DS-015 UAVCAN Drone Standard

Revision: 1.0.1
Revision date: Feb 23, 2021

Abstract

This document is the formal version of the UAVCAN Drone Standard developed by the Dronecode Foundation UAVCAN Drone Special Interest Group.

Official Drone Profile of

UAVCAN

Table of contents

Document revisions

Revision	Editor	Reviewer	Comments
0.0.1	Nuno Marques	Lorenz Meier	Initial specification
0.2.0	Nuno Marques	Lorenz Meier	Detailed specification for initial message set.
0.3.0	Nuno Marques	Lorenz Meier	Reduced message set with alpha level specification
0.4.0	Nuno Marques	Lorenz Meier	Define port ID ranges and mechanisms
0.5.0	Nuno Marques	Lorenz Meier	Add physical layer recommendations
0.6.0	Pavel Kirienko	Nuno Marques	Move the data types to the public_regulated_data_types repository.
0.7.0	Pavel Kirienko	Nuno Marques	Restructure the document, add introduction, define application-level services, add implementation suggestions, add diagrams, remove backlogged content.
0.8.0	Pavel Kirienko	Nuno Marques	Finalize sections “Profiles” and “Topology”, add section “Conformance”, add examples and elaborations throughout. Move changes to the public document.
1.0	Pavel Kirienko	Nuno Marques	Official v1.0
1.0.1	Pavel Kirienko		Correct references to externally defined entities. Copy-paste network service definitions from the DSDL repository to enhance discoverability (as discussed at the UAVCAN SIG call on Jan 26).

Contact and public developer call

For further questions regarding the specification, please contact the maintainers: lorenz@px4.io, nuno@auterion.com and pavel@uavcan.org, or raise issues on the forum. This standard is being developed by the UAVCAN Drone Standard Special Interest Group, hosted by the Dronecode Foundation.

Trademark guideline

UAVCAN is a registered trademark for the transport and application layer protocols. This standard message set is endorsed by UAVCAN, however, this special interest group and its members do not represent UAVCAN as a protocol design body, but only this message set. This message set is an approved standard of the Dronecode Foundation.

License and disclaimer

Redistribution and use in products, without modification, are permitted provided that the following conditions are met:

- Implementations of the standard must be compliant with the full specification.
- A royalty-free, non-exclusive license is provided to adopters with a valid adopter agreement for schematics and drawings based on the standard documentation.

Terminology and acronyms

- **UAVCAN** – Uncomplicated Application-layer Vehicular Computing And Networking (standard).
- **RPC** – Remote Procedure Call.
- **Network service** – application-layer functionality whose behavioral contracts are defined in terms of UAVCAN interaction primitives. Not to be confused with *RPC*.
- **UAVCAN subject** – a category of messages exchanged between the participants of a distributed intravehicular computing system pertaining to the same application-layer function or process.
- **Node** – participant of a distributed intravehicular computing system.
- **COTS** – Commercial Off-The-Shelf (equipment).
- **ESC** – Electronic Speed Controller (of an electric motor).

Introduction

This standard defines the standardized application layer for drones and the suitable physical connectivity layer optimized for unmanned vehicles implementing complex behaviors while ensuring a high degree of functional safety. The intermediate network abstraction layers are addressed by the UAVCAN networking technology that this standard is based upon. Therefore, to understand DS-015, one needs to understand what UAVCAN is.

UAVCAN stands for *Uncomplicated Application-layer Vehicular Computing And Networking*. It provides publish-subscribe and RPC (*remote procedure call*) interactions for real-time intravehicular distributed systems, similar to DDS or ROS, but with a reduced capability set to manage the complexity of implementation, verification, and validation.

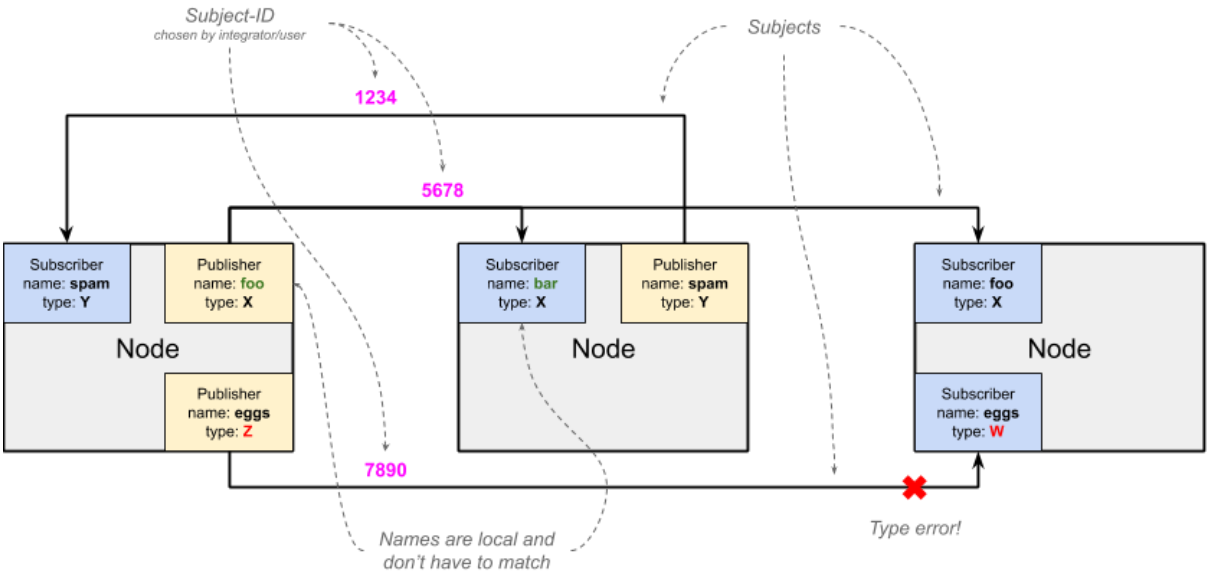
Decentralized anonymous publish-subscribe is the main interaction model where the data links are organized into *subjects*. Subjects are functionally similar to topics found in most other publish-subscribe technologies with the critical difference being that instead of a textual name, subjects are identified by their numeric identifier — a *subject-ID*. The subject-ID is chosen by the network integrator (the engineer responsible for constructing the distributed system), excepting some special circumstances where it is rigidly pre-defined by some other applicable standard (this principle is similar to well-known TCP/UDP port numbers).

A network participant (called *node*) may publish and/or subscribe to an arbitrary number of subjects provided that every node that joins a subject uses a compatible data type for serializing and deserializing data objects exchanged over that subject. The data types are defined using a domain-specific language called DSDL — *Data Structure Description Language*. A large part of this standard is dedicated to defining DSDL types and the related *network services*. DSDL data type definitions are organized into *namespaces* much like classes in an object-oriented programming language.

Despite the fact that subjects are identified by their numeric identifiers rather than names, they may still be named in order to differentiate between different subjects published by or subscribed to by a given node. Since this distinction is confined to the same network participant, the names of the subject bear no relevance for the runtime performance of the network and it is possible that multiple nodes connected to the same subject name it differently. For example, node A may publish the setpoint for a control loop over some subject X, while node B responsible for effecting the control output would subscribe to the same subject; in this example, node A may name the subject **`output_setpoint`** while node B may name it **`command`**. As long as the involved nodes agree on the subject-ID value of X and the data types used for this subject are compatible, they will be able to communicate successfully despite the fact that *internally* they refer to the same subject under different names.

