

Opening CIP: An OpenAPI for Describing CIP-Enabled Products

Greg Majcher, Rockwell Automation
Joakim Wiberg, ODVA

October 21, 2026

Agenda

01 **Background & Motivation**
Why standardized device APIs matter in industrial automation

02 **Problem Statement**
Current limitations in accessing device information

03 **Proposed Approach**
REST API for CIP — architecture and principles

04 **OpenAPI Deep Dive**
Schema design, versioning, servers block, tooling, security

05 **API Endpoints & Structure**
Device identity, object discovery, attributes, documentation

06 **Use cases**
Device discovery, feature identification, data harvesting

07 **Implementation**
Python server, Excel VBA, Power Automate / Power BI

08 **Results & Conclusion**
Benefits and next steps

© 2026 ODVA, Inc.
All rights reserved.

Background & Motivation

- Industrial automation relies on interoperable communication across a growing diversity of CIP-enabled devices from many vendors
- Product information — capabilities, characteristics, metadata — remains largely inaccessible in a standardized, machine-readable form
- Engineers and integrators rely on manual processes or proprietary tools to retrieve and work with product information
- Industrial systems increasingly intersect with data-driven applications, digital repositories, and modern software toolchains
- The need for structured, programmatic access to device information is becoming more pressing

© 2026 ODVA, Inc.
All rights reserved.

Problem Statement

Current Challenges

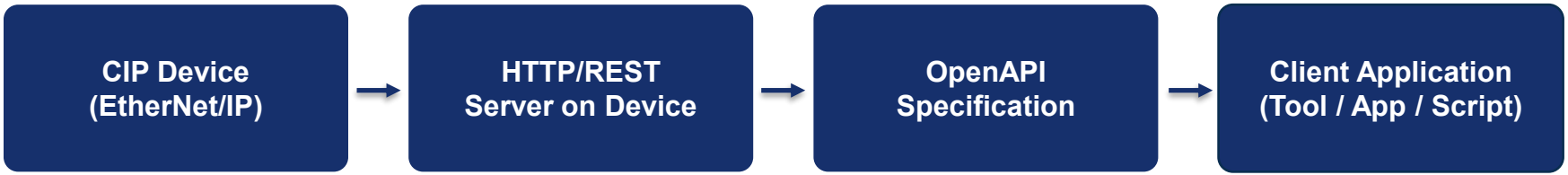
- EDS files are widely used but are static artifacts designed for configuration tooling, not web-friendly programmatic access
- No standardized REST interface exists for accessing device identity, object model, or live attribute values
- Integration with data platforms requires deep CIP expertise and custom development per device type
- Manual processes or proprietary tools are needed to retrieve and work with product information
- No mechanism for vendors to expose product documentation directly from the device

Business Impact

High	Integration cost for multi-vendor installations
Slow	Time-to-insight for device inventory and diagnostics
Manual	Asset management and documentation workflows
Limited	Ability to harvest device data at scale

© 2026 ODVA, Inc.
All rights reserved.

Proposed Approach: REST API for EtherNet/IP devices



Design Principles

Interoperability

A common interface any client, regardless of vendor or platform, can use to retrieve product information from any compliant CIP device.

Simplicity

A well-bounded set of product attributes through a small number of clearly defined endpoints — avoiding unnecessary complexity while remaining practically useful.

Extensibility

Vendors can supplement the core attribute set with product-specific information without breaking compatibility with generic clients.

Read-only + HTTPS

The API exposes product information but does not accept writes. Where TLS is available, all communication is encrypted — consistent with CIP Security practices.

 © 2026 ODVA, Inc. All rights reserved.

The OpenAPI Specification

What is OpenAPI?

