# GENERATED OPC UA PUBSUB STATUS PUBLISHING FOR UNICOS CPC

D. Grkovic, CERN, Geneva, Switzerland

*Abstract*

During my internship I worked on OPC UA PubSub status communication for UNICOS CPC. The work covered the status data used by 20 device families, a comparison of UADP encoding options, and an on-change publisher added to the UNICOS CPC resources-package (RP_CPC) generator. From the same UNICOS model, the generator produces RawData layouts, one DataSetWriter per device, lookup tables for devices and groups, WriterGroups, PLC structures, and receiver metadata. Supporting work included PLC scan-cycle benchmarking, TIA Portal validation, and packet decoding. Validation confirmed that the implementation compiled cleanly in TIA Portal and delivered decodable traffic to an independent computer used to capture and analyse the received packets. The contribution was submitted as an RP_CPC merge request.

## INTRODUCTION

UNICOS CPC is CERN's framework for building industrial control applications with programmable logic controllers (PLCs) and SCADA systems. RP_CPC is its resources package: it contains CPC device definitions and generation templates used by the UNICOS Application Builder (UAB) to produce PLC code and SCADA configuration for industrial control projects. UNICOS CPC currently supports communication paths including Modbus, S7, and S7Plus. OPC UA PubSub lets a PLC publish UADP messages over UDP without a client polling each PLC [1, 3]. My project investigated how this mechanism could be generated for UNICOS applications while respecting the memory and packet-size limits of an S7-1500 PLC and enabling repeatable PLC scan-cycle benchmarking.

I started from an existing PubSub prototype and the Siemens LOpcUa library [4], then implemented the PubSub path in the existing UNICOS and RP_CPC codebase. My contribution covered the generated data contract, PLC runtime code, and the tests used to examine its behaviour.

## FROM DEVICE MODEL TO GENERATED PUBLISHER

### *Status mapping and encoding choice*

The first step was to map the binary and analogue status fields used by 20 UNICOS device families. The layouts mainly contain WORD and REAL values, with individual payloads ranging from 2 to 64 bytes. This mapping supported a comparison between self-describing Variant fields and schema-defined RawData. Variant encoding carries type information with each value and is convenient for a generic receiver. For stable, generated PLC datasets, however, that repeated metadata increases the payload and PLC representation size and requires per-field encoding logic. The selected design therefore uses fixed-order RawData layouts with matching generated JSON metadata for receivers.

The design study also compared per-device DataSetWriters with grouping several devices into a single dataset. The selected model keeps state and lookup information at two levels: each device has its own DataSetWriter and dirty flag, while each WriterGroup records the group of compatible devices that can be sent together. This makes it possible to find a changed device directly and then locate the group that must be scheduled. One UADP NetworkMessage can therefore carry the eligible datasets from that group without exceeding the configured packet limit.

Under the shared-baseline packet model used in this study, $N$ changed 10-byte DataSetWriters are batched into one message. The resulting RawData message occupies $20 + 15N$ bytes, whereas the evaluated DeltaFrame form occupies $26 + 19N$ bytes. Figure 1 shows the calculated trend under that batching assumption. It is a model comparison, not a network measurement, but it made the repeated cost of field indexes and related metadata visible before PLC implementation.

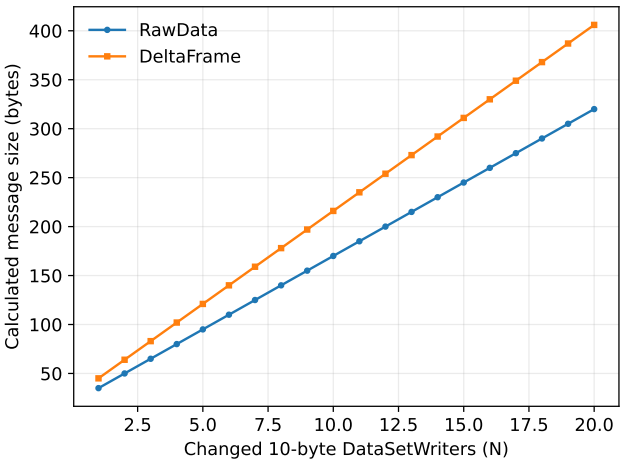


Figure 1: Calculated UADP message size for the RawData and DeltaFrame models evaluated in the project.