Nodes are configured using the *register API* — an application-layer capability that offers a trivial strongly typed key-value storage per node used to set and read that node's configuration options. The register API is accessed using the RPC functionality mentioned earlier. Fundamentally, RPC is like publish-subscribe with the key difference being that every interaction involves bidirectional data exchange that is not anonymous. The common term covering both subjects and services is *port*; similarly, there is *port-ID*.

The key principles explained above are illustrated in the following diagram. While this introduction is not expected to be an exhaustive description of UAVCAN and is not expected to provide sufficient knowledge for implementing a compliant product, it should be sufficient to serve as a minimal primer. Readers wishing to learn more should turn to **The UAVCAN Guide** and the **UAVCAN Specification**.



Contrary to some prior art, this standard makes heavy emphasis on service-oriented design of its network services as a response to the growing complexity and safety requirements of the addressed vehicular applications. This is covered in chapter **Application layer**.

Even though UAVCAN is capable of supporting different underlying transport layer protocols and it shields the upper protocol layers from the specifics of the underlying transport, this standard is focused primarily on UAVCAN/CAN – a transport defined on top of Classic CAN and CAN FD protocols. The only part of this standard that is not transport-agnostic is the chapter **Physical layer** and the related conventions and recommendations. Future revisions may be extended to cover other transports depending on the demands of the addressed vehicular applications.

The first few chapters are the normative part of this specification that defines the abstract requirements to conformant implementations. To aid understanding, they are followed by various predefined options and examples provided in chapter **Profiles**.

The final chapter **Conformance** addresses issues pertaining to conformance testing.

Protocol layers and standards

Per the OSI model, this standard covers the layers 1 and 7. The OSI layer mapping is provided in the left column for reference.

L7	**UAVCAN Standard Application Layer** (defined in UAVCAN Specification)	**DS-015 Drone Application Layer** (this standard)	Optional vendor-specific application layer services
L5~L6	**Presentation Layer** (defined in UAVCAN Specification)		
L2~L4	**Transport Layer** (defined in UAVCAN Specification)		
L1	**DS-015 Drone Physical Layer** (this standard)		

Application layer

The application layer is defined in two parts:

- The standard application layer is defined by the UAVCAN Specification. Being highly generic, the network services and data types defined there are common for all UAVCAN-capable systems and components regardless of their application domain.
- The Drone Application Layer, being part of this standard, is defined in the regulated namespace **`reg.drone`** maintained by the UAVCAN project at **github.com/UAVCAN/public_regulated_data_types**.

Additionally, implementers may extend the application layer with custom network services and types. Relevant information can be gathered from **The UAVCAN Guide**.

Presentation layer

The presentation layer is responsible for data representation and fundamental network interaction primitives such as publish-subscribe links and remote procedure calls. This layer is based on the UAVCAN Specification, particularly on DSDL – a domain-specific Data Structure Description Language.

Transport layer

The transport layer is responsible for transferring blocks of binary data between network participants, including its decomposition, reassembly, prioritization, routing, timing management, etc. This layer is based on the UAVCAN Specification.

Physical layer

The physical layer is responsible for electromechanical compatibility of the network participants. It is defined in the Physical layer chapter of this document with references to the applicable third-party standards.

Application layer

UAVCAN standard application layer

UAVCAN defines certain functionality and conventions that are expected to be useful in any vehicular application regardless of its domain. Instead of re-defining its own alternatives, DS-015 relies on such standard functionality offered by UAVCAN. Most of the functions covered in this section are optional per the underlying UAVCAN standard, but mandatory per this DS-015 standard.

Despite the fact that UAVCAN itself implements a flat peer network that is devoid of centralized responsibilities, in the context of the application layer it may be convenient to define a special class of nodes that are responsible for auto-configuring other nodes in the network, barring manual intervention by a human. A typical example of such a node would be the mission computer or a flight management unit. In this document, such nodes are referred to as *autoconfiguration authority* nodes.

Conventions

Design conventions, physical notation conventions, and other conventions specified in section **5.2 Application-level conventions** of the UAVCAN Specification shall be followed.

Conformant implementations are not allowed to use unregulated fixed port identifiers.

Heartbeat

The UAVCAN Specification requires that every node shall publish a heartbeat message at least once per second. This requirement is therefore redundant but is listed here for completeness.

Introspection

The following network services, optional per the UAVCAN Specification, are mandatory to support:

- Node introspection service `uavcan.node.GetInfo`.

Register API

The register API and the standard registers are defined in `uavcan.register.Access`.

The following standard registers shall be supported:

- **`uavcan.node.id`** – 65535 (unconfigured/PnP) by default.
- **`uavcan.node.description`** – empty by default.
- **`uavcan.(pub|sub|cln|srv).PORT_NAME.id`** – 65535 (unconfigured/disabled) by default.
- **`uavcan.(pub|sub|cln|srv).PORT_NAME.type`** – constant, set by vendor.

Where ***PORT_NAME*** stands for the name of a publisher, subscriber, server, or client.

A **`uavcan.(pub|sub|cln|srv)`** register set shall be provided for every port for which there is no fixed port-ID. For example, suppose that a node provides electric power estimates of type **`reg.drone.physics.electricity.PowerTs.1.0`** over some subject that is referred to as **`electric_power`** in its documentation. Then, per the requirements, it would have the following registers (among others):

- **`uavcan.pub.electric_power.id`** of type **`natural16[1]`**, persistent and mutable, that the integrator will use to link the subject with the subscribers.
- **`uavcan.pub.electric_power.type`** of type **`string`**, persistent and immutable, that contains a constant string **`reg.drone.physics.electricity.PowerTs.1.0`** defined by the vendor.

Plug-and-play

The plug-and-play (PnP) capability is an optional functionality that allows new nodes to auto-configure their node-ID upon connection to the network, provided that there is a *plug-and-play allocator* available online. Such auto-configured nodes are said to implement the *plug-and-play allocatee* (sic!) behaviors. The DS-015 standard assumes that a PnP allocator is available on the network.

An autoconfiguration authority node would typically implement the PnP allocator logic, being the configuration authority that grants node-IDs to other network participants. Barring that, the node shall implement the PnP allocatee logic, while also supporting manual assignment of the static port-ID.

A Python implementation available in PyUAVCAN can be consulted with.

Bootloader

Autoconfiguration authority nodes are exempted from these requirements. Other nodes shall implement the standard UAVCAN bootloader logic, which involves supporting the following services:

- uavcan.node.ExecuteCommand (server) — specifically, at least the following commands shall be supported both in the bootloader mode and in the application mode:
  - `COMMAND_RESTART`
  - `COMMAND_BEGIN_SOFTWARE_UPDATE`
- `uavcan.file.Read` (client) — the file read service is used for downloading the firmware image in the bootloader mode.

While the software update is in progress, it is recommended to emit diagnostic messages of type `uavcan.diagnostic.Record` periodically to keep external observers informed about the status of the update process.

Drone application layer

The drone application layer service definitions are the core piece of this standard. They are specified in the DSDL notation in the namespace **`reg.drone`** stored in the official public regulated data type repository maintained by the UAVCAN team ("reg" is short for "regulated"). Please refer to the enclosed documentation for technical details.