- A standardized, language-agnostic format for describing REST APIs — in YAML or JSON
- Formally specifies endpoints, parameters, request and response structures, and metadata
- Both human-readable and machine-processable — enables automatic code generation, interactive documentation, and validation
- Version 3.1.0 used — aligned with JSON Schema for rich type definitions
- Widely supported by tools including Swagger Editor, Redoc, Connexion

```yaml
paths:
  /weather/current:
    get:
      summary: Get current weather for a city
      parameters:
        - name: city
          in: query
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Current weather data
          content:
            application/json:
              schema:
                type: object
        '404':
          description: City not found
```


© 2026 ODVA, Inc.
All rights reserved.

The servers block: Where versioning lives

The servers block:

```
servers:
  - url: https://api.example.com/weather/v2
    description: Example weather API
```

How the full URL is formed:

```
base servers:
  https://api.example.com/weather/v2

+ path:
  /weather/current

= full URL:
  https://api.example.com/weather/v2/current
```

Why keep version out of the path?

- All path definitions stay identical across versions — nothing to rewrite when the API evolves
- Bumping to /v3 means changing one line in the servers block, not every endpoint definition
- This separation of concerns is a deliberate OpenAPI design choice

© 2026 ODVA, Inc.
All rights reserved.

The Same Pattern, Applied to CIP Devices

The CIP API definition uses the identical servers-block pattern — but with a twist: the base URL includes a variable, {device_ip}.

```yaml
servers:
  - url: http://{device_ip}/cip/v1
    description: Device-hosted API
    variables:
      device_ip:
        default: 192.168.1.1
        description: IP address of the CIP device
```

Versioning:

/identity + servers base = https://{device_ip}/cip/v1/identity

Bumping to /cip/v2 changes one line here — every endpoint definition stays untouched.

What {device_ip} does:

A server variable — a placeholder, not a fixed value. The same YAML file describes every device on the network: only {device_ip} changes per instance, from 192.168.1.1 (the documented default) to any real device address.

© 2026 ODVA, Inc.
All rights reserved.

Schema Design: Attribute Values

CIP attribute values range from simple scalars (BOOL, UINT, REAL) to complex nested STRUCTs. The schema must handle all without combinatorial explosion.

Simple scalar value — GET .../attributes/1/value

```
{
  "data_type": "UINT",
  "value": "1"
}
```

Value is always a string — clients use data_type to parse

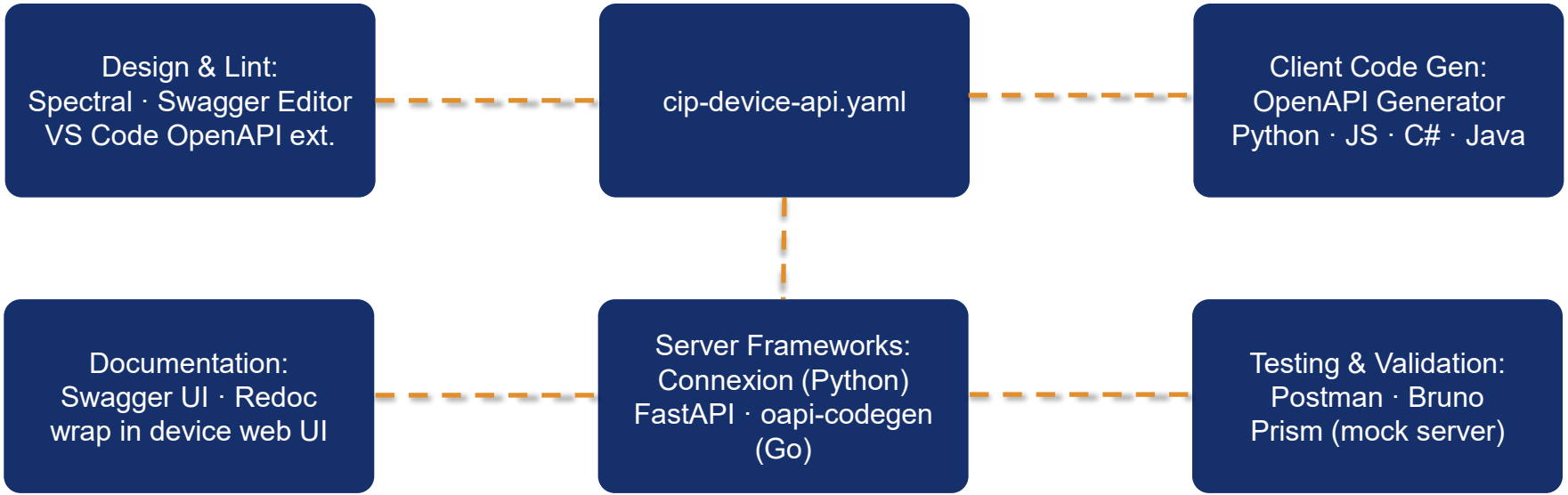
Nested STRUCT — GET /objects/245/instances/1/attributes/11/value
Each member carries its own data_type and value — the response is fully self-describing at any nesting depth

```
{
  "data_type": "STRUCT",
  "value": {
    "acd_activity": { "data_type": "USINT", "value": 2 },
    "remote_mac": { "data_type": "ARRAY[6] USINT", "value": "001D9CC83A5B" },
    "arp_pdu": { "data_type": "ARRAY[28] USINT", "value": "0001080006040002001D9CC83A5B0A000101000000000000A000102" }
  }
}
```


