Unmanned Maritime Autonomy Architecture (UMAA) Sensor and Effector Management (SEM) Interface Control Document (ICD) (UMAA-SPEC-SEMICD)

MDE Version 4.2.1 Commit 7c2b513

UMAA Spec Commit f2fa426

Version 3.0.1
25 February 2021

UMAA Sensor and Effector Management ICD

(UMAA-SPEC-SEM-ICD)

Signature Page

Submitted by: Rothgeb.Mark.E.ORC3011008580.ID Digitally signed by Rothgeb.Mark.E.ORC3011008580.ID Date: 2021.05.05 11:55:53 -04'00' **Date Signed:** _____

Mark Rothgeb
Unmanned Maritime Autonomy Architecture
Standards Board Chair

PMS 406 Approvals:

ANDERSON.JEREMIAH.P.1254266481 Digitally signed by ANDERSON.JEREMIAH.P.1254266481 Date: 2021.05.10 14:53:10 -04'00' **Date Signed:** _____

CDR Jeremiah Anderson
PMS 406 Advanced Autonomous Capabilities,
Principal Assistant Program Manager

SMALL.PETER.D.1048778646 Digitally signed by SMALL.PETER.D.1048778646 Date: 2021.06.03 17:19:50 -04'00' **Date Signed:** _____

CAPT Pete Small
Program Manager
PMS 406 Unmanned Maritime Systems,
PEO Unmanned and Small Combatants

Contents

List of Figures

List of Tables

1 Scope

1.1 Identification

This document defines a set of services as part of the Unmanned Maritime Autonomy Architecture (UMAA). As such, it provides services that focus on sensors and effectors that are often referred to as payloads but can also be organic to the vehicle. This ICD focuses on control and management of these resources. Examples include radars, sonars, manipulators, payload deployments, kinetic effects, and electronic warfare effects. The services and their corresponding interfaces covered in this ICD encompass the functionality to specify an Unmanned Maritime Vehicle (UMV) (surface or undersea) mission. This document is generated automatically from data models that define its services and their interfaces as part of the Unmanned Systems (UxS) Control Segment (UCS) Architecture as extended by UMAA to provide autonomy services for UMVs.

To put each ICD in context of the UMAA Architecture Design Description (ADD), the UMAA functional decomposition mapping to UMAA ICDs is shown in Figure 1.

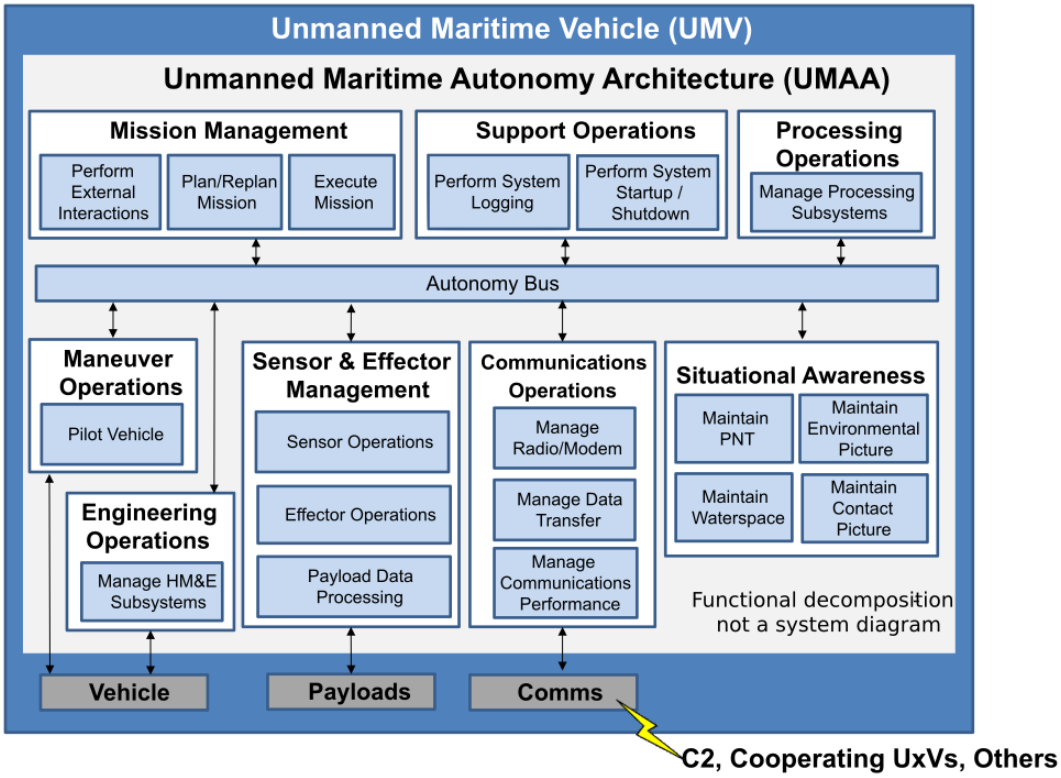

Figure 1: UMAA Functional Organization

1.2 Overview

The fundamental purpose of UMAA is to promote the development of common, modular, and scalable software for UMV's that is independent of a particular autonomy implementation. Unmanned Maritime Systems (UMSs) consist of Command and Control (C2), one or more UMVs, and support equipment and software (e.g. recovery system, Post Mission Analysis applications). The scope of UMAA is focused on the autonomy that resides on-board the UMV. This includes the autonomy for all classes of UMVs and must support varying levels of communication in mission (i.e., constant, intermittent, or none) with its C2 System. To enable modular development and upgrade of the functional capabilities of the on-board autonomy, UMAA defines eight high-level functions. These core functions include: Communications Operations, Engineering Operations, Maneuver Operations, Mission Management, Processing Operations, Sensor and Effector Operations, Situational Awareness, and Support Operations. In each of these areas, it is anticipated that new capabilities will be required to satisfy evolving Navy missions over time. UMAA seeks to define standard interfaces for these functions so that individual programs can leverage capabilities developed to these standard interfaces across programs that meet the standard interface specifications. Individual programs may group services and interfaces into components in different ways to serve their particular vehicle's needs. However, the entire interface defined by UMAA will be required as defined in the ICDs for all services that are included in a component. This requirement is what enables autonomy software to be ported between heterogeneous UMAA-compliant

vehicles with their disparate vendor-defined vehicle control interfaces without recoding to a vehicle specific platform interface.

