

Output Format Comparison

The Payload Schema decoder can output data in multiple formats for different platforms and standards.

Example

Schema: Environmental sensor with temperature, humidity, pressure, CO2, battery

Payload: 092982140C03520D (8 bytes)

1. Raw Format (Default)

Simple flat dictionary - easiest to use in applications.

```
{  
    "temperature": 23.45,  
    "humidity": 65.0,  
    "pressure": 1013.2,  
    "co2": 850,  
    "battery": 3.3  
}
```

Use case: Direct application use, custom backends, simple integrations.

2. IPSO Smart Objects Format

OMA LwM2M standard object IDs - interoperable with LwM2M servers.

```
{  
    "3303": {  
        "value": 23.45,  
        "unit": "\u00b0C"  
    },  
    "3304": {  
        "value": 65.0,  
        "unit": "%RH"  
    },  
    "3315": {  
        "value": 1013.2,  
        "unit": "hPa"  
    },  
    "3325": {  
        "value": 850,  
        "unit": "ppm"  
    },  
    "3316": {  
        "value": 3.3,  
        "unit": "V"  
    }  
}
```

IPSO Object Reference: | Object ID | Name | |——|——|——| | 3303 | Temperature Sensor | | 3304 | Humidity Sensor | | 3315 | Barometer | | 3316 | Voltage | | 3325 | CO2 Sensor | | 3330 | Distance | | 3301 | Illuminance |

Use case: LwM2M platforms (Leshan, Wakaama), cloud IoT services, OMA-compliant systems.

3. SenML Format (RFC 8428)

IETF Sensor Measurement Lists standard.

```
[  
  {  
    "n": "temperature",  
    "v": 23.45,  
    "u": "°C"  
  },  
  {  
    "n": "humidity",  
    "v": 65.0,  
    "u": "%RH"  
  },  
  {  
    "n": "pressure",  
    "v": 1013.2,  
    "u": "hPa"  
  },  
  {  
    "n": "co2",  
    "v": 850,  
    "u": "ppm"  
  },  
  {  
    "n": "battery",  
    "v": 3.3,  
    "u": "V"  
  }  
]
```

SenML Fields: - n - name - v - value (numeric) - vs - value (string) - vb - value (boolean) - u - unit - t - time (optional) - bt - base time (optional)

Use case: CoAP integration, CBOR encoding, RFC-compliant systems.

4. TTN Normalized Format

The Things Network v3 payload format with `normalized_payload`.

```
{  
  "decoded_payload": {  
    "temperature": 23.45,  
    "humidity": 65.0,  
    "pressure": 1013.2,  
    "co2": 850,  
    "battery": 3.3  
  },  
  "normalized_payload": [  
    {
```

```

    "measurement": {
      "temperature": {
        "value": 23.45,
        "unit": "°C"
      }
    },
    {
      "measurement": {
        "humidity": {
          "value": 65.0,
          "unit": "%RH"
        }
      }
    },
    {
      "measurement": {
        "pressure": {
          "value": 1013.2,
          "unit": "hPa"
        }
      }
    },
    {
      "measurement": {
        "co2": {
          "value": 850,
          "unit": "ppm"
        }
      }
    },
    {
      "measurement": {
        "battery": {
          "value": 3.3,
          "unit": "V"
        }
      }
    }
  ]
}

```

Use case: The Things Network, TTN Console, TTN integrations, Cayenne LPP compatibility.

Format Comparison

Format	Structure	Size	Interoperability	Best For
Raw	Flat dict	Smallest	Application-specific	Custom backends
IPSO	Object-based	Medium	High (OMA/LwM2M)	LwM2M platforms
SenML	Record array	Medium	High (IETF/CoAP)	RFC-compliant systems

Format	Structure	Size	Interoperability	Best For
TTN	Normalized	Largest	TTN ecosystem	TTN integrations

Schema Definition

To enable semantic output formats, add `semantic` annotations to fields:

```
fields:
  - name: temperature
    type: s16
    mult: 0.01
    unit: "°C"
    semantic:
      ipso: 3303
      senml: "urn:dev:ow:temp"

  - name: humidity
    type: u8
    mult: 0.5
    unit: "%RH"
    semantic:
      ipso: 3304
```

API Usage

Python

```
from schema_interpreter import SchemaInterpreter

interpreter = SchemaInterpreter(schema)
result = interpreter.decode(payload)

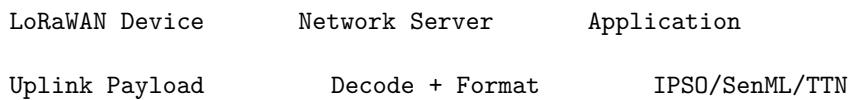
# Raw format (default)
raw = result.data

# IPSO format
ipso = interpreter.get_semantic_output(result.data, 'ipso')

# SenML format
senml = interpreter.get_semantic_output(result.data, 'senml')

# TTN format
ttn = interpreter.get_semantic_output(result.data, 'ttn')
```

Network Server Integration



The network server can:

1. Decode payload using schema
2. Transform to target format based on application requirements
3. Forward to appropriate backend (LwM2M, MQTT, HTTP, etc.)