Domain-specific services

Please consult with the above-linked **`reg.drone`** namespace for the formal domain-specific service definitions. Links to *some* of the domain-specific service definitions are given below; if the links have become invalid, please navigate from the root of the repository manually. The list given here may not be exhaustive and the text may become obsolete. In case of any divergence, definitions given in the upstream DSDL repository take precedence.

Air data computer

```
# A generic air data computer service.
#
# An air data computer service is generally a non-interactive publish-only service: when activated,
# it keeps publishing its measurements at a fixed rate over several subjects until deactivated.
# The activation/deactivation, if supported, is managed via the standard readiness control service.
# The data update rates and other parameters are controlled via the Register API using implementation-defined
# register names.
#
# An air data computer whose readiness state is SLEEP is allowed to cease all network activity.
#
# An air data computer whose readiness state is STANDBY is recommended to behave as if its state was ENGAGED;
# implementations are recommended to leverage this state to perform any necessary maintenance activities
# such as calibration, self-heating, etc. The service should not report its own status as ENGAGED until said
# maintenance activities have been completed. Therefore, the high-level controller (e.g., the flight
# management unit) may delay the take-off until the service is ready.
#
# An air data computer whose readiness state is ENGAGED is required to comply with the requirements set out below.
#
# An ENGAGED service should publish the following data subjects synchronously at the same configurable rate
# which should not be lower than the specified limit. The measurements should be low-pass filtered
# to avoid frequency aliasing effects.
#
#   PUBLISHED SUBJECT NAME    SUBJECT TYPE                                                NOTE
#   calibrated_airspeed       reg.drone.physics.kinematics.translation.Velocity1VarTs
#   true_airspeed             reg.drone.physics.kinematics.translation.Velocity1VarTs     optional
#   pressure_altitude         reg.drone.physics.kinematics.translation.LinearVarTs        assume ISA; NED frame (+down, -up)
#   static_air_data           reg.drone.physics.thermodynamics.PressureTempVarTs          pressure and OAT
#
# Observe that there is no subject for indicated airspeed. This is because per the principles of service-oriented
# design, the air data computer is responsible for applying the necessary transformations on its data to render it
# ready for consumption by clients. If desired, indicated airspeed may be reported over an implementation-defined
# subject.
#
# In addition to the above, an ENGAGED service should publish the following service subjects at an
# implementation-defined rate:
#
#   PUBLISHED SUBJECT NAME    SUBJECT TYPE
#   heartbeat                 reg.drone.service.common.Heartbeat
#   sensor_status             reg.drone.service.sensor.Status
#
# Despite being a non-interactive service, it is recommended to subscribe to the readiness command subject.
# This recommendation may be omitted if the service does not support readiness state selection, in which case it should
# always report itself as being ENGAGED.
#
#   SUBSCRIBED SUBJECT NAME   SUBJECT TYPE
#   readiness                 reg.drone.service.common.Readiness
```

Smart battery

```
# This is the smart battery monitoring service. A smart battery is required to publish the following subjects:
#
#   SUBJECT                 TYPE                                          TYP. RATE [Hz]
#   energy_source           reg.drone.physics.electricity.SourceTs        1...100
#   battery_status          reg.drone.service.battery.Status              ~1
#   battery_parameters      reg.drone.service.battery.Parameters          ~0.2
#
# Observe that only the first subject can be used for estimating the endurance of the power source. The other subjects
# are designed for monitoring, diagnostics, and maintenance.
#
# Optionally, the battery service can subscribe to a readiness control subject (see reg.drone.service.common.Readiness),
# which enables the following two optional capabilities:
#
#   - SLEEP mode: when the readiness subject commands the sleep state, the battery management system may enter a
#     low power consumption state, possibly deactivating some of its capabilities.
#
#   - STANDBY mode: the battery management system may implement additional safety protections that may otherwise
#     interfere with the normal operation of the vehicle. For example, the traction battery may limit the maximum
#     load current and the depth of discharge unless the commanded state is ENGAGED. By doing so, the battery can
#     protect itself and the supplied high-voltage DC network from accidental damage while the vehicle is parked.
#     Limiting the output power or discharge of the traction battery might lead to catastrophic consequences in
#     an aerial vehicle, hence such safety checks are to be disabled once the battery is commanded into the ENGAGED
#     state.
#
# If readiness state selection is not supported, the battery may not subscribe to the readiness control subject,
# in which case it should permanently report its state as ENGAGED unless the battery is unfit for use (e.g., due
# to degradation of a failure).
#
# By convention, positive power (current) flows from the DC network into the battery. Therefore, the current is
# negative when the battery powers the system, and positive when it is being charged.
#
# Systems that leverage multiple battery packs simultaneously should be configured to publish the status of each
# pack on a separate subject.
#
# Published quantities should be low-pass filtered to avoid aliasing effects.
# Publishers should strive to sample all parameters atomically.
#
# The reported quantities are focused on the amount of energy that can be reclaimed from the battery. In a
# simplified view, this can be seen as the amount of energy that is "stored" in the battery; however, this
# interpretation is not strictly correct because the amount of retrievable energy may be dependent on external
# factors such as the temperature of the battery or the load current. Energy estimation is hard and requires
# accurate modeling of the state of the battery, which may be impossible to do without precise tracking of each
# charging cycle. Despite the complications, this is considered to be a superior approach compared to the commonly
# used alternative where the state estimation is focused on the electric charge, because the latter cannot be used
# directly to predict the endurance of the system.
#
# The methods of energy estimation are implementation-defined.
```