Sensor and Effector Management defines the services required to describe an UMV's sensors and effectors. Figure 2 depicts an example of a possible component service grouping is shown with the dashed lines. While not required, a document describing the format for oceanographic data collected for the Naval Oceanographic Office (NAVO) is provided in Section 2.

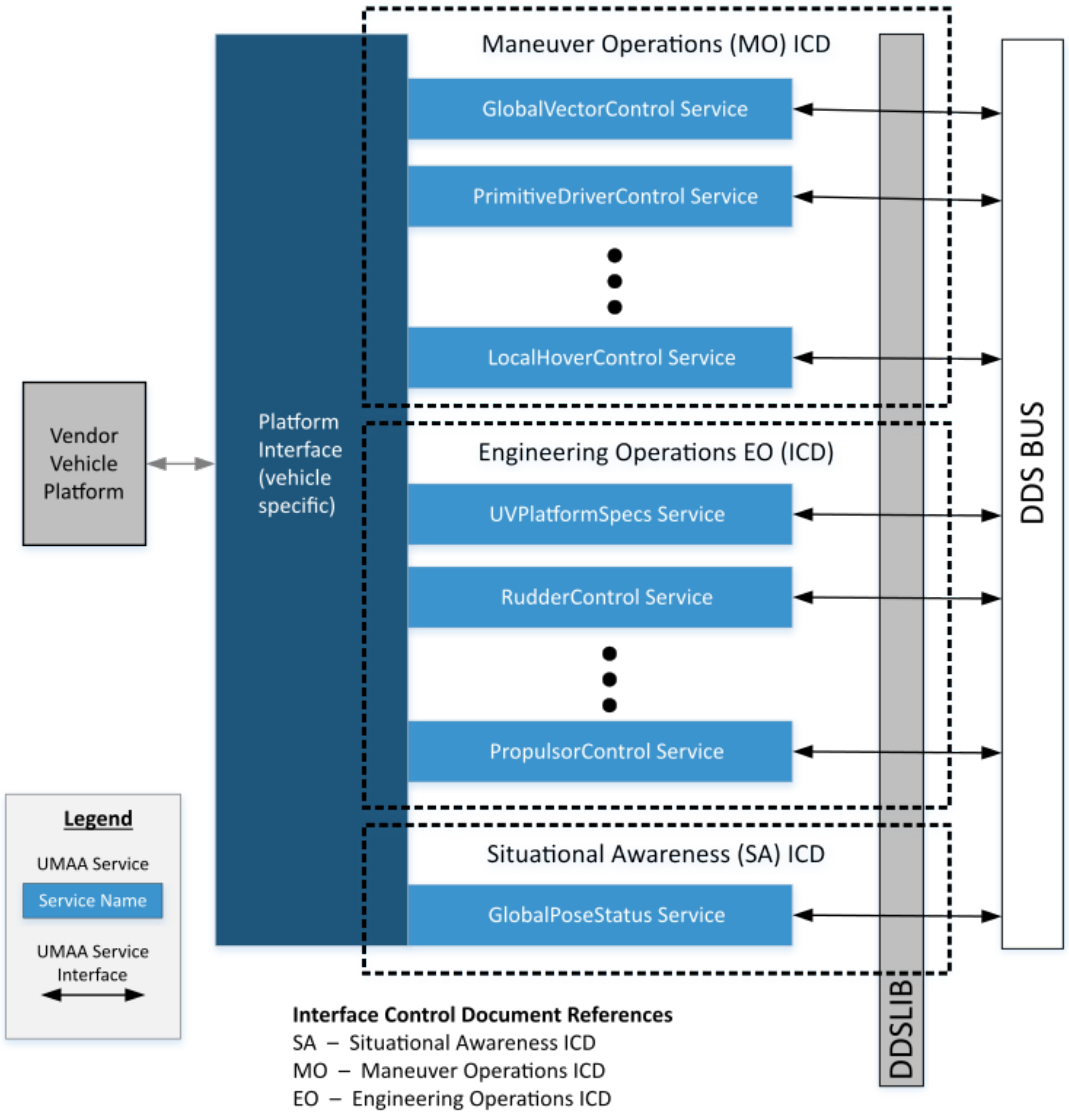

Figure 2: UMAA Services and Interfaces Example

1.3 Document Organization

This interface control document is organized as follows:

Section 1 – Scope: A brief purview of this document

Section 2 – Referenced Documents: A listing of associated of government and non-government documents and standards

Section 3 – Introduction to Data Model, Services, and Interfaces: A description of the common data model across all services and interfaces

Section 4 – Flow Control: A description of different flow control patterns used throughout UMAA.

Section 5 – Sensor and Effector Management (SEM) Services and Interfaces: A description of specific services and interfaces for this ICD

2 Referenced Documents

The documents in the following table were used in the creation of the UMAA interface design documents. Not all references may be applicable to this particular document.