© 2026 ODVA, Inc.
All rights reserved.

OpenAPI Tooling Ecosystem

One YAML spec file — design, documentation, server routing, client generation, runtime validation.
One spec. Zero custom glue. All tools speak OpenAPI natively.



© 2026 ODVA, Inc.
All rights reserved.

Security & Authentication Considerations

Current v1 Approach

- **HTTPS / TLS**
  - **When HTTP is replaced by HTTPS, all communication is encrypted and authenticated using TLS — the same underlying security mechanism employed by CIP Security.**
- **Read-only**
  - **Read-only GET — attack surface is intentionally minimal**
  - **The API does not accept writes. Write access would require additional security considerations and is left for future work.**
- **No mandatory auth in v1**
  - **A complete security model — addressing authorization and the relationship with CIP Security — is acknowledged as important future work.**

© 2026 ODVA, Inc.
All rights reserved.

Security & Authentication Considerations

Future Work (v2+)

- **Write access + authorization**
  - **Extending to write operations requires controlling which clients are permitted to modify which attributes — and protection against denial-of-service threats.**
- **CIP Security alignment**
  - **The relationship between the proposed REST API security model and CIP Security deserves further study. TLS provides the same underlying transport security.**
- **API keys / OAuth 2.0**
  - **Token-based auth for multi-tenant or cloud-connected scenarios. Mechanisms declared in the OpenAPI security**
  - **Schemes block — clients and code generators understand them automatically.**

© 2026 ODVA, Inc.
All rights reserved.

API Endpoint Summary

/cip/v1/identity	Device identity	DeviceIdentity
/cip/v1/objects	List of implemented CIP object classes	CIPObjectSummary[]
/cip/v1/objects/instances	Object classes with their instance numbers	CIPObjectWithInstances[]
/cip/v1/objects/{class_id}/instances/{instance_id}/attributes	List of implemented CIP object	CIPAttribute[]
/cip/v1/objects/{class_id}/instances/{instance_id}/attributes/{attr_id}/value	Live value of a single attribute	CIPAttributeValue
/cip/v1/documentation	Product documentation links from device	DeviceIdentity

© 2026 ODVA, Inc.
All rights reserved.