ESC

```
# The electronic speed controller (ESC) service is designed for controlling and monitoring electric drives.
# From the standpoint of this standard, an electric drive is just a special case of a servo. For generality,
# COTS electric drives are recommended to also support the servo interface defined in the adjacent namespace.
#
# ESCs (drives) are segregated into groups. Each ESC in a group has an index that is unique within the group.
# Drives in a group are commanded synchronously by publishing a message containing an array of setpoints.
# There are several subjects defined:
#
#   - Setpoint array subject. Every participant subscribes to the same setpoint subject.
#     Every message is consumed by all participants according to their index in the group.
#     The setpoint subject defines the group. There may be an arbitrary number of such groups in the network.
#
#   - Readiness subject. Every participant subscribes to the same readiness control subject which is used to command
#     the state of the group: sleep, standby, or engaged. In many cases there will be one global subject controlling
#     the state of the entire system; in other cases there will be dedicated controls on a per-subsystem basis.
#
#   - Individual subjects, whose subject-ID is offset from the setpoint subject-ID `S` as a function of
#     group member index `i` as specified below.
#
#                                                SUBJECT
#                                                NAME        SUBJECT TYPE                                    SUBJECT-ID
#  +-----------------+
#  |   Controller    |---------+------------+----... setpoint    reg.drone.service.actuator.common.sp.*      S
#  |                 |-------+-)-----------+-)----... readiness   reg.drone.service.common.Readiness          S+1
#  +-----------------+       | |           | |
#   ^ ^ ^ ^  ^ ^ ^ ^        v v           v v
#   | | | |  | | | |   +----------+  +----------+
#   | | | |  | | | |   |Drive i=0|  |Drive i=1| ...
#   | | | |  | | | |   +----------+  +----------+
#   | | | |  | | | |     | | | |       | | | |
#   | | | |  | | | +-----+ | | |       | | | |       feedback    reg.drone.service.actuator.common.Feedback   S+(i+1)*5+1
#   | | | |  | | +----------+ | |       | | | |       status      reg.drone.service.actuator.common.Status     S+(i+1)*5+2
#   | | | |  | +--------------+ |       | | | |       power       reg.drone.physics.electricity.PowerTs        S+(i+1)*5+3
#   | | | |  +------------------+       | | | |       dynamics    reg.drone.physics.dynamics.rotation.PlanarTs S+(i+1)*5+4
#   | | | |                             | | | |
#   | | | +-----------------------------+ | | |
#   | | +---------------------------------+ | |
#   | +-------------------------------------+ |
#   +-----------------------------------------+
#
# Therefore, in order to configure a group member, one has to set up two parameters: the setpoint subject-ID and
# the group member index.
#
# Notice that the physics subjects are timestamped.
#
# Vendor/application-specific subjects are not shown here; their subject-IDs are implementation-defined.
# Vendors are encouraged to publish additional data (e.g., temperatures) on separate subjects.
#
#
#   SETPOINT SUBJECT
#
# The setpoint subject is ignored unless the drive is ENGAGED. As long as the drive is not ENGAGED, it shall not apply
# any power to the load excepting non-operational scenarios such as maintenance and diagnostics, which are
# outside of the scope of this service definition. More on readiness and safety in the next section.
#
# Upon reception of a setpoint message, a group participant fetches its setpoint from the array using the array
# element whose index equals the index of the group participant. By virtue of the Implicit Zero Extension Rule,
# if the message is too short, the setpoint will be interpreted as zero.
#
# If no valid setpoint was received in CONTROL_TIMEOUT or a lower implementation-specific value,
# the drive should assume a zero setpoint for safety reasons.
# The minimum setpoint publication period should be at least twice lower than its timeout.
#
# While stopped, the drive may either allow the load to freewheel or it may force it to a particular parking position,
# depending on the implementation requirements. The actual state of the load may be continuously reported using the
# dynamics subject. Notice that per the safety rule introduced earlier, the parking position may be impossible
# to enforce unless the drive is ENGAGED because it may require delivering power to the load.
#
# The setpoint message types that can be used to command a group of drives are defined in
# reg.drone.service.actuator.common.sp; please read the documentation related to that namespace for further information.
# Servo setpoint message types may also be supported on an implementation-specific basis for enhanced interoperability.
# If the group is controlled using different setpoint subjects concurrently, the behavior is implementation-defined.
#
# The following control modes are defined, none of which are mandatory to support.
# The control mode in use is to be specified using the register API.
# This service does not support switching the control mode or setting the motion profile at runtime;
# for that, please refer to the servo service.
#
#   0. Ratiometric voltage control. Each setpoint scalar is a value normalized/saturated in [-1, +1] representing
#      the Q-axis/phase/armature (depending on the type of the drive) voltage as a fraction of the maximum.
#      This control mode emulates the behavior of a typical RCPWM-controlled BLDC drive.
#
#   1. Ratiometric current/torque control. Each setpoint scalar is a value normalized/saturated in [-1, +1] representing
#      the Q-axis/phase/armature (depending on the type of the drive) current as a fraction of the maximum.
```

```
#      A negative setpoint during forward rotation (positive during reverse rotation) commands braking.
#
#   2. Speed control. Each setpoint scalar contains the target angular velocity of the load in radian/second.
#
#   -. More control modes may be added later. Which control modes are supported is implementation-defined.
#
# Considerations that apply to all control modes:
#  -  Negative setpoint values represent reversal; a positive setpoint is co-directed with positive rotation/torque.
#  -  If reverse operation is not supported, negative values should be clamped to zero.
#  -  A non-finite setpoint is to be treated as zero.
#
#
#   READINESS SUBJECT
#
# The default state is STANDBY. While in this state, the drive is not allowed to deliver power to the load,
# and the setpoint subject is ignored. The drive shall enter this state automatically if the readiness subject
# is not updated for CONTROL_TIMEOUT.
#
# While the drive is ENGAGED, the setpoint commands are processed normally as described in the adjacent section.
# If the drive does not support bidirectional operation, implementations are recommended to ensure that the load
# is driven at some minimum power level (idling) while the drive is ENGAGED regardless of the commanded setpoint,
# unless such behavior is deemed incompatible with the functional requirements of the controlled drive.
#
# If the selected readiness state is SLEEP, the behavior is implementation-defined. Implementations are recommended to
# power off the high-voltage circuitry and all non-essential components (e.g., LED indication, sensors, etc.)
# to minimize the power consumption.
#
# Implementations are recommended to announce transitions between the readiness states using audiovisual feedback.
#
# The worst-case state transition latency is not defined. The controlling element (that is, the unit that publishes
# to the setpoint and readiness subjects) is expected to monitor the actual readiness status of each component using
# the feedback subject. For example, a sensorless electric motor drive may choose to spool-up before entering the
# ENGAGED state, which would obviously take time; as soon as the spool-up is finished, the drive would switch its
# reported status from STANDBY to ENGAGED, thereby indicating that it is ready for normal operation.
#
#
#   PUBLISHED SUBJECTS
#
# The following subjects shall be published immediately after a new setpoint is applied even if the drive is STANDBY:
#
#   SUBJECT             RECOMMENDED PRIORITY
#   ------------------------------------------------
#   feedback            same as the setpoint
#   power               second to the setpoint
#   dynamics            second to the setpoint
#
# If no setpoint is being published, these subjects should continue being updated at least at 1/MAX_PUBLICATION_PERIOD.
# The publication rate requirements do not apply if the readiness state is SLEEP.
#
# If the setpoint publication rate exceeds 50 Hz, implementations are allowed (but not required) to throttle these
# subjects by dropping some of the messages such that the publication rate of each subject does not exceed 50 Hz.
# Implementations operating over Classic CAN are recommended to do this.
#
# The other subjects may be published at an implementation-defined rate and priority,
# which should be consistent across the group.
#
# Implementations are encouraged to provide additional subjects for enhanced feedback and monitoring.
#
# The measurements carried by the published messages should be low-pass filtered with an adequate cutoff frequency to
# avoid aliasing effects. Implementations should strive to sample all parameters simultaneously.
#
# If a float-typed reported quantity is unknown, the corresponding value should be NaN.
#
#
#   CONVENTIONS AND ASSUMPTIONS
#
# A drive powers a rotary mechanical load that may be connected via a gearbox. It is the responsibility of
# the drive to account for the gear ratio of the gearbox when calculating related parameters such as angular
# velocity or torque.
#
# It is assumed that there is a well-defined direction of rotation that is referred to as forward rotation.
# A positive angular velocity represents forward rotation. Likewise, forward torque is positive.
#
# It is assumed that the drive is powered from a DC electric power supply network. A positive electric current
# represents current flowing from the network into the drive, also referred to as the state of driving/motoring.
# The opposite -- braking/regeneration -- is represented by negative current.
#
# Excepting edge cases and transients, torque and current are generally of the same sign.
# The above is summarized on the following four-quadrant diagram:
#
```

```
#                                 +velocity
#                                    ^
#                               braking,| forward,
#                               negative| positive
#                               power   | power
#                              ---------+---------->  +torque/current
#                               reverse,| braking,
#                               positive| negative
#                               power   | power
```