Table 3: Standards Documents

Title	Release Date
A Universally Unique IDentifier (UUID) URN Namespace	July 2005
Data Distribution Service for Real-Time Systems Specification, Version 1.4	March 2015
Data Distribution Service Interoperability Wire Protocol (DDSI-RTPS), Version 2.3	April 2019
Object Management Group Interface Definition Language Specification (IDL)	March 2018
Extensible and Dynamic Topic Types for DDS, Version 1.3	February 2020
UAS Control Segment (UCS) Architecture, Architecture Description, Version 2.4	27 March 2015
UCS Architecture, Conformance Specification, Version 2.2	27 September 2014
UCS-SPEC-MODEL v3.4 Enterprise Architect Model	27 March 2015
UCS Architecture, Architecture Technical Governance, Version 2.5	27 March 2015
System Modeling Language Specification, Version 1.5	May 2017
Unified Modeling Language Specification, Version 2.5.1	December 2017
Interface Definition Language (IDL), Version 4.2	March 2018
U.S. Department Of Homeland Security, United States Coast Guard "Navigation Rules International-Inland" COMDTINST M16672.2D	March 1999
IEEE 1003.1-2017 - IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7	December 2017

Table 4: Government Documents

Title	Release Date
Unmanned Maritime Autonomy Architecture (UMAA) Architecture Design Description (ADD), Version 1.0	January 2019
MANUAL FOR THE SUBMISSION OF OCEANOGRAPHIC DATA COLLECTED BY UNMANNED UNDERSEA VEHICLES (UUVs)	October 2018

3 Introduction to Data Model, Services, and Interfaces

3.1 Data Model

A common data model is at the heart of UMAA. The common data model describes the entities that represent system state data, the attributes of those entities and relationships between those entities. This is a "data at rest" view of system level information. It also contains data classes that define types of messages that will be produced by components, a "data in motion" view of system level information.

The common data model and coordinated service interfaces are described in a Unified Modeling Language (UML™) modeling tool and are represented as UML™ class diagrams. Interface definition source code for messages/topics and other interface definition products and documentation will be automatically generated from the common data model to assure they are consistent with the data model and to ensure delivered software matches its interface specification.

The data model is maintained as a maritime extension to the UCS Architecture and will be maintained under configuration control by the UMAA Board. Section 5 content is automatically generated from this data model as are other automated products such as IDL that are used for automated code generation.

3.2 Definitions

UMAA ICDs follow the UCS terminology definitions found in the UCS Architecture Description v2.4. The normative (required) implementation to satisfy compliance with a UMAA ICD is to provide service and interface specification compliance. Components may group services and their required interfaces in any manner so long as every service meets its interface specifications. Figure 3 shows a particular grouping of services into components. The interfaces are represented by the blue and green lines and may represent 1 or more independent input and output interfaces for each service. The implementation of the service into software components is left up to the individual system development. Compliance is satisfied at the individual service level. Given this context, section 5 correspondingly defines services with their interfaces and not components.

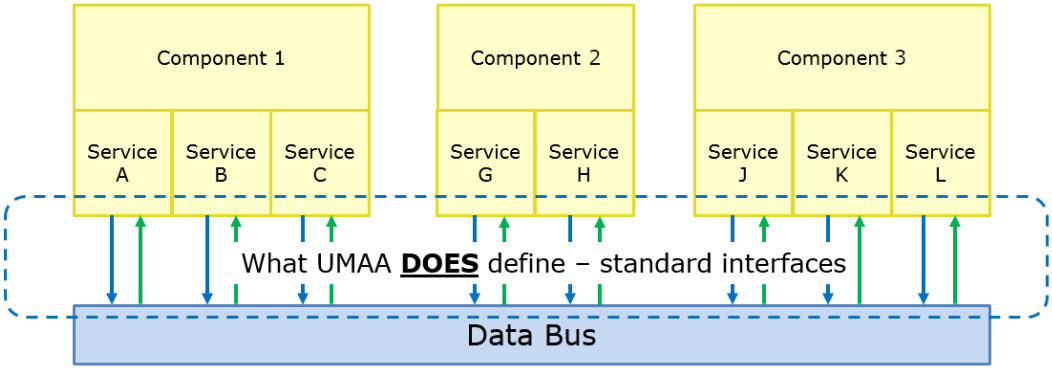

Figure 3: Services and Interfaces Exposed on the UMAA Data Bus

Services may use other services within this ICD or in other UMAA defined ICDs in order to provide their capability. Additionally, components for acquisition and development may span ICDs. An example of this would be a vehicle control system on a UMV. The control of the vehicle would be found in the Maneuver Operations ICD. However, an Inertial Navigation Unit (INU) that gives dynamic vehicle status is found in the Situational Awareness ICD. These are often organic to a vehicle and in that case are provided together with the vehicle as a component.

3.3 Data Distribution Service (DDS™)

The data bus supporting autonomy messaging as depicted in figure 3 is implemented via DDS™. DDS is a middleware protocol and API standard for data-centric connectivity from the Object Management Group (OMG). It integrates the components of a system together, providing low-latency data connectivity, extreme reliability, and a scalable architecture. In a distributed system, middleware is the software layer that lies between the operating system and applications. It enables the various components of a system to more easily communicate and share data. It simplifies the development of distributed systems by letting software developers focus on the specific purpose of their applications rather than the mechanics of passing information between applications and systems. The DDS specification is fully described in free reference material on the OMG website and there are both open source and commercially available implementations.