### *Generator and runtime integration*

The RP_CPC generator reads the UNICOS application model and `s7plus_ids.json`. The PubSub contribution uses those inputs to generate dataset layouts and identifiers, pack writers into compatible groups, create SCL data structures and payload encoders, insert calls into the required organisation blocks, and emit JSON contracts for decoding or subscriber configuration. Generating the PLC artifacts and receiver metadata from one model reduces the chance of layout drift. The generator keeps normal and fail-safe (FI) device layouts separate and preserves fail-safe device

execution while the PubSub branch replaces only the legacy status transport calls.

Figure 2 shows the schema-derived generator outputs. TiaCli was used with the TIA Portal Web Server to import and compile the generated project.

OB1 compares current status values with their previous state and marks the corresponding DataSetWriter dirty. As shown in Figure 3, the runtime keeps a flat per-device table for direct lookup and a WriterGroup table that links each group to its member devices. A cyclic communication interrupt walks through the groups in round-robin order and submits at most one eligible group through TUSEND. The device table does not store a second payload copy; the cyclic interrupt reads the current status data for the devices in the selected group when it packs the message.

UDP does not acknowledge delivery to a subscriber. A periodic `RequestAll` operation therefore marks all writers again so that later datagrams can restore the receiver's current state after an undetected loss. It does not recover a local transport fault. During development, a TUSEND error deliberately stops publication until the enable signal is cycled. Keeping the fault latched makes it easier to reproduce and diagnose the failing condition. A production implementation will add automatic recovery steps, including controlled retry and communication-state reinitialisation. TUSEND `DONE` means that the local PLC send job completed; it is not proof that a subscriber received or processed the packet.

### *Decision path*

I compared UDP with MQTT for status publishing. MQTT offers brokered distribution and persistent sessions, but it also requires a broker and uses a different recovery model. For the local S7-1500 case studied here, UADP over UDP matched the available implementation and avoided that extra component.

I also considered DeltaFrame messages and a parallel table of changed fields. That design can express a subset of a larger dataset, but it needs field-index metadata and more runtime bookkeeping. Under the packet model used in the study, one RawData DataSetWriter per device had lower wire overhead for the tested change counts and matched the existing UNICOS device status layouts. The DeltaFrame designs remain part of the design history; the generated publisher uses fixed-layout RawData DataKeyFrame messages.

To size realistic and deliberately large tests, I analysed 706 cooling and ventilation specifications from CERN's SVN repository, containing 442,403 variables. The median specification contained 488 variables and the 75th percentile was 810. Figure 4 shows the diversity in total application size, while Figure 5 shows counts per device family where that family is present. DigitalAlarm has the highest median (134), followed by DigitalInput (78) and AnalogParameter (71.5). I used the observed distribution of application sizes and device-family counts to select representative and high-count configurations for the integration and stress tests [2].

### *Generated contract and handover*

Before UAB integration, I built a standalone JSON-configurable generator and scripts for faster iteration. From shared configuration it produced Siemens SCL and receiver metadata, plus packet-size and validation reports and protocol documentation. A publishing command automatically uploaded the reports and documentation to Confluence. I then moved the production design into RP_CPC. The integrated generator produces the communication SCL, runtime data types, OB1 change-detection calls, integration of the cyclic interrupt, and three JSON files for decoding and subscriber configuration. Configuration parameters control activation, addressing, publication timing, the periodic full marking interval, and packet limits. The project limits each generated UADP NetworkMessage to 1,400 bytes. This is a conservative application limit that keeps the UDP payload below the 1,472 bytes available within a standard 1,500-byte Ethernet MTU after the IPv4 and UDP headers are added. It is not the S7-1500 TUSEND ceiling: Siemens documents lengths up to 2,048 bytes for S7-1500 CPUs from firmware V2.5 when using unicast or multicast, although messages above 1,472 bytes require IP fragmentation on a standard Ethernet network [4]. The allocated PLC-side buffer is therefore 2,048 bytes, while generated messages remain capped at 1,400 bytes.

