MIL-STD-3071 REFERENCE IMPLEMENTATION

Nexus TMS

Build a compliant Tactical Microgrid device **without first becoming a DDS expert.**

generator.cpp

```
nexus_tms::device_config cfg;
cfg.device_id     = "SRC-24VDC-0001";
cfg.role          = tms::ROLE_SOURCE;   // a generator
cfg.strength_file = "generator.strength";
nexus_tms::device gen;
gen.set_power_ports(my_ports);
gen.set_start_stop(my_level);
gen.start(cfg, my_device_info);   // announce to the grid
while (running)
{
    gen.poll();   // the standard, handled for you
    nexus_tms::start_stop_order order;
    if (gen.take_start_stop_order(order))
    {
        drive_relay(order.to_level);   // your hardware
        gen.set_start_stop(my_new_level);
    }
}
gen.stop();
```

A TMS GENERATOR The library handles the network. Your code drives the hardware.

Picture a forward operating base. The generators come from one vendor, the batteries from another, the solar array from a third, and the controller from a fourth. Making them work as one grid has meant custom integration: slow, expensive, and fragile.

MIL-STD-3071, the Tactical Microgrid Standard (TMS), gives every device one shared language over DDS. But to build a TMS device today, you first have to learn DDS, then implement hundreds of pages of rules on timing, rate limits, controller failover, and request handling.

Nexus TMS takes both off your plate

The protocol is done for you

Identity, heartbeats, controller selection and failover, request validation and replies, and the standard's rate limits.

The real standard, unmodified

Types, topics, and QoS come straight from the files that ship with MIL-STD-3071. Nothing is retyped by hand.

Built on the Nexus API

A data-centric DDS API that vendors new to DDS can learn quickly.

Standard DDS on the wire

TMS compliance is defined by the network, and Nexus TMS speaks standard RTPS. Interop testing with OpenDDS is milestone M3.

What one call to `poll()` does for you

Sends a heartbeat every second	§B.4
Tracks every controller and picks the one to obey, with the 3, 6, and 10 s timers	§A.7.5
Refuses invalid requests with the correct reply code	§B.2, §B.16
Replies within 0.1 seconds	§B.2.1
Holds every state topic to its rate class and burst limit	§A.8.5

Every requirement in the code cites the section of the standard it implements, so you can check it yourself.

AT A GLANCE

Standard
MIL-STD-3071 with Change 2 (draft of 2026-07-14), data model 2.0.3

Coverage
103 structs, 33 enums, and all 37 topics from the standard's own files

First roles
Starting with generator, energy storage, load, and microgrid controller

Built on
Nexus API and Nexus DDS (Community Edition is free)

Platforms
Linux, macOS, and Raspberry Pi OS; C++17 and CMake

Simulation
A whole grid runs on one laptop, no hardware needed

Availability
Free and open, released on GitHub and federateddesigns.com at milestone M5

NOT YET INCLUDED

DDS Security will follow the MIL-STD-3071 cybersecurity revision now in progress. Roles and per-role topic permissions are already explicit, so it can be added without a redesign.

The standard, as written. Built in the open.

Driven by the standard's own files

FILE SHIPPED WITH MIL-STD-3071	WHAT NEXUS TMS TAKES FROM IT
`mil-std-3071_data_model.idl`	Every TMS type, generated into C++ by `nexus_gen` at build time
`mil-std-3071_topics.csv`	All 37 topics: publishers, subscribers, triggers, rates, and QoS
`mil-std-3071_qos_library.xml`	The standard's QoS profiles
`mil-std-3071_data_flows.csv`	The topics that make up each request and reply exchange

The files are checksummed and never edited. If a tool can't handle the standard, the tool gets fixed.

Why the Nexus API

TMS compliance is about what goes over the network, not which API you program against. The Nexus API lets a vendor say what kind of data each topic carries:

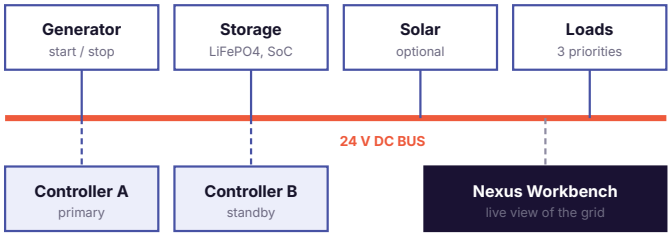
INTENT	TMS EXAMPLES
`state`	DeviceInfo, PowerPortState, Heartbeat
`event`	Start/stop requests, replies
`stream`	Fast measurements

8 → 1 The standard's eight QoS profiles become one 20-line function covering all 37 topics.

19 vs 49 Lines to send and read a TMS Heartbeat: Nexus API vs. the classic OMG DDS API.

The tabletop microgrid

A safe, 24 V DC demonstration grid you can plug in, unplug, and break on purpose, coming with milestone M4. The protocol doesn't care whether the bus is 24 V or 480 V.



Each box is a Raspberry Pi running one TMS role. TMS traffic runs over Ethernet as standard DDS.

1. **Plug and play.** Plug in a generator; the grid finds it.
2. **Controller failover.** Unplug the active controller; the standby takes over.
3. **Battery runs low.** The controller starts the generator.
4. **Overload.** The lowest-priority load is shed.
5. **Mixed vendors.** Swap in another vendor's TMS stack; the grid doesn't notice.

Where we are

STAGE	SCOPE	STATUS
M0	Standard and IDL in the repo, types generated, QoS from the topic table	Done
M1	A generator with every topic the standard requires, plus a test controller	Nearly complete
M2	Microgrid controller with primary and standby failover	Next
M3	Interoperability with the OpenDDS TMS example, both directions	Planned
M4	The tabletop grid running scenarios 1 to 4	Planned
M5	Public release with a getting-started guide for people new to DDS	Planned

Build the device.
We'll handle the standard.

TALK TO US

www.federateddesigns.com
eva@federateddesigns.com

© 2026 Federated Designs. All product names and trademarks are the property of their respective owners.