3.4 Naming Conventions

UMAA services are modeled within the UCS Architecture under the Multi-Domain Extension (MDE). The UCS Architecture uses SoaML concepts of participant, serviceInterface, service port and request port to describe the interfaces that make up a service and show how the service is used. Each service defines the capability it provides as well as required interfaces. Each interface consists of an operation that accepts a single message (A SoaML MessageType). In SoaML, a MessageType is a defined as a unit of information exchanged between participant Request and Service ports via ServiceInterfaces. Instances of a MessageType are passed as parameters in ServiceInterface operations. (UCSArchitecture,ArchitectureTechnicalGovernance)

In order to promote commonality across service definitions, a common way of naming services and their set of operations and messages has been adopted for defining services within UCS-MDE. The convention uses the Service Base Name (SBN) and an optional Function Name (FN) to derive all service names and their associated operations and messages. As this is meant to be a guide, services might not include all of the defined operations and messages and their names might not follow the convention where a more appropriate name adds clarity.

Furthermore services in UMAA will not be broken up as indicated below when all parts of the service capabilities are required for the service to be meaningful (such as ResourceAllocation).

Additionally, note that for UMAA not all operations defined in UCS-MDE result in a message being published to the DDS bus, e.g., since DDS uses publish/subscribe, most query operations result in a subscription to a topic and do not actually publish the associated request message. In the case of cancel commands, there is no associated implementation of the cancel<SBN><FN>CommandStatus as it is just the intrinsic response of the DDS dispose function so it is essentially a NOOP in implementation. The conventions used to define UCS-MDE services are as follows:

Service Name
- <SBN>Config
- <SBN>Control
- <SBN>Specs
- <SBN>Status

where the SBN should be descriptive of the task or information provided by the service.

Table 5: Service Requests and Associated Responses

	Service Requests (Inputs)	**Service Responses (Outputs)**
Config	query<SBN><FN>Config	report<SBN><FN>Config
Control	set<SBN><FN> query<SBN><FN>CommandAck cancel<SBN><FN>Command query<SBN><FN>ExecutionStatus	report<SBN><FN>CommandStatus report<SBN><FN>CommandAck report<SBN><FN>CancelCommandStatus report<SBN><FN>ExecutionStatus
Specs	query<SBN><FN>Specs	report<SBN><FN>Specs
Status	query<SBN><FN>	report<SBN><FN>

Service Requests (operation:message)
- query<SBN><FN>Config:<SBN><FN>ConfigRequestType[1]
- set<SBN><FN>:<SBN><FN>CommandType
- query<SBN><FN>CommandAck:<SBN><FN>CommandAckRequestType[1]
- cancel<SBN><FN>Command:<SBN><FN>CancelCommandType
- query<SBN><FN>ExecutionStatus:<SBN><FN>ExecutionStatusRequestType[1]
- query<SBN><FN>Specs:<SBN><FN>SpecsRequestType[1]
- query<SBN><FN>:<SBN><FN>RequestType [1] [2]

[1]These message types are required for compatibility with the UCS model but are not used by the UMAA specification.
[2]At this time there are no Requests in the specification but when they have been added, this will be the message format.

Service Responses (operation:message)
report<SBN><FN>Config:<SBN><FN>ConfigReportType
report<SBN><FN>CommandStatus:<SBN><FN>CommandStatusType
report<SBN><FN>CommandAck:<SBN><FN>CommandAckReportType
report<SBN><FN>CancelCommandStatus:<SBN><FN>CancelCommandStatusType
report<SBN><FN>ExecutionStatus:<SBN><FN>ExecutionStatusReportType
report<SBN><FN>Specs:<SBN><FN>SpecsReportType
report<SBN><FN>:<SBN><FN>ReportType

where,

- Config (Configuration) Report – the setup of a resource for operation of a particular task. Attributes may be static or variable. Examples include: maximum RPM allowed, operational sonar frequency range allowed, maximum allowable radio transmit power.
- Command Status – the current state of a particular command (either control or configuration)
- Command – the ability to influence or direct the behavior of a resource during operation of a particular task. Attributes are variable. Examples include a vehicleâ€™s speed, engine RPM, antenna raising/lowering, controlling a light or gong.
- Command Ack (Acknowledgement) Report – the command currently being executed.
- Cancel – the ability to cancel a particular command that has been issued.
- Execution Status Report – the status related to executing a particular command. Examples associated with a waypoint command include cross track error, time to achieve, distance remaining.
- Specs (Specifications) Report – a detailed description of a resource and/or its capabilities and constraints. Attributes are static. Examples include: maximum RPM of a motor, minimum frequency of a passive sonar sensor, length of the UMV, cycle time of a radar.
- Report – the current information provided by a resource. Examples include a vehicle speed, rudder angle, current waypoint, contact bearing.

3.5 Namespace Conventions

Each UMAA service and the messages under the service can be accessed through their appropriate UMAA namespace. The namespace reflects the mapping of a specific service to its parent ICD, and the parent ICD's mapping to the overall UMAA Design Description. For example:

Access the Primitive Driver service under Maneuver Operations:
UMAA::MO::PrimitiveDriver
Access the Feature Service under Situational Awareness:
UMAA::SA::Feature

The UMAA model uses common data types that are re-used through the model to define service interface topics, interface topics, and other common data topics. These data types are not intended to be directly utilized but for reference they can be accessed in the same manner:

Access the common UMAA Report Message Fields:
UMAA::UMAARpt
Access the common UMAA Position2D (i.e., latitude and longitude) structure:
UMAA::Measurement::Position2D

3.6 Cybersecurity

The UMAA standard addressed in this ICD is independent from defining specific measures to achieve Cybersecurity compliance. This UMAA ICD does not preclude the incorporation of security measures, nor does it imply or guarantee any level of Cybersecurity within a system. Cybersecurity compliance will be performed on a program specific basis and compliance testing is outside the scope of UMAA.