Endpoint: Device Identity

GET /cip/v1/identity


```
{
  "vendor_id": 1,
  "device_type": 12,
  "product_code": 65,
  "revision": {
    "major": 3,
    "minor": 1
  },
  "status": {
    "owned": true,
    "configured": true,
    "minor_recoverable_fault": false,
    "minor_unrecoverable_fault": false,
    "major_recoverable_fault": false,
    "major_unrecoverable_fault": false,
    "extended_device_status": "No I/O connection established"
  },
  "serial_number": 16909060,
  "product_name": "1756-L84ES ControlLogix 5580",
  "catalog_number": "1756-L84ES",
  "vendor_name": "Rockwell Automation",
  "assigned_name": [
    {"language": "eng", "value": "Drive train - sub assembly 3"},
    {"language": "spa", "value": "Tren de transmisión - subconjunto 3"},
    {"language": "deu", "value": "Antriebsstrang - Unterbaugruppe 3"}
  ]
}
```


© 2026 ODVA, Inc.
All rights reserved.

Endpoint: CIP Object Discovery

GET /cip/v1/objects

GET /cip/v1/objects/instances

```
[
  {
    "class_id": 1,
    "class_name": "Identity Object",
    "instances": [1]
  },
  {
    "class_id": 4,
    "class_name": "Assembly Object",
    "instances": [1, 2, 3, 100, 101]
  },
  {
    "class_id": 6,
    "class_name": "Connection Manager Object",
    "instances": [1]
  }
]
```


© 2026 ODVA, Inc.
All rights reserved.

Endpoint: CIP Object Discovery

GET	/cip/v1/objects
GET	/cip/v1/objects/instances

What these endpoints enable

- **Runtime discovery**
  - A client determines what the device supports without any prior knowledge of its profile or EDS file.
- **Adaptive clients**
  - Integration tools query /objects and adapt their behavior based on the response — no hardcoded assumptions about device capabilities.
- **Instance mapping**
  - /objects/instances gives the full picture: Assembly Object with instances [1, 2, 3, 100, 101] in a single request.
- **No EDS required**
  - The same information an EDS file provides about object model is now queryable programmatically over HTTP.

© 2026 ODVA, Inc.
All rights reserved.

Endpoint: Attribute Discovery

GET /cip/v1/objects/{class_id}/instances/{instance_id}/attributes


```
[
  {
    "attribute_id": 1,
    "name": "Vendor ID",
    "data_type": "UINT",
    "description": "Identification number assigned to each
vendor by ODVA"
  },
  {
    "attribute_id": 2,
    "name": "Device Type",
    "data_type": "UINT",
    "description": "General type of the product"
  },
```

```
  {
    "attribute_id": 7,
    "name": "Product Name",
    "data_type": "SHORT_STRING",
    "description": "Human-readable identification of the
product"
  }
]
```


© 2026 ODVA, Inc.
All rights reserved.

Endpoint: Attribute Value

GET /cip/v1/objects/{class_id}/instances/{instance_id}/attributes/{attr_id}/value

```
{
  "data_type": "UINT",
  "value": "1"
}
```

Value is always a string — clients use data_type to parse

```
{
  "data_type": "STRUCT",
  "value": {
    "acd_activity": {
      "data_type": "USINT",
      "value": 2
    },
    "remote_mac": {
      "data_type": "ARRAY[6] USINT",
      "value": "001D9CC83A5B"
    },
    "arp_pdu": {
      "data_type": "ARRAY[28] USINT",
      "value":
"000108000604000 2001D9CC83A5B0A000101000000000000A000102"
    }
  ........
```


© 2026 ODVA, Inc.
All rights reserved.

Endpoint: Device Documentation

GET /cip/v1/documentation


```
[
  {
    "title": "User Manual",
    "description": "Complete user manual covering installation, configuration, and operation",
    "availability": "both",
    "device_url": "https://192.168.86.30/docs/user-manual.pdf",
    "external_url": "https://www.vendor.com/products/1756-L84ES/user-manual.pdf"
  },
  {
    "title": "Installation Guide",
    "description": "Quick-start installation and wiring guide",
    "availability": "external",
    "external_url": "https://www.vendor.com/products/1756-L84ES/installation-guide.pdf"
  },
  {
    "title": "EDS File",
    "description": "Electronic Data Sheet for use with CIP configuration tools",
    "availability": "device",
    "device_url": "https://192.168.86.30/docs/1756-L84ES.eds"
  }
]
```


© 2026 ODVA, Inc.
All rights reserved.