Servo

```
# A servo can actuate either a translational or rotary load using electric power from the high-voltage DC bus.
#
# The type of load (translational or rotational) dictates which type is used for commanding the setpoint and reporting
# the status:
#   - reg.drone.physics.dynamics.rotation.Planar[Ts]
#   - reg.drone.physics.dynamics.translation.Linear[Ts]
# For generality, either or both of these types are referred to as "timestamped dynamics" or "non-timestamped dynamics".
#
# The default readiness state is STANDBY. While in this state, the servo is not allowed to apply force to the load,
# and the setpoint subject is ignored. The servo shall enter the STANDBY state automatically if the readiness subject
# is not updated for CONTROL_TIMEOUT.
#
# The subjects defined by this service are shown on the following canvas. Implementers are encouraged to add
# custom subjects with additional data. Notice that the physics subjects are timestamped.
#
#                    SUBJECT NAME                        SUBJECT TYPE                                    RATE
#
#   +------------+   setpoint       +------------+      (non-timestamped dynamics) (see below)         R
#   |            |----------------->|            |
#   |            |   readiness      |            |      reg.drone.service.common.Readiness              any
#   |            |----------------->|            |
#   |            |   feedback       |            |      reg.drone.service.actuator.common.Feedback      R
#   |            |<-----------------|            |
#   | Controller |   status         |   Servo    |      reg.drone.service.actuator.common.Status        any
#   |            |<-----------------|            |
#   |            |   power          |            |      reg.drone.physics.electricity.PowerTs           R
#   |            |<-----------------|            |
#   |            |   dynamics       |            |      (timestamped dynamics)                          R
#   |            |<-----------------|            |
#   +------------+                  +------------+
#
# Should it be necessary to control a group of servos in lockstep, an arbitrary number of them may subscribe
# to the same setpoint subject (their published subjects would be different of course).
#
# If the servo is ENGAGED, setpoint messages are processed as follows: the first field of the kinematic setpoint type
# that contains a finite value is taken as the commanded setpoint. The following non-negative finite fields define
# the motion profile, where negative and non-finite values are ignored.
#
# For example, a translational dynamics message containing the following values:
#   position     = +0.35
#   velocity     = NaN
#   acceleration = NaN
#   force        = 30
# ...is interpreted as follows: position the load at 0.35 meters relative to the neutral, limit the force to 30 newton,
# do not limit the velocity and acceleration. Here is another example:
#   angular position     = NaN
#   angular velocity     = +400
#   angular acceleration = NaN
#   torque               = 50
# which is interpreted as follows: reach the angular velocity of 400 radian/second in the forward direction,
# limit the torque to 50 newton*meters, do not limit the acceleration.
#
# The motion profile parameters that are not supported are to be silently ignored by the servo. If the commanded
# parameter cannot be controlled by the servo, the setpoint is to be ignored. For example, in the second example above,
# if the servo does not support angular velocity control, the setpoint message would be discarded.
#
# The above describes the typical use case where each servo is controlled over a dedicated setpoint
# subject independently (or a group of servos are controlled in lockstep using the same setpoint subject).
# Some applications may require synchronous independent control of multiple servos in a group, similar to ESC.
# To address this, a compliant servo should support another operating mode where the controlled quantity
# (position, velocity, force, etc.) is selected statically along with the motion profile (using the register API),
# and the servo subscribes to the setpoint subject of type "reg.drone.service.actuator.common.sp.*".
# Having its index in the group configured statically, the servo fetches the setpoint from the appropriate
# index in the setpoint array.
# The resulting topology closely resembles that of the ESC service:
#
```

```
#                                                SUBJECT
#                                                NAME         SUBJECT TYPE
#  +----------------+
#  |   Controller   |---------+------------+----... setpoint     reg.drone.service.actuator.common.sp.*
#  |                |-------+-)----------+-)----... readiness    reg.drone.service.common.Readiness
#  +----------------+       | |          | |
#   ^ ^ ^ ^  ^ ^ ^ ^        v v          v v
#   | | | |  | | | |   +----------+  +----------+
#   | | | |  | | | |   |Servo i=0|  |Servo i=1| ...
#   | | | |  | | | |   +----------+  +----------+
#   | | | |  | | | |     | | | |      | | | |
#   | | | |  | | | +-----+ | | |      | | | |       feedback     reg.drone.service.actuator.common.Feedback
#   | | | |  | | +----------+ | |      | | | |       status       reg.drone.service.actuator.common.Status
#   | | | |  | +--------------+ |      | | | |       power        reg.drone.physics.electricity.PowerTs
#   | | | |  +------------------+      | | | |       dynamics     (timestamped dynamics)
#   | | | |                            | | | |
#   | | | +----------------------------+ | | |
#   | | +--------------------------------+ | |
#   | +------------------------------------+ |
#   +----------------------------------------+
#
# If the selected readiness state is SLEEP, the behavior is implementation-defined. Implementations are recommended to
# power off the high-voltage circuitry and all non-essential components (e.g., LED indication, sensors, etc.)
# to minimize the power consumption. The publication rate requirements do not apply if the state is SLEEP.
#
# The worst-case readiness state transition latency is not defined.
#
# The following subjects shall be published immediately after a new setpoint is applied even if the servo is STANDBY:
#
#   SUBJECT NAME        RECOMMENDED PRIORITY
#   ------------------------------------------------
#   feedback            same as the setpoint
#   power               second to the setpoint
#   dynamics            second to the setpoint
#
# If no setpoint is being published, these subjects should continue being updated at least at 1/MAX_PUBLICATION_PERIOD.
#
# If the setpoint publication rate exceeds 50 Hz, implementations are allowed (but not required) to throttle these
# subjects by dropping some of the messages such that the publication rate of each subject does not exceed 50 Hz.
# Implementations operating over Classic CAN are recommended to do this.
#
# The other subjects may be published at an implementation-defined rate and priority,
# which should be consistent across the group.
#
# The measurements carried by the published messages should be low-pass filtered with an adequate cutoff frequency to
# avoid aliasing effects. Implementations should strive to sample all parameters simultaneously.
#
# It is assumed that the servo is powered from a DC electric power supply network. A positive electric current
# represents current flowing from the DC network into the servo (negative represents regeneration).
#
# Excepting edge cases and transients, torque/force and current are generally of the same sign (barring the difference
# introduced by the power dissipated by the servo itself).
#
#                                  +velocity
#                                      ^
#                              braking,| forward,
#                              negative| positive
#                              power   | power
#                             ---------+---------->  +torque/force/current
#                              reverse,| braking,
#                              positive| negative
#                              power   | power
```

Implementation guidelines

Examples and reference implementations that can be used as a starting point when building a compliant avionic unit can be asked for at the UAVCAN Forum.

Regardless of the above, the general design process would normally look as follows:

1. Understand the basic principles of UAVCAN by skimming through The UAVCAN Guide.
2. Read the documentation for the `reg.drone` namespace to understand how the Drone application layer is composed.
3. Pick a suitable UAVCAN library (e.g., libcanard) or a full-featured example application to use it as a starting point.
4. Build a minimal UAVCAN node that is only capable of publishing its heartbeat.
5. Gradually add the required standard application layer services listed in the previous section of the document. If the implementation is based on a low-level library like libcanard, this step may require implementing the required application-layer functions like the Register API manually, although this is not expected to require more than a few hundred lines of trivial code.
6. Implement the desired domain-specific services defined in the `reg.drone` namespace.