3.7 GUID algorithm

The UMAA standard utilizes the Globally Unique IDentifier (GUID), conforming to the variant defined in RFC 4122 (variant value of 2). Generators of GUIDs may generate GUIDs of any valid, RFC 4122-defined version that is appropriate for their specific use case and requirements. (Reference: A Universally Unique IDentifier (UUID) URN Namespace)

3.8 Large Sets

Some reports under the UMAA standard utilize Large Sets, which are unordered sets of related data. The purpose of a Large Set is to provide the ability to update one or more elements of the set without having to republish the entire set on the DDS bus and consuming more resources as a set is appended or updated. In a given DDS topic, each element of the set is tracked to its identifier through the use of the <service>SetID identifier (a key). Additionally, users will be able to trace an element in a set by its source attribute (a NumericGUID) to the Service Provider that is generating the report with this set.

When elements of the set are updated, the timestamp of the metadata must be updated as well to signal a change in the set. The element timestamp for the update must be later than the current metadata timestamp. Once the element is updated, the timestamp of the metadata must be updated to a time equal to or later than the timestamp of the individual element update. The set can be updated as a batch (multiple elements in a single "update cycle," as determined by the provider) provided the metadata timestamp is updated to a time that is no earlier than the the most recent timestamp of all element updates in the batch. This allows for a coarse synchronization: data elements with timestamps later than the current metadata timestamp can be assumed to be part of an in-progress update cycle. Consumers can choose to immediately act on those data individually or wait until the metadata timestamp is advanced beyond the element's timestamp to signal the complete update cycle has finished and consider the set as a whole.

4 Flow Control

4.1 Command / Response

This section defines the flow of control for command/response over the DDS bus. A command/response is used to control a specific service. While the exact names and processes will depend on the specific service and command being executed, all command/responses in UMAA follow a similar pattern. A notional "Function" command `FunctionCommand` is used in the following examples. As will be described in subsequent paragraphs, DDS publish/subscribe methods are used in implementations to issue commands and responses.

To direct a `FunctionCommand` at a specific Service Provider, UMAA includes a `destination` GUID in all commands. A Service Provider is required to respond to all `FunctionCommand`s where the `destination` is the same as the Service Provider's ID. The Service Consumer will also create a unique `sessionID` for the command when commanded. The `sessionID` is used to track the command execution as a key into other command-related messages. Service Provider and Service Consumer terminology in the following sections is adopted from the OMG Service-oriented architecture Modeling Language (SoaML).

To initialize, a Service Provider (controllable resource) subscribes to the `FunctionCommand` DDS topic. At startup or right before issuing a command, the Service Consumer (controlling resource) subscribes to the `FunctionCommandStatus` DDS topic. Optionally, the Service Consumer may also subscribe to the `FunctionCommandAckReport` to monitor which command is currently being executed, and the `FunctionExecutionStatusReport`, if defined for the Function service, that provides reporting on function-specific data status.

Both Service Providers and Service Consumers are required to recover or clean up any previous persisted commands on the bus during initialization.

To execute a command the Service Consumer publishes a `FunctionCommandType` to the DDS bus. The Service Provider will be notified and will begin processing the request. During each phase of processing, the Service Provider will provide updates to the Service Consumer via published updates to a related `FunctionCommandStatus` topic. Command responses are correlated to their originating command via the `sessionID`. Command status updates are provided in the command responses via the `commandStatus` field with additional details included in the `commandStatusReason` field. The Service Provider will also publish the current executing command to the `FunctionCommandAckReport` topic. When defined for the Function service, the Service Provider must also publish the `FunctionExecutionStatusReport` topic and update it as appropriate throughout the execution of the command.

The required state transitions for the `commandStatus` field are shown in Figure 4. Every command must transition through the states as defined. For example, it is a violation to transition from `ISSUED` to `EXECUTING` without transitioning through `COMMANDED`. Even in the case where there is no logic executing between the `ISSUED` and `EXECUTING` states the Service Provider is required to transition through `COMMANDED`. This ensures consistent behavior across different Service Providers, including those that do require the `COMMANDED` state.

