-Toggle`
Description: Holds previous decoded value for that row when enabled.

Usage

Encoding JSON

1. Set `Mode` to `Encode`
2. Set `Data Pair Count`
3. For each row:

- Enter key in `Keyx`
- Select type in `ValueTypex`
- Enter value in `Valuex`

4. Optionally enable `JsonPretty`
5. Read result from `JsonString`

Example input:

- `Key1`: `temperature`, `ValueType1`: `number`, `Value1`: `23.5`
- `Key2`: `unit`, `ValueType2`: `string`, `Value2`: `celsius`
- `Key3`: `active`, `ValueType3`: `boolean`, `Value3`: `true`

Example output:

```
{"temperature":23.5,"unit":"celsius","active":true}
```

Decoding JSON

1. Set `Mode` to `Decode`
2. Set `Data Pair Count`
3. Enter JSON text in `JsonString`
4. Use `KeySelectx` to choose keys
5. Review decoded `ValueTypex`, `Valuex`, and `ValueBooleanx`
6. Use `Holdx` to retain values by row

Example input:

```
{"name":"Living Room","volume":75,"muted":false}
```

Example decoded rows:

- Key: `name`, Value Type: `string`, Value: `Living Room`
- Key: `volume`, Value Type: `number`, Value: `75`
- Key: `muted`, Value Type: `boolean`, Value: `false`

Deep Encode and Deep Decode

EzJSON supports nested workflows by daisy-chaining multiple plugin instances.

- Deep encode: feed `JsonString` output from one instance into another row as `object`/`array`
- Deep decode: decode top-level JSON, then feed nested JSON values to downstream instances

Data Types

String

Text data enclosed in quotes.

Number

Numeric data (integer or floating-point).

Boolean

Boolean data represented as `true`/`false`.

Null

JSON null value.

Object

Nested JSON object with key/value pairs.

Array

Ordered JSON array.

Boolean Value Accepted Inputs

The parser accepts these values as `true` (case-insensitive):

- Basic: `true`, `1`, `yes`, `yup`, `y`
- Status/control words: `good`, `ok`, `on`, `active`, `valid`, `ready`, `up`, `enable`, `enabled`
- Process/state words: `start`, `started`, `run`, `running`, `launch`, `launched`, `open`, `opened`, `connected`, `online`, `available`, `pass`, `success`, `go`, `confirm`
- Audio/security words: `mute`, `muted`, `live`, `lock`, `locked`, `armed`
- Numeric values greater than `0`

All other values are treated as `false`.

Control Pins

PIN NAME	VALUE	STRING	POSITION	PINS AVAILABL
Status	-	-	-	-
Status	0..5	Status String	0..1	Output
Encode Mode	-	-	-	-
JsonType	-	Object/Array	-	Input / Output
JsonPretty	0..1	false/true	0..1	Input / Output
JsonString (Encode)	-	JSON String	-	Output

Decode Mode	-	-	-	-
JsonString (Decode)	-	JSON String	-	Input
Data Pair x (1..64)	-	-	-	-
Keyx	-	Key Name	-	Input / Output
ValueTypex (Encode)	-	string/number/boolean/null/object/array	-	Input / Output
ValueTypex (Decode)	-	Detected Type	-	Output
Valuex (Encode)	-	Value	-	Input / Output
Valuex (Decode)	-	Value	-	Output
ValueBooleanx (Decode)	0..1	false/true	0..1	Output
KeySelectx (Decode)	-	Selected Key	-	Input / Output
Holdx (Decode)	0..1	false/true	0..1	Input