R’DASH API Reference

Version: 1.1

Protocol: HTTP / HTTPS
Author: Vamsi Karnam
License: Apache 2.0

Table of Contents

Overview

R’DASH exposes a **stateless HTTP + WebSocket API** for live telemetry ingestion, visualization, and management. It accepts push events from **R’DASH Agents** or any compatible client, and serves data, metadata, and MJPEG streams to scripts.

- Base URL example: http://host-ip:8080 or https://host-ip:8443
- All data is held **in-memory only** (no persistence).
- All requests require authentication unless the server was started without --auth-token.

Authentication

Bearer Token

All protected endpoints accept:

Authorization: Bearer <YOUR_TOKEN>

Example with curl:

```
curl -H "Authorization: Bearer devtestkey" http://host-ip:8080/api/robots
```

Query-param token (for video endpoints only)

You can alternatively append:

?token=<YOUR_TOKEN>

Example:

https://host-ip:8443/video/mybot/camera%2Fimage%2Fcompressed?token=devtestkey

---

Data Handling

Data Model

Concept	Meaning
Robot	Unique name of a connected robot or agent (--robot-name).
Sensor	A topic name, flattened to a URL-safe identifier (slashes encoded as %2F).
Metric key	A flattened field key from the message (e.g., twist.linear.x, data).
t	UNIX timestamp in seconds (float).
data	Map of { field_name: value } for numeric pushes.

Data Types

Field	Type	Description
robot	text	Robot name
sensor	text	Sensor name (topic path)
type	text	ROS2 type (sensor_msgs/msg/Image or CompressedImage)
image	file	Image binary (image/jpeg, image/png, etc.)
data	data	Numeric value (float32)

Status Codes

Code	Meaning
200	OK
201	Created (optional)
400	Malformed input
401	Unauthorized (bad/missing token)
404	Robot/sensor not found
413	Payload too large (violates max_image_bytes, etc.)
500	Server error

Core Endpoints

1. POST /api/push

- Push a **numeric sample** (float values).

Request (JSON)

```
{
  "robot": "mybot",
  "sensor": "drone/speed_mps",
  "t": 1731010101.234,
  "data": { "data": 2.22 },
  "type": "std_msgs/msg/Float32",
  "units": { "data": "m/s" }
}
```

Curl

```
curl -X POST http://host-ip:8080/api/push -H "Authorization: Bearer devtestkey" \
-H "Content-Type: application/json" \
-d
'{"robot":"mybot","sensor":"drone/speed_mps","t":1731010101.234,"data":{"data":2.22},"type":"st
d_msgs/msg/Float32","units":{"data":"m/s"}}'
```

Python

```
import requests, time

payload = {
    "robot": "mybot",
    "sensor": "drone/speed_mps",
    "t": time.time(),
    "data": {"data": 2.22},
    "type": "std_msgs/msg/Float32",
    "units": {"data": "m/s"}
}

r = requests.post("http://host-ip:8080/api/push",
              headers={"Authorization": "Bearer devtestkey"},
              json=payload)

print(r.status_code)
```

Response

```
{ "status": "ok" }
```

2. POST /api/push_image

- Push an **image or video frame**.

Curl

```
curl -X POST http://host-ip:8080/api/push_image \
 -H "Authorization: Bearer devtestkey" \
 -F "robot=mybot" \
 -F "sensor=drone/camera/image/compressed" \
 -F "type=sensor_msgs/msg/CompressedImage" \
 -F "image=@frame.jpg;type=image/jpeg"
```

Python

```
import requests
with open("frame.jpg", "rb") as f:
  files = {"image": ("frame.jpg", f, "image/jpeg")}
  data = {
    "robot": "mybot",
    "sensor": "drone/camera/image/compressed",
    "type": "sensor_msgs/msg/CompressedImage"
  }
  r = requests.post("http://host-ip:8080/api/push_image",
           headers={"Authorization": "Bearer devtestkey"},
           files=files, data=data)
print(r.json())
```

Response

```
{ "status": "ok", "bytes": 84231 }
```

3. POST /api/push_tf

- Push **TF edges** (parent → child links).