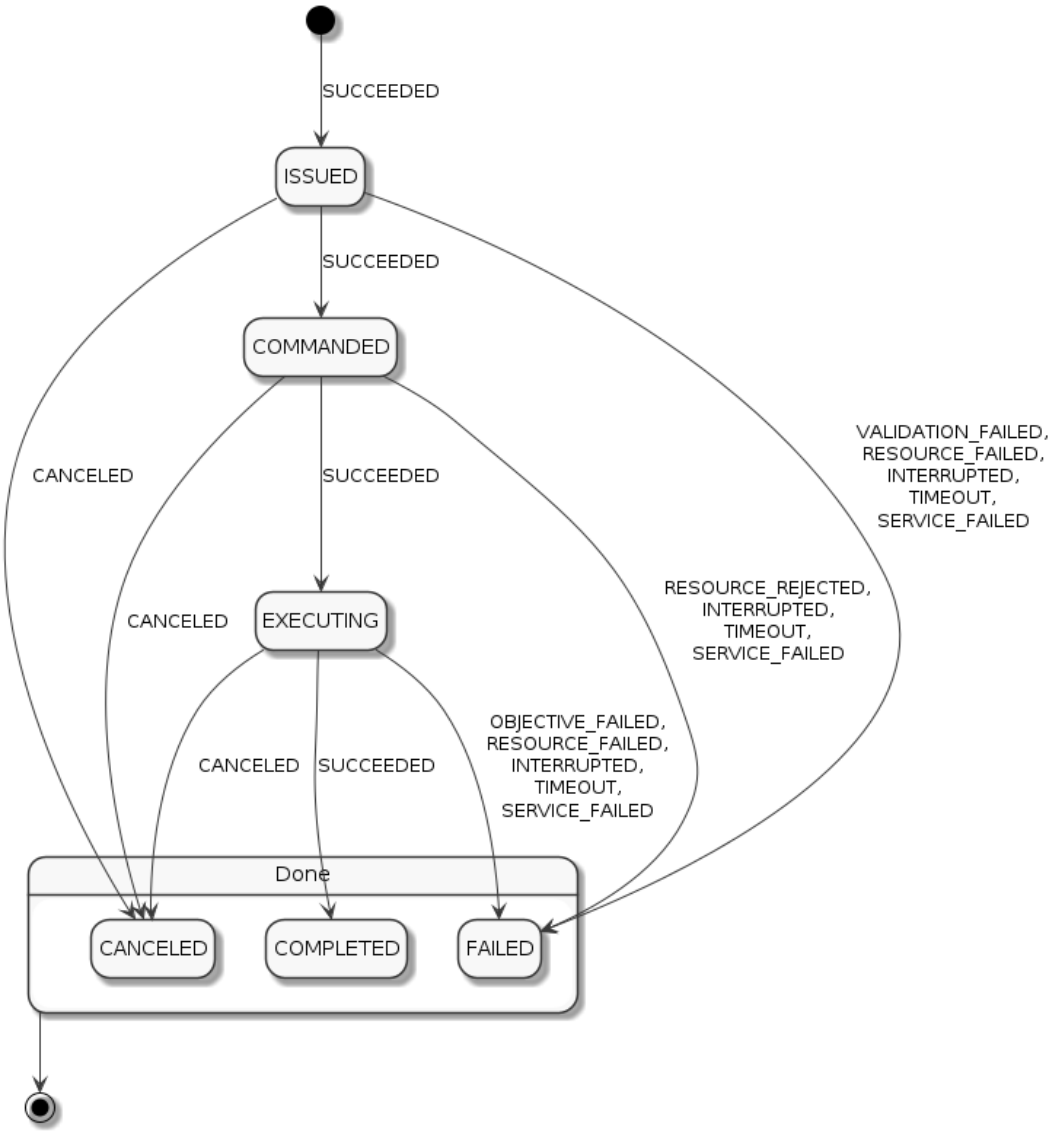

Figure 4: The state transitions of the `commandStatus` as commands are processed. Labels on the arrows represent valid `commandStatusReason` values for each transition.

In the following sections, the sequence diagrams demonstrate different exchanges between a Service Consumer and Service Provider. Within the diagrams, the dashed arrows represent implementation-specific communications that are outside of UMAA's scope. These sequence diagrams are just an example of one possible implementation. Other implementations may have different communication patterns between the Service Provider and the Resource or be implemented completely within the Service Provider process itself (no dependency on an external Resource). Likewise, the interactions between the User and Service Consumer may follow similiar or different patterns. However, the UMAA-defined exchanges with the DDS bus between the Service Consumer and Service Provider must happen in the order shown within the sequence diagrams.

4.1.1 High-Level Flow

The high-level flow of a command sequence is shown in Figure 5 and can be described as follows:

1. The Command Startup Sequence is performed
2. For each command to be executed
   (a) The Command Start Sequence is performed
   (b) The command is executed (sequence depends on the execution path, i.e., success, failure, or cancel)
   (c) The Command Cleanup Sequence is performed

3. The Command Shutdown Sequence is performed

The `ref` blocks will be defined in later sequence diagrams. Note that the duration of the system execution for any particular `FunctionCommandType` is defined by the combination of the Service Provider(s) and Service Consumer(s) in the system and my not be identical to the overall system execution duration. For example, providers may only be available to execute certain commands during specific phases of a misison or when certain hardware is in specific configurations. This Command Startup Sequence is not required to happend during a system startup phase. The only requirement is it must be completed by at least one Service Provider and one Service Consumer before any `FunctionCommandType` commands can be fully executed. Likewise, the Command Shutdown sequence may occur at anytime the `FunctionCommandType` will no longer be supported. There is no requirement the Command Shutdown Sequence only be performed during a system shutdown phase.