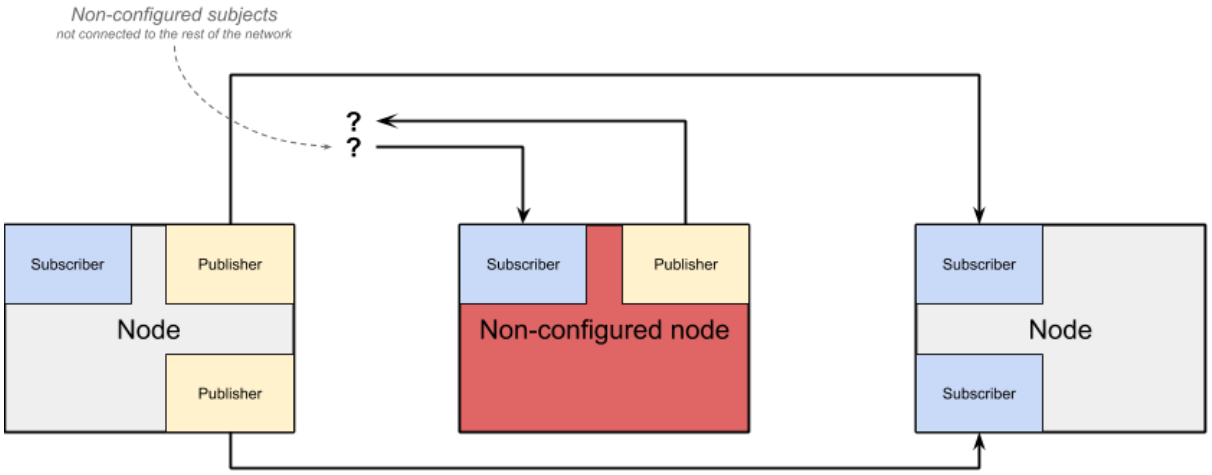
System integration guidelines

In the absence of advanced autoconfiguration capabilities, a new COTS node is to be integrated into the system following these steps:

1. The new unconfigured node is connected to the network and the vehicle is turned on.
2. The node-ID is configured.
    - If the node-ID has previously been defined statically via the register **`uavcan.node.id`**, the preconfigured static value is used. This is the case if the node is an autoconfiguration authority. Initially, the static node-ID can be configured using some other means of communication like a CLI management interface, MAVLink, etc. Should there happen to be a node-ID conflict on the bus, it is to be resolved by disconnecting one of the conflicting nodes and changing its static node-ID before reconnecting it back.
    - If the node-ID is not defined and the node supports the PnP protocol, it sends PnP node-ID allocation requests until a node-ID is granted by the local autoconfiguration authority.
3. At this point, the node is online but is unable to communicate with the rest of the intravehicular distributed computing system because its subject identifiers are not yet configured. The human integrator sets up the subject-ID parameters using the standard

registers defined above to establish the application-layer pub-sub data links with the rest of the network.

4. Once the subject-IDs have been set up, the node is ready to work.



Future capabilities

Future revisions of this standard are expected to provide the following additional capabilities:

- Highly automated plug-and-play node configuration. The intention is to support automatic or semi-automatic node-ID and port-ID configuration for newly connected nodes to reduce the system integration efforts and reduce the risk of misconfiguration by reducing the cognitive workload on the human.
- UAVCAN-MAVLink interoperation services for integration with ground control software.
- Security facilitation services – key exchange, authentication, etc.
- Additional capabilities in the drone application layer (namespace `reg.drone`).

Physical layer

This chapter covers the electromechanical aspects of DS-015: physical layer, connectivity options, electrical and electromechanical design considerations. It contains a set of transport-agnostic recommendations followed by transport-specific sections. This chapter is necessary because the underlying networking standard UAVCAN does not define the physical layer, leaving it to domain-specific standards instead like this one. Following the requirements and recommendations of this chapter will ensure the highest level of inter-vendor compatibility and allow the developers to avoid common design pitfalls.

General recommendations

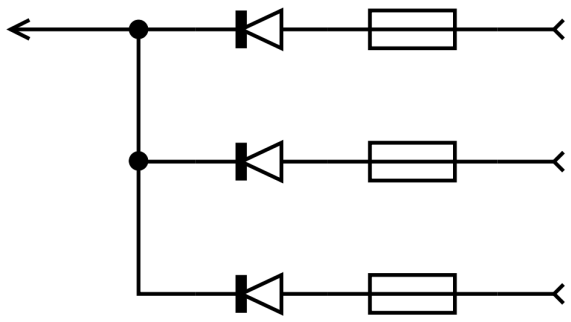
Integrated power supply network

Integration of the power distribution functionality with the communication infrastructure removes the need for a dedicated power distribution network, which has the potential to simplify the system design and reduce the complexity and weight of the wiring harnesses. Redundant power supply topologies can be easily implemented on top of a redundant communication infrastructure.

Power input

A node that draws power from the power supply network should protect its power inputs with an over-current protection circuitry that is capable of disconnecting the input if the power consumption of the node exceeds its design limits. This measure is necessary to prevent a short-circuit or a similar failure of an individual node from affecting other nodes connected to the same power supply network.

In the case of redundant power supply connections where a node is connected to more than one power supply network concurrently, each such connection should be equipped with a circuit that prevents reverse current flow from the node into the power supply network. This measure is necessary to prevent a short-circuit or a similar failure of an individual power supply network from affecting other power supply networks in the same redundant group.

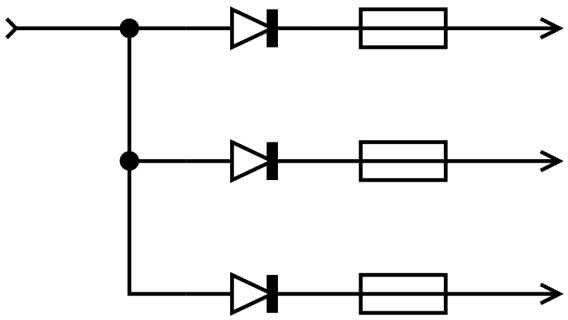


Redundant power input schematic

Power output

A node that delivers power to the power supply network should equip each of its power outputs with a circuit that prevents reverse current flow from the power supply network into the node. This measure is necessary to prevent a short-circuit or a similar failure of the node from affecting the power supply network.

In the case of redundant power output connections where a node provides power to more than one power supply network concurrently, each such connection should be equipped with a circuit that is capable of disconnecting the output if the power consumption per network exceeds the design limits. This measure is necessary to prevent a short-circuit or a similar failure of an individual power supply network from affecting other power supply networks in the same redundant group.



Redundant power output schematic

UAVCAN/CAN recommendations

This section specifies the DS-015 physical layer for UAVCAN/CAN. Here and in the following parts of this section, "CAN" implies both Classic CAN and CAN FD, unless specifically noted otherwise.

Physical connector specification

This standard defines several connector types optimized for different applications: from highly compact systems to large deployments, from low-cost to safety-critical applications.