Request

```
{
  "robot": "mybot",
  "edges": [["base_link","camera_link"],["base_link","lidar_link"]]
}
```

Curl

```
curl -X POST http://host-ip:8080/api/push_tf \
  -H "Authorization: Bearer devtestkey" \
  -H "Content-Type: application/json" \
  -d '{"robot":"mybot","edges":[["base_link","camera_link"],["base_link","lidar_link"]]}'
```

Response

```
{ "status": "ok", "count": 2 }
```

---

4. POST /api/push_text

- Push a **log / textual line**.

Request

```
{
  "robot": "mybot",
  "sensor": "drone/log",
  "t": 1731010105.0,
  "text": "[DRONE] armed=1 mode=GUIDED"
}
```

Curl

```
curl -X POST http://host-ip:8080/api/push_text \
  -H "Authorization: Bearer devtestkey" \
  -H "Content-Type: application/json" \
  -d '{"robot":"mybot","sensor":"drone/log","t":1731010105.0,"text":"[DRONE] armed=1
mode=GUIDED"}'
```

Python

```
import requests, time
payload = {
   "robot": "mybot",
   "sensor": "drone/log",
   "t": time.time(),
   "text": "[DRONE] altitude=5.1m speed=2.2m/s"
}
requests.post("http://host-ip:8080/api/push_text",
          headers={"Authorization": "Bearer devtestkey"},
          json=payload)
```

---

5. GET /api/robots

- List all active robots and their sensors.

Curl

```
curl -s http://host-ip:8080/api/robots -H "Authorization: Bearer devtestkey" | jq
```

Response

```
{
 "robots": [
  {
   "name": "mybot",
   "sensors": ["drone/speed_mps","drone/altitude_m","drone/log"],
   "last_seen": 1731010111.22
  }
 ]
}
```

6. GET /api/meta/<robot>/<sensor>

- Retrieve metadata for a given sensor.

Curl

```
curl http://host-ip:8080/api/meta/mybot/drone%2Fspeed_mps \
  -H "Authorization: Bearer devtestkey"
```

Response

```
{
  "type": "std_msgs/msg/Float32",
  "units": { "data": "m/s" },
  "last": 1731010110.232,
  "status": "active"
}
```

---

7. GET /api/history/<robot>/<sensor>

- Return recent numeric samples (from RAM ring buffer).

Curl

```
curl http://host-ip:8080/api/history/mybot/drone%2Fspeed_mps \
  -H "Authorization: Bearer devtestkey"
```

Response

```
{
  "t": [1731010098.1, 1731010098.2, 1731010098.3],
  "data": { "data": [2.15, 2.20, 2.22] }
}
```

8. GET /api/text_history/<robot>/<sensor>

- Return the recent text log lines for that sensor.

Response

```
{
 "lines": [
  {"t":1731010101.0, "text":"[DRONE] armed=1"},
  {"t":1731010102.0, "text":"[DRONE] altitude=5.2"}
 ]
}
```

---

9. POST /api/delete_series/<robot>/<sensor>

- Clear numeric series (RAM reset).

```
curl -X POST http://host-ip:8080/api/delete_series/mybot/drone%2Fspeed_mps \
 -H "Authorization: Bearer devtestkey"
```

---

10. POST /api/delete_text/<robot>/<sensor>

- Clear text/log history (RAM reset).

```
curl -X POST http://host-ip:8080/api/delete_text/mybot/drone%2Flog \
 -H "Authorization: Bearer devtestkey"
```

Video Streaming Endpoint

1. GET /video/<robot>/<sensor>

- Returns an **MJPEG multipart stream** suitable for embedding in a browser or HTML <img> tag.