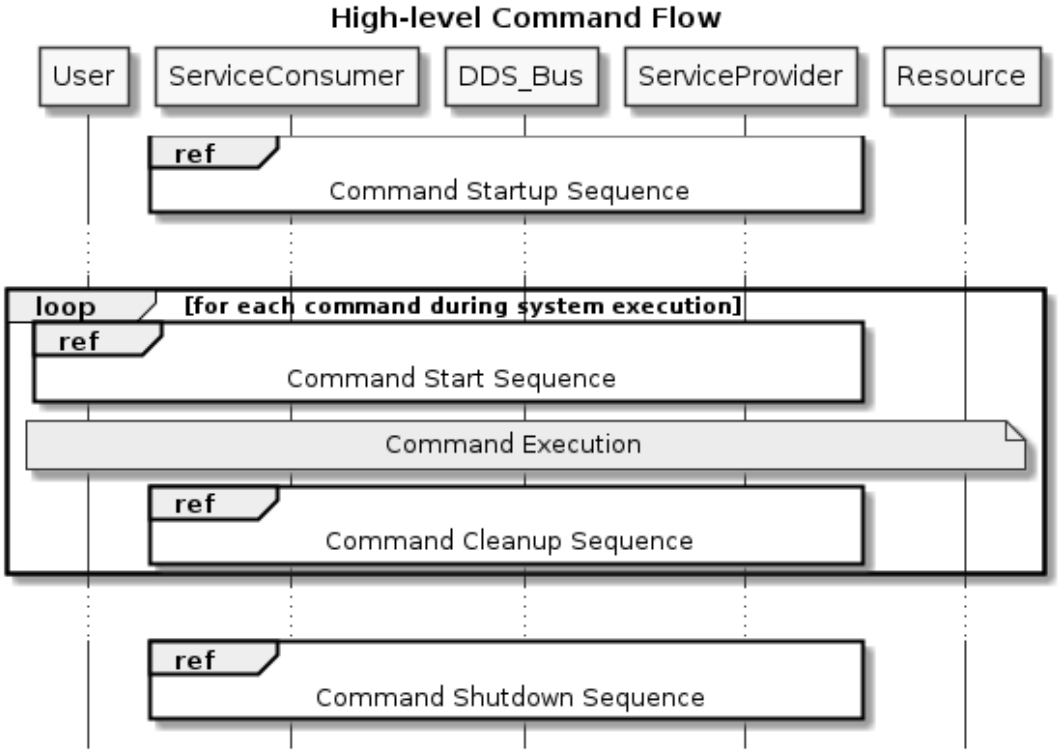

Figure 5: The sequence diagram for the high-level description of a command exeuction.

4.1.2 Command Startup Sequence

As part of initialization both the Service Provider and Service Consumer are required to perform a startup sequence. This startup prepares the Service Provider to execute commands and the Service Consumer to request commands and monitor the progress of those requested commands.

The Service Provider and Service Consumer can initialize in any order. Commands will not be completely executed until both have completed their initialization. The sequence diagram is shown in Figure 6.

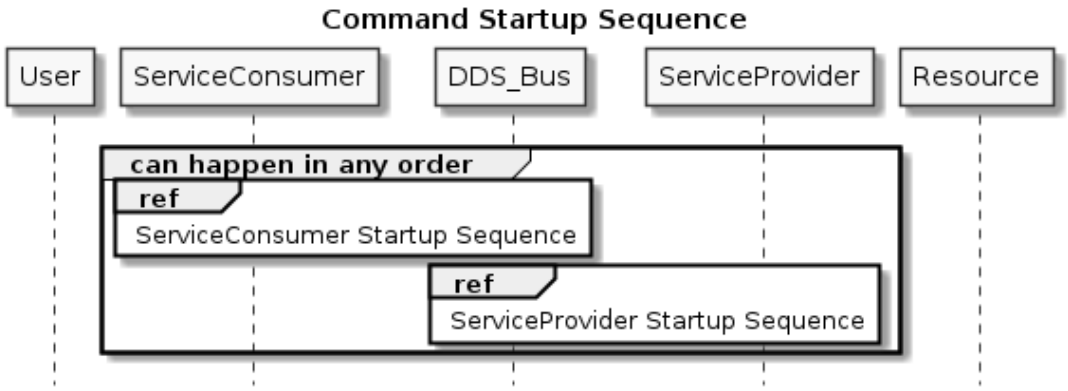

Figure 6: The sequence diagram for command startup.

4.1.2.1 Service Provider Startup Sequence

During startup the Service Provider is required to register as a publisher to the `FunctionCommandStatus`, `FunctionCommandAckReport`, and, if defined for the Function service, the `FunctionExecutionStatus` topics.

The Service Provider is also required to subscribe to the `FunctionCommand` topic to be notified when new commands are published.

Finally, the Service Provider is required to handle any existing `FunctionCommandType` commands persisted on the DDS bus with the Service Provider's ID. For each command, if the Service Provider can and wishes to recover, it can continue to execute the command. To obtain the last published state of the command, the Service Provider must subscribe to the `FunctionCommandStatusType`. The Service Provider will continue following the normal status update sequence, picking up from the last status on the bus. If the Service Provider cannot or choses not to continue processing the command, it must fail the command by publishing a `FunctionCommandStatus` with a `commandStatus` of `FAILED` and a `reason` of `SERVICE_FAILED`.

The Service Provider Startup sequence is shown in Figure 7.