Endpoint: Device Documentation

GET	/cip/v1/documentation

Returns a list of device document entries. Each entry has: title, description, availability ("device" | "external" | "both"), device_url?, external_url?

Why this endpoint matters

- **No vendor website needed**
  - A technician next to a device can retrieve its full documentation index using a browser — no installed software, no credentials.
- **device_url vs external_url**
  - Documents can be hosted on the device itself, at an external vendor URL, or both. Clients pick the appropriate source.
- **EDS served directly**
  - An EDS file stored on the device is accessible at device_url — configuration tools can fetch it directly without manual search.
- **Browser-friendly**
  - Standard JSON response renders immediately in any browser — URLs in the response are directly clickable.

© 2026 ODVA, Inc.
All rights reserved.

Use Case 1: Device Discovery

A client issues GET /cip/v1/identity to each reachable device on the network — no CIP-specific tooling required.

```
curl http://192.168.86.30:8080/cip/v1/identity
```

```python
import requests


for i in range(1, 255):
    url = f"http://192.168.86.{i}:8080/cip/v1/identity"
    try:
        response = requests.get(url, timeout=1)
        if response.status_code == 200:
            identity = response.json()
            print(f"{url}: {identity['product_name']} ({identity['vendor_name']})")
    except requests.exceptions.RequestException:
        pass
```


© 2026 ODVA, Inc.
All rights reserved.

Use Case 1: Device Discovery

A client issues GET /cip/v1/identity to each reachable device on the network — no CIP-specific tooling required.

```
Sub GetDeviceIdentity(deviceNumber As Integer)
    Dim http As Object
    Dim json As String
    Dim ws As Worksheet

    Set ws = ThisWorkbook.Sheets(1)
    Set http = CreateObject("MSXML2.XMLHTTP")

    http.Open "GET", "http://192.168.86." & deviceNumber &
":8080/cip/v1/identity", False
    http.setRequestHeader "Cache-Control", "no-cache"
    http.setRequestHeader "Pragma", "no-cache"
    http.Send
```

```
      If http.Status = 200 Then
        json = http.responseText

        ' Extract data and write headers in column A, values
in column B
        ws.Cells(1, 1).Value = "Vendor ID"
        ws.Cells(1, 2).Value = GetJsonValue(json,
"vendor_id")
    Else
        MsgBox "Error: " & http.Status
    End If
End Sub
```

Three clients, one API — curl / Python / Excel VBA. No CIP stack required in any of them.

© 2026 ODVA, Inc.

All rights reserved.

Use Case 2: Feature Identification

Once a device is identified, GET /objects and GET /objects/instances determine what it implements — no EDS file, no vendor documentation.

1. **GET /objects**
   Immediate answer: a concise list of all supported CIP object classes with no prior knowledge of the device

2. **GET /objects/instances**
   Where more detail is needed: same list augmented with instance numbers — the complete object model in one request.