I organised the Confluence documentation into current implementation pages and earlier design studies. It records the data structures, packet-size formulae, scheduling and error behaviour, generated files, and source references. Each result is identified as a measurement, calculation, or expected value. For the stress test, I recorded whether the evidence came from PLC timing, the packet capture, or the independent subscriber, making the scope of each result clear.

## SUPPORTING ENGINEERING TOOLS

I also developed supporting tools for PLC timing, packet inspection, TIA validation, and preparation of test specifications.

### *S7 measurement and packet decoding*

As a separate tooling project, I developed the UNICOS S7 Benchmark VS Code extension to make PLC execution measurements easier to configure and repeat. The user selects statements in generated UNICOS SCL, and the extension inserts timing calls and generates the shared `UNICOS_Benchmarking.SCL` source. It stores the benchmark definitions in the application workspace and can restore the measurement points after UNICOS regenerates the SCL. The extension records either windowed statistics or duration changes, verifies and archives completed PLC DataLogs, and provides plots, selected-range statistics, threshold indicators, and CSV export. Keeping the definition, settings, raw data, and analysis together made reruns easier to reproduce and compare.

In the 5,000-writer stress-test application, three 100-sample status windows had mean execution times from 23.1

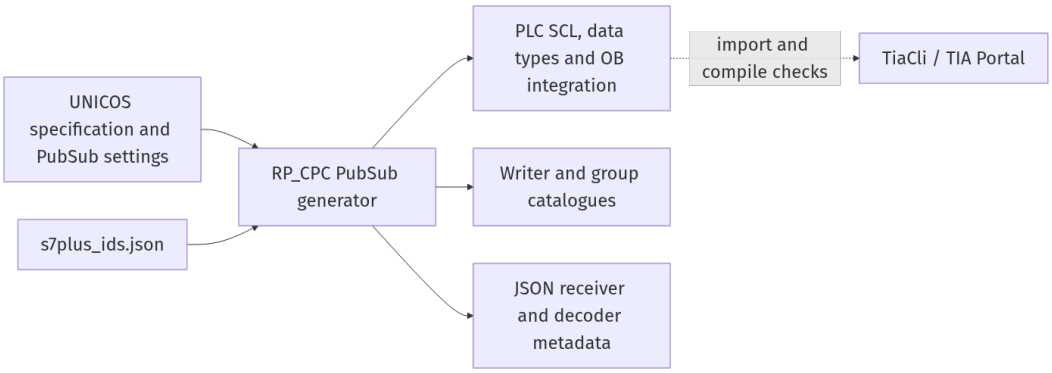


Figure 2: Schema-derived PubSub generation with development-time TiaCli checks using the TIA Portal Web Server.