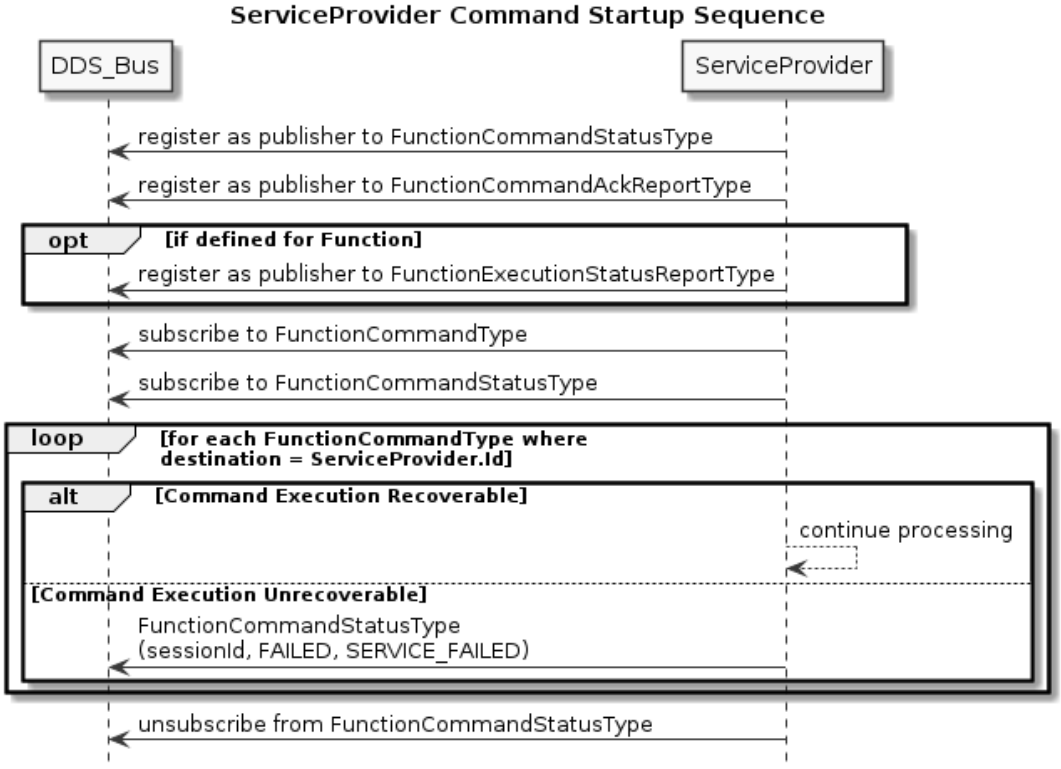

Figure 7: The sequence diagram for command startup for Service Providers.

4.1.2.2 Service Consumer Startup Sequence

During startup the Service Consumer is required to register as a publisher of the `FunctionCommandType`.

The Service Consumer is also required to subscribe to the `FunctionCommandStatusType` to monitor the execution of any published commands. The Service Consumer can optionally register for the `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` if it desires to track additional status of the execution of commands.

Finally, the Service Consumer is required to handle any existing `FunctionCommandType` commands persisted on the DDS bus with this Service Consumer's ID. To find existing `FunctionCommandType`s on the bus, it must first subscribe to the topic. If the Service Consumer can and wishes to recover, it can continue to monitor the execution of the command. If the Service Consumer cannot or choses not to continue the execution of the command, it must cancel the command via the normal command cancel method.

The Service Consumer Startup sequence is shown in Figure 8.

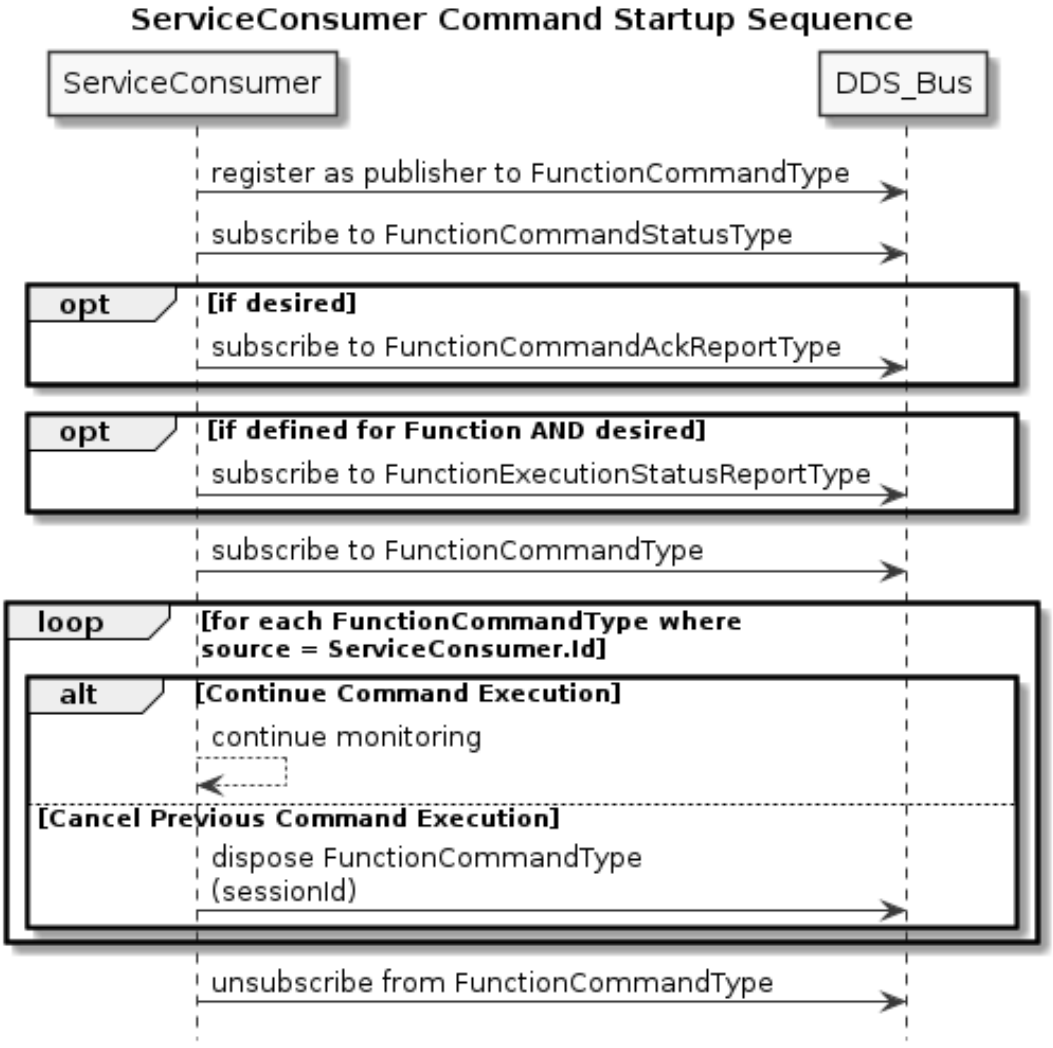

Figure 8: The sequence diagram for command startup for Service Consumers.

4.1.3 Command Execution Sequences

Once both the Service Provider and Service