Each connector type specification includes an integrated power supply interface. Implementations should provide two identical parallel connectors for each CAN interface per device instead of relying on T-connectors.

T-connectors should be avoided because typically they increase the stub length, weight, and complexity of the wiring harnesses.

UAVCAN/CAN Micro connector

The UAVCAN/CAN Micro connector is intended for weight and space-sensitive applications. It is a board-level connector, meaning that it is installed on the PCB rather than on the panel. The Micro connector is compatible with the Pixhawk Connector Standard.

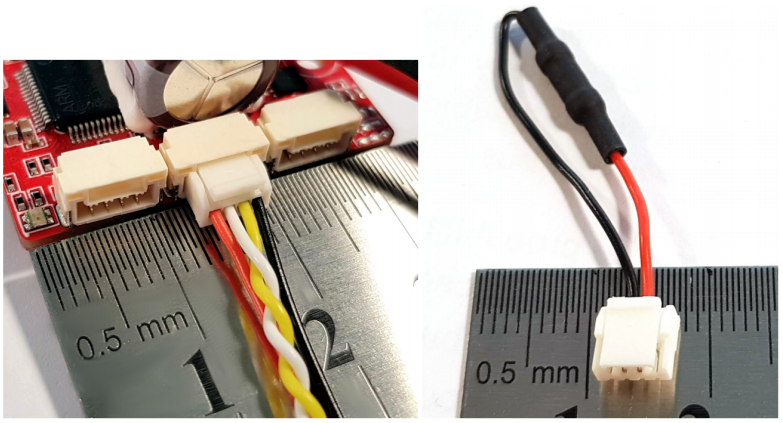
Advantages	Disadvantages
➔ Extremely compact, low-profile. The PCB footprint is under 9×5 millimeters. ➔ Secure positive lock ensures that the connection will not self-disconnect when exposed to vibrations. ➔ Low cost.	➔ Board-level connections only. No panel-mounted options available. ➔ No shielding available. ➔ Not suitable for safety-critical hardware.

The UAVCAN/CAN Micro connector is based on the proprietary JST GH 4-circuit connector type . The CAN physical layer standard that can be used with this connector type is ISO 11898-2.

Devices that deliver power to the bus are required to provide 4.9–5.5 V on the bus power line, 5.0 V nominal. Devices that are powered from the bus should expect 4.0–5.5 V on the bus power line.

The following table documents the pinout specification for the UAVCAN/CAN Micro connector type. The suitable wire type is #30 to #26 AWG, outer insulation diameter 0.8–1.0 mm, multi-strand. Wires “CAN high” and “CAN low” shall form a twisted pair.

#	Function	Note
1.	Bus power supply	5 V nominal. See the power supply requirements
2	CAN high	Twisted with “CAN low” (pin 3).
3	CAN low	Twisted with “CAN high” (pin 2).
4	Ground	-



UAVCAN/CAN Micro connectors

Right-angle connector with a twisted pair cable connected; a 120Ω termination plug.

Future capabilities

It is recognized that the UAVCAN/CAN Micro connector is unsuitable for many applications, particularly those where ruggedness, reliability, and resilience to adverse environments is required. Future versions of this standard are expected to address this by adding new connector options. The current candidates are listed below for reference; please provide feedback on the forum:

- **UAVCAN/CAN D-Sub**: Generic **D-Subminiature DE-9** with 24V, 3A integrated power. This is the de-facto standard connector for CAN, supported by many current specifications.
- **UAVCAN/CAN M8**: Generic **M8 5-circuit B-coded** with 24V, 3A integrated power. This connector type is also commonly found in various CAN applications and is compatible with the CiA 103 (CANopen) standard.

CAN bus physical layer parameters

Vendors should follow the recommendations provided in this section in the interest of maximizing the cross-vendor compatibility.

Classic CAN

The following table lists the recommended parameters of the ISO 11898-2 Classic CAN 2.0 physical layer. The estimated bus length limits are based on the assumption that the propagation delay does not exceed 5 ns/m, not including additional delay times of CAN transceivers and other components.

Parameter	Value				Unit
Bit rate	1000	500	250	125	kbit/s
Permitted sample point location	75–90	85–90	85–90	85–90	%
Recommended sample point location	87.5	87.5	87.5	87.5	%
Maximum bus length	40	100	250	500	m
Maximum stub length	0.3	0.3	0.3	0.3	m

Designers are encouraged to implement CAN auto bit rate detection when applicable. Refer to the CiA 801 application note for the recommended practices.

The development team maintains a spreadsheet that can be used to gauge the Classic CAN network resource utilization.

Note: UAVCAN allows the use of a simple bit time measuring approach, as it is guaranteed that any functioning UAVCAN network will always exchange node status messages, which can be expected to be published at a rate no lower than 1 Hz, and that contain a suitable alternating bit pattern in the CAN ID field. Refer to chapter 5 of the UAVCAN v1 specification for details.

CAN FD

This section is under development and will be populated in a later revision of the standard.

Parameter	Segment	Value				Unit
Bit rate	**Arbitration**	1000	500	250	125	kbit/s
	Data	4000	2000	1000	500	
Permitted SPL	**Arbitration**	TBD	TBD	TBD	TBD	%
	Data	TBD	TBD	TBD	TBD	
Recommended SPL	**Arbitration**	TBD	TBD	TBD	TBD	%
	Data	TBD	TBD	TBD	TBD	
Maximum bus length		TBD	TBD	TBD	TBD	m
Maximum stub length		TBD	TBD	TBD	TBD	

Profiles

CAN bus topologies

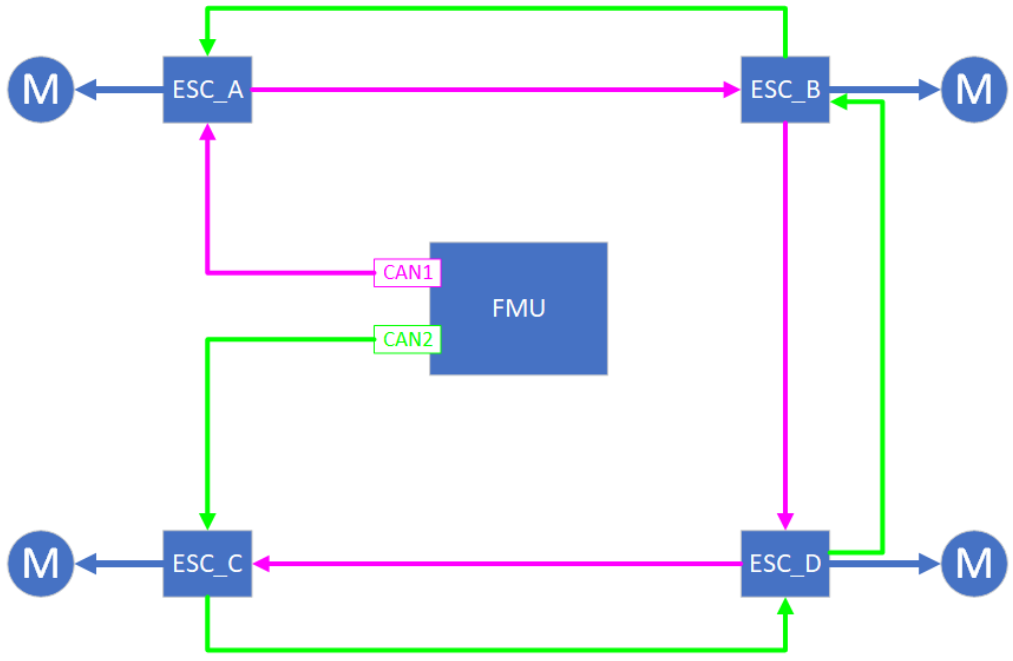
This section lists typical CAN bus topologies recommended for use with this standard. These are not mandatory requirements.