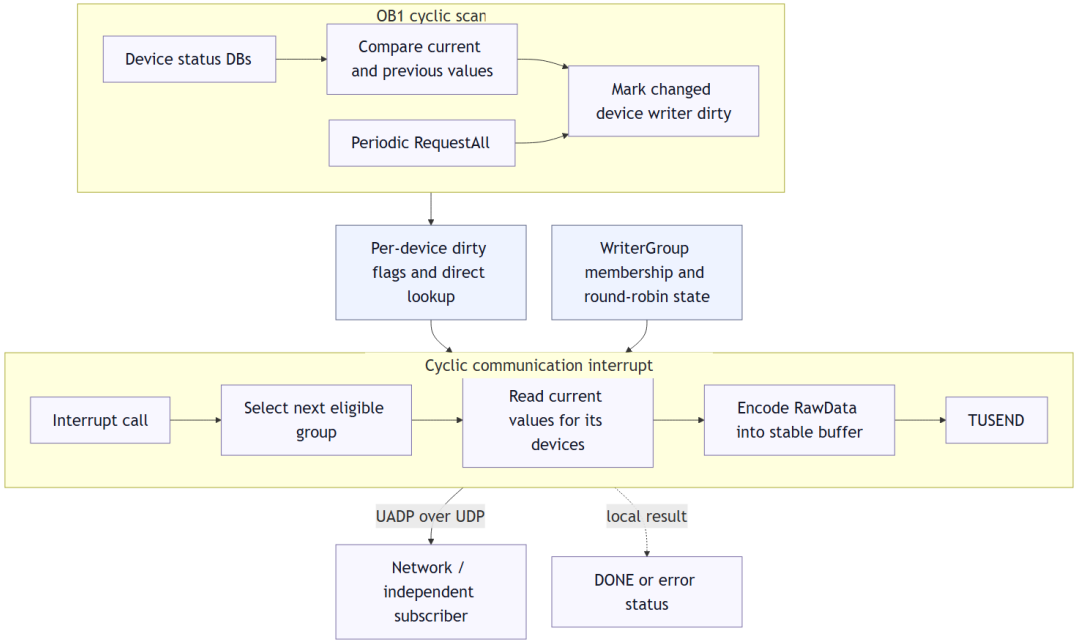


Figure 3: PLC runtime path. OB1 marks changed devices, while a periodic cyclic communication interrupt selects one eligible WriterGroup, packs its current values, and calls TUSEND.

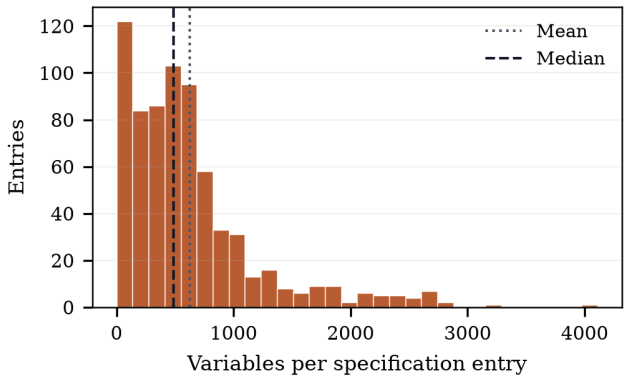


Figure 4: Distribution of total variables across 706 cooling and ventilation specifications.

to 26.2 ms, with a maximum of 29.7 ms. Sixteen 100-sample send windows had means from 1.18 to 1.37 ms, with a maximum of about 1.62 ms. These are PLC execution timings at the instrumented points; network delivery was assessed separately.

RawData also required decoder metadata because the field types are not repeated in every packet. I adapted an existing Wireshark and Tshark OPC UA dissector fork to load the generated JSON description, identify WriterGroups and DataSetWriters, and decode their fixed field layouts. This turned packet captures into named status values that could be compared with PLC counters.

## *TIA Portal automation*

I used the CERN TiaCli tool in scripts that create or open a TIA project, generate and import sources, check integration of the cyclic interrupt, and compile the PLC software through the TIA Portal Web Server. This made large cases repeatable and helped distinguish generator faults from earlier workbook or device-model errors.

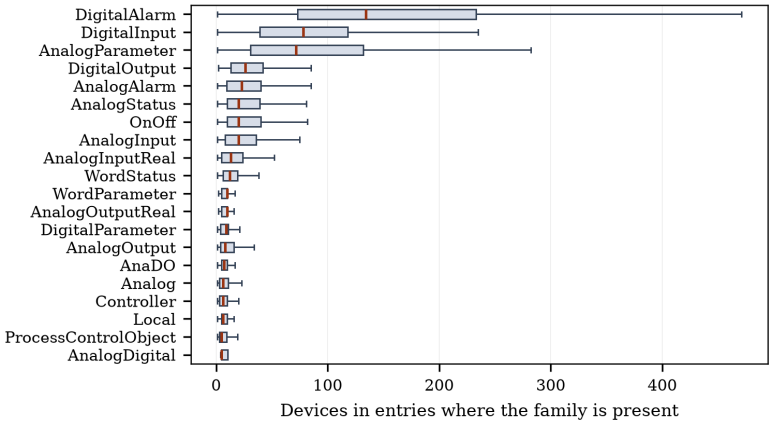


Figure 5: Device counts across specification entries in which each family is present: medians, middle 50%, and 1.5-IQR whiskers; individual outliers are omitted.