Example (browser)

http://host-ip:8080/video/mybot/drone%2Fcamera%2Fimage%2Fcompressed?token=devtestkey

Curl

```
curl -H "Authorization: Bearer devtestkey" \ http://host-ip:8080/video/mybot/drone%2Fcamera%2Fimage%2Fcompressed --output stream.mjpeg
```

WebSocket Stream

URL: /ws
Protocol: wss://HOST/ws or ws://HOST/ws

- Authenticated via Bearer token in headers (handled automatically by browser).
- Emits coalesced JSON updates for all active sensors at RDASH_WS_FLUSH_HZ (default 20 Hz).

Sample event payload

```
{
  "robot": "mybot",
  "sensor": "drone/speed_mps",
  "t": 1731010102.25,
  "data": { "data": 2.23 }
}
```

Example Workflow (End-to-End)

1. **Push a numeric sample**

   ```
   curl -X POST $BASE/api/push "${HDR[@]}" -d '{"robot":"demo","sensor":"power/battery_mv","t":1731010100,"data":{"data":25000},"type":"std_msgs/msg/Float32","units":{"data":"mV"}}'
   ```

2. **Fetch history**

   ```
   curl $BASE/api/history/demo/power%2Fbattery_mv "${HDR[@]}"
   ```

3. **Push a text log line**

   ```
   curl -X POST $BASE/api/push_text "${HDR[@]}" -d '{"robot":"demo","sensor":"robot/log","t":1731010110,"text":"Battery 25.0mV"}'
   ```

4. **View MJPEG feed**
   Open:

   ```
   http://host-ip:8080/video/demo/drone%2Fcamera%2Fimage%2Fcompressed?token=devtestkey
   ```

5. **Inspect metadata**

   ```
   curl $BASE/api/meta/demo/power%2Fbattery_mv "${HDR[@]}"
   ```

Notes

- All times are **UNIX float seconds (UTC)**.
- JSON keys are **case-sensitive**.
- Images larger than rdash_MAX_IMAGE_BYTES are rejected.
- Text history and numeric samples are **per-sensor ring buffers** trimmed to rdash_MAX_SAMPLES.
- Restarting the server wipes all state.

Attribution

R'DASH - Robot Information Telemetry Transport Dashboard
Developed by **Vamsi Karnam**, 2025.
Apache License 2.0

Appendix

I. DDS Topics

Rdash_v1.1 introduces support for DDS-originated telemetry by design.
Any topic whose path includes the segment **dds** (case-insensitive) is automatically routed to the **DDS side panel** in the web dashboard, keeping infrastructure and diagnostic data separate from regular robot sensors.

This feature is entirely **agnostic** - it does not introduce a new data type or API endpoint.
DDS data continues to flow through the same core interfaces:

DDS Data Type	**API Endpoint**	**Visualization**
Numeric DDS metric	/api/push	Displayed under **DDS → Metrics**
Textual DDS log	/api/push_text	Displayed under **DDS → Logs**
(Optional) DDS media data	/api/push_image	Accepted but **not visualized**

Behavior

- The presence of "dds" in a topic's path determines whether it appears in the DDS drawer.
- DDS filtering is purely **semantic** - it changes presentation, not storage or transport.
- Numeric and text topics under dds namespaces behave identically to their non-DDS counterparts, but are grouped separately in the UI.

DDS Topic Guidelines

The DDS panel is designed for **numeric** and **textual** system telemetry such as diagnostics, CPU usage, latency, and network metrics.
While R'DASH will technically accept sensor_msgs/Image or sensor_msgs/CompressedImage topics under a dds namespace, such data will **not be visualized** as live camera feeds.

To ensure correct behavior:

- Use the dds namespace for diagnostics and performance data.
- Keep camera, LiDAR, or other media topics outside the DDS path.

Examples:

Topic_1: */foo/dds/network/latency_ms*

Topic_2: */bar/dds/system/diagnostics*