Non-redundant daisy-chain topology

Configurations where a high degree of reliability is not required may implement the conventional topology where a single physical CAN bus is routed through every node in series, forming the typical daisy-chain circuit. This configuration is prone to partitioning should the cable system or at least one of the nodes fail in a mode that disrupts the electrical continuity of the bus.

Redundant daisy-chain topology

In this configuration, the redundant physical buses are routed in the opposite directions to mitigate the partitioning failure mode where one node causes a disruption of the physical buses passing through it, or the redundant cabling system is damaged in the same place.



Quad copter system with dual CAN bus

Star topology

Simple networks where the total cable length is expected to be small or the required data rate is low may leverage the star topology (whether redundant or not) such that the bus is routed through a central passive hub. The hub interconnects all branches into a single electrical network (i.e., it is not an active unit) per redundant bus. Due to signal reflection and the associated signal integrity issues, this topology does not scale to large networks.

Redundant transports

The transport redundancy capability supported by UAVCAN is completely transparent to the application: an application exchanging data over UAVCAN does not have to be aware of the configuration of the underlying transport layer, as the UAVCAN stack automatically fans-out outgoing transfers and deduplicates received ones. It follows that should one of the redundant transports fail at runtime, the UAVCAN stack will automatically find a new configuration that is still functional, shielding the application from the related complexity and transparently handling the edge cases that arise in such scenarios.

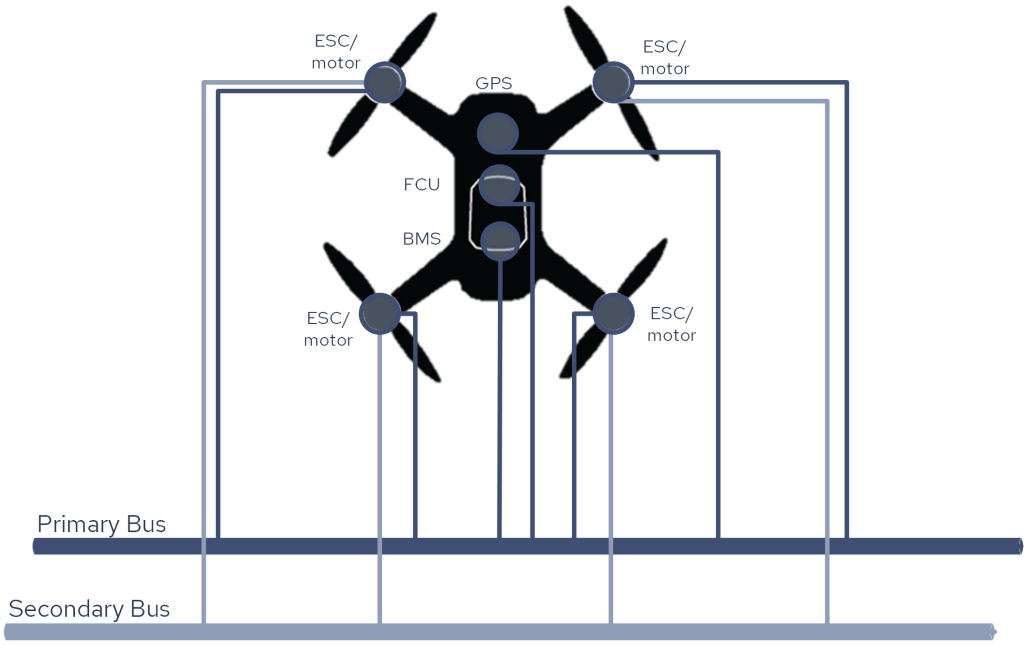
Normally, in a redundant configuration all transports interconnect all of the involved nodes, which is referred to as *symmetric redundancy*. *Asymmetric redundancy* is also supported, where the redundant transports interconnect different sets of nodes:

- The primary transport interconnects all nodes in the network.
- The secondary transport interconnects only the mission-critical nodes in the network, which are the subset of the above.
- The tertiary transport, if present, interconnects yet more critical nodes, which are the subset of the above.

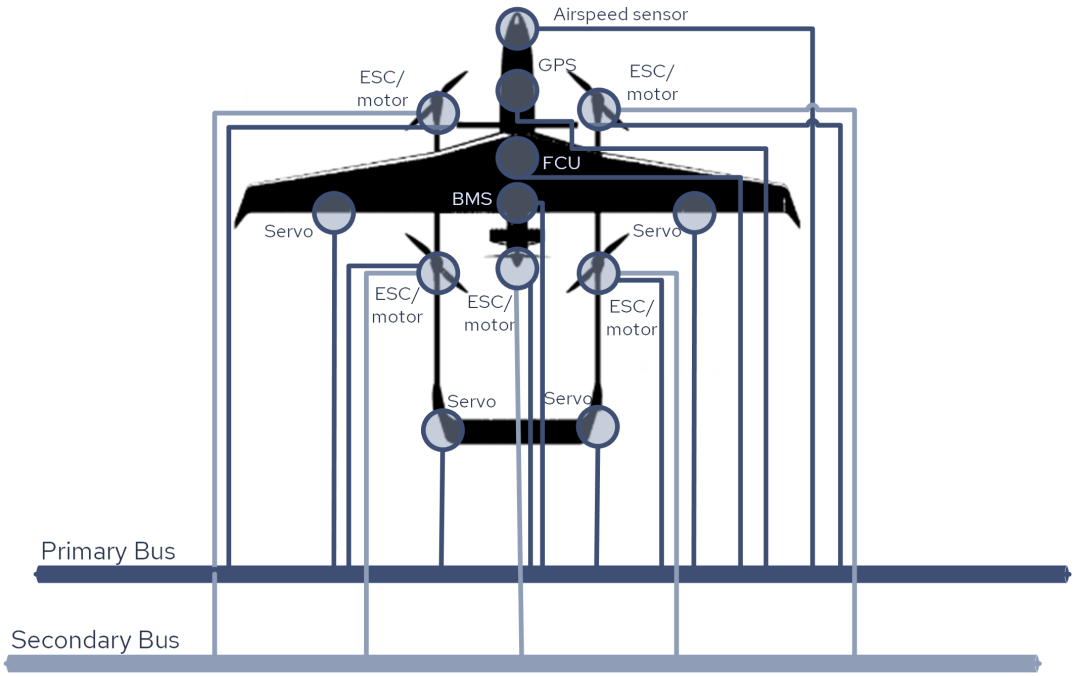
Therefore, nodes that are connected to the tertiary transport use a triply-redundant transport, nodes that are connected to the secondary transport use doubly-redundant transport, and the first category of nodes (non-critical) use the non-redundant transport.

Vehicular topologies examples

Quadrotor



Quad VTOL



Conformity

Conformity testing is performed by the UAVCAN Consortium. Please reach out to consortium@uavcan.org for details.