### *Separate MCP side project*

Separately from the PubSub implementation, I developed *MCP UNICOS CPC Spec*, a TypeScript MCP server and plugin for UNICOS and UCPC specification workbooks [5]. I adapted Stefan Broenner's Excel MCP and added UNICOS field documentation, `SpecDoc` and FE-encoding support, validation, object-level operations, and packages for Codex and Claude Code. I used it to prepare test specifications with varied device configurations and edge cases, then exercised the UNICOS integration and PubSub implementation with those variants. Read-only inspection parses OOXML without Excel; edits are applied to a temporary copy and checked before acceptance. This remained a separate tooling project.

## MEASUREMENT AND VALIDATION

Validation covered the complete generated path on the S7-1500 selected for the project. The generated sources completed a clean TIA Portal compile. Generation was also checked with an input of 5,001 devices, which produced 71 WriterGroups. Network traffic was received and decoded on a separate computer, providing an observation point independent from the PLC. After integration and testing, I submitted the contribution as a merge request to RP_CPC.

The stress test used a generated application with 5,000 test writers: 1,000 analogue inputs, 1,000 analogue outputs, 2,000 analogue input real objects, and 1,000 analogue output real objects. Approximately 30 seconds of traffic were captured at cyclic-interrupt periods of 10, 4, and 2 ms. Figures 6 and 7 separate the state-coverage result from the timing result so that their different scales are not visually conflated.

For the 2 ms run, every counter transition between the first and last captured sample of each writer was present in the decoded capture. The omission percentages in Figure 6 apply only between those boundaries. At 10, 4, and 2 ms, the captures contained 156,781, 408,635, and 821,803 received states, while the boundary-based analysis counted 1,245,316, 790,897, and zero omitted intermediate states. Figure 7 compares the corresponding PLC mean cycle values of 104.78, 122.99, and 182.87 with packet cadences of

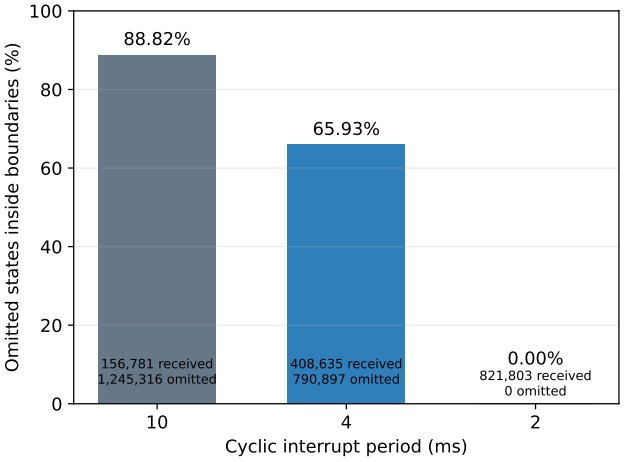


Figure 6: Boundary-limited counter-state omission in each capture. The percentages are not UDP packet-loss rates.

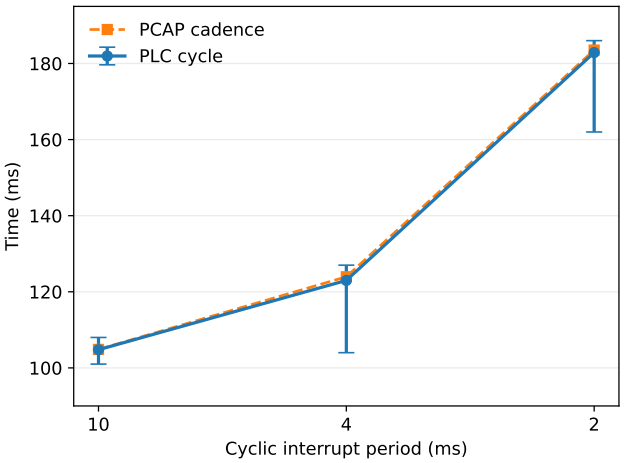


Figure 7: PLC mean cycle with observed extrema and packet cadence calculated from the capture.