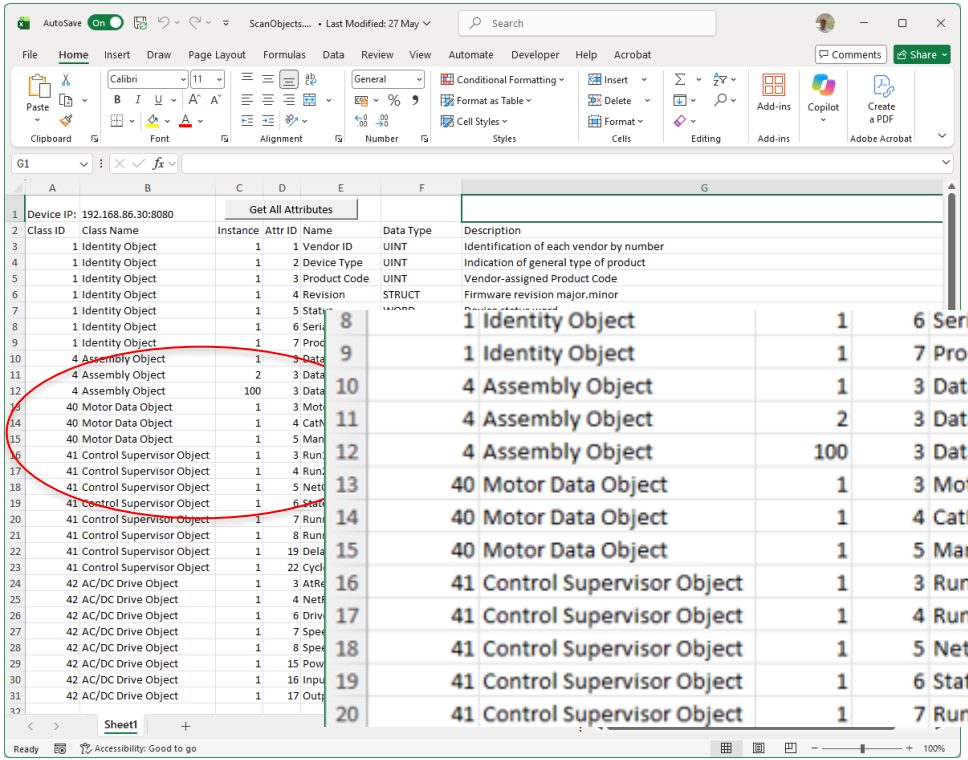
3. **Adapt behavior at runtime**
   Integration tools query the API directly and adapt based on the response — enabling runtime adaptability not possible with static EDS files.

4. **Write to Excel**
   The VBA subroutine from Use Case 1 extended to query these endpoints and write object classes + instances into the workbook.

Particularly valuable in heterogeneous environments — rather than consulting EDS files for each device, a tool queries the API and adapts at runtime.

© 2026 ODVA, Inc.
All rights reserved.

Use Case 2: Feature Identification



8	1 Identity Object	1	6	Serial Number	UDINT	Device serial number
9	1 Identity Object	1	7	Product Name	SHORT_STRING	Human-readable product name
10	4 Assembly Object	1	3	Data	USINT array	Assembly instance data
11	4 Assembly Object	2	3	Data	USINT array	Assembly instance data
12	4 Assembly Object	100	3	Data	USINT array	Assembly instance data
13	40 Motor Data Object	1	3	MotorType	USINT	Identifies the motor connected to the drive
14	40 Motor Data Object	1	4	CatNumber	SHORT_STRING	Manufacturers Motor Catalog Number
15	40 Motor Data Object	1	5	Manufacturer	SHORT_STRING	Manufacturers Name
16	41 Control Supervisor Object	1	3	Run1	BOOL	Run forward
17	41 Control Supervisor Object	1	4	Run2	BOOL	Run reverse
18	41 Control Supervisor Object	1	5	NetCtrl	BOOL	Requests Run Stop control to be local or from network
19	41 Control Supervisor Object	1	6	State	USINT	Current state
20	41 Control Supervisor Object	1	7	Running1	BOOL	Running forward

© 2026 ODVA, Inc.

All rights reserved.

Use Case 3: Large-Scale Data Harvesting

Traverse the API hierarchy: /objects/instances → /attributes → /value — harvesting live data from any device without CIP expertise.

What this example demonstrates

- **Objects 41 & 42**
  - 41 = Control Supervisor
  - 42 = AC/DC Drive Object
- **Z-score anomaly**
  - Rolling 20-sample window. Flag if current value > 2 std deviations from mean — e.g., unexpected voltage drop
- **No CIP code**
  - The entire script uses only standard Python requests library — no CIP stack, no drivers
- **Extensible pattern**
  - Same loop extended to multiple devices, fed into a time-series database, or connected to ML models

© 2026 ODVA, Inc.
All rights reserved.

Use Case 4: Engineering Tools & Digital Repositories

Standard HTTP actions — no vendor-specific plugins, no CIP knowledge — connect device data directly to the platforms engineers and managers already use.

Digital repository sync

Periodically query GET /identity for each device — update records automatically when revision, product name, or status change. No manual data entry.

Commissioning validation

Compare identity and object inventory from the API against an expected device profile. Flag discrepancies automatically without consulting EDS files.

Power Automate → Power BI

A Power Automate flow uses standard HTTP actions to query the device API, parses JSON responses, and feeds structured data into Power BI — no CIP tooling, no custom code.

OpenAPI → client code gen

Engineering tools generate client code directly from the OpenAPI definition — reducing integration effort and ensuring alignment with the API contract as it evolves.

© 2026 ODVA, Inc.
All rights reserved.

Results

6

Endpoints defined (identity, objects, instances, attributes, value, documentation)

4

Use cases demonstrated (curl, Python, VBA/Excel, Power Automate/BI)

0

CIP expertise needed by client applications

Key outcome

- ✓ Device discovery performed with a single curl command or short Python script — no drivers, no configuration tools, no CIP messaging
- ✓ Asset inventories populated automatically into Excel workbooks using VBA — no additional software beyond Excel
- ✓ Operational device data surfaced in Power BI via Power Automate — accessible to business controllers with no knowledge of CIP
- ✓ Live attribute data collected and analyzed for predictive maintenance using standard Python libraries only
- ✓ The same standardized interface serves clients from command-line tools to enterprise BI platforms, across any compliant device

© 2026 ODVA, Inc.
All rights reserved.

Comparison with Existing Approaches

Approach	Scope	Relationship to this paper
EDS files	Device configuration & parameters for tooling ecosystems	Complementary — EDS are static files; this API is dynamically queryable over HTTP. Devices can implement both.
Extended Logical segment (Majcher & Schaffner 2025)	Device metadata discoverable via CIP messaging infrastructure	Complementary — CIP-based; this API is HTTP-based and accessible to any client. Both can coexist.

© 2026 ODVA, Inc.
All rights reserved.

Impact on Stakeholder Workflows

Stakeholder	Before	After
System integrators	Manual inspection, proprietary tools, EDS parsing	Query any compliant device over HTTP — single script covers all vendors
Data scientists	Deep CIP expertise required; no systematic programmatic path	Traverse /objects/instances → /attributes → /value using standard HTTP; feed directly into analytics
Application developers	Vendor-specific documentation; no shared interface	Generate client code from the OpenAPI definition; target any compliant device
Business controllers	No access to live device data without CIP knowledge	Power BI reports refreshed via Power Automate — no industrial protocol knowledge needed
Maintenance engineers	Search vendor websites for documentation	GET /documentation returns links to manuals, EDS files, guides — directly from the device

© 2026 ODVA, Inc.
All rights reserved.

Conclusion

This paper has presented a proposed API based on the OpenAPI Specification for describing and accessing information from CIP-enabled devices.

- ✓ Standardized REST interface for device identity, object model, and live attribute values — accessible to any HTTP-capable client
- ✓ Device discovery with a single curl command; asset inventory via Python or VBA; dashboards via Power Automate + Power BI
- ✓ The same interface serves clients from command-line tools to enterprise BI platforms, across any vendor that implements the definition
- ✓ Complements EDS files and the Extended Logical segment rather than replacing them
- ✓ Grounded in OpenAPI — a rich ecosystem of tools for code generation, documentation, and validation comes with it

"This paper represents a first step in that direction — an invitation to the CIP community to engage with the proposed approach and contribute to its further development."

© 2026 ODVA, Inc.
All rights reserved.


2026
ODVA
Industry Conference and Annual Meeting