104.86, 123.95, and 183.52. The two measurements agree within about 1 ms.

Only one run was made for each setting. The PLC cycle increased as cyclic interrupts became more frequent, but it remained within reasonable limits for the purpose of this

test. The case was intentionally severe: all 5,000 test writers were marked as changed together, whereas regular operation would not be expected to produce simultaneous changes across an application of that size. The results therefore describe stress behaviour rather than a typical operating load. The independent computer used as a subscriber captured the UADP packets with Wireshark. I used the generated metadata to decode the fixed field layouts and post-processed the captured values to check the PLC counters. This covered the full PLC-to-computer network path. Within this setup, shorter cyclic interrupts reduced the observed omissions at the cost of a longer PLC cycle.

The runtime also deliberately coalesces repeated changes. If a value changes several times before its writer is sent, only the current value is packed. A short transition that changes and returns between two OB1 scans can be missed. The design therefore publishes current state, not a complete event history. Whether that behaviour is acceptable depends on the process requirements.

## WORKING METHOD AND LESSONS LEARNED

The main technical lesson for me was that a compact wire format cannot be designed separately from the application model and PLC scheduler. RawData reduced repeated information in each message, but it also transferred responsibility to the generator and receiver metadata. A field-order error would still produce valid bytes on the network while giving them the wrong meaning at the receiver. For that reason, I treated the generated SCL and JSON files as one contract and kept their identifiers and layouts derived from the same input. The device and WriterGroup lookup tables serve the same purpose inside the PLC: they make the relationship between a changed device and the group to be transmitted explicit rather than reconstructed during every interrupt.

I also learned to separate the available observation points. A PLC TUSEND result, a sequence in a packet capture, and reception at a subscriber describe different parts of the system. I therefore recorded each measurement with its source and kept PLC timing, capture analysis, and subscriber results separate.

The work crossed Python, Siemens SCL, TIA Portal, packet captures, TypeScript, and documentation. I organised Confluence to separate current behaviour from measurements and earlier experiments, then prepared handover material. My aim was practical: another engineer should be able to find the design, repeat a calculation, and trace the merge-request evidence.

## CONCLUSION

I added a generated PubSub publisher to RP_CPC. One device model now defines the PLC structures and receiver metadata, with change detection in OB1 and communication in a cyclic interrupt. Supporting tools made PLC measurement, packet decoding, generation, and compile checks repeatable. Stress testing showed that shorter cyclic interrupts reduced boundary-limited omissions while increasing the PLC cycle.

I submitted the implementation and supporting tests as a merge request to RP_CPC. The Confluence pages link the generated contract, runtime behaviour, measurements, and test evidence used for that submission [2].

## ACKNOWLEDGEMENTS

I would like to thank my supervisors, Loreto Gutiérrez Prendes and Brad Schofield, for their guidance and technical feedback throughout the project.

## REFERENCES

[1] OPC Foundation, *OPC Unified Architecture Specification, Part 14: PubSub*.

[2] CERN Industrial Controls, “OPC UA PubSub,” UNICOS CPC engineering documentation, 2026.

[3] L. Gutierrez Prendes *et al.*, “Enabling high-performance PLC communication through open standards: OPC UA PubSub,” in *Proc. ICALEPCS’25*, Chicago, USA, 2025, pp. 1117–1121.

[4] Siemens AG, *Library for OPC UA PubSub for SIMATIC S7-1500*; *Examples of Open User Communication: UDP*, Entry ID 109747710, 2019.

[5] D. Grkovic, *MCP UNICOS CPC Spec*, CERN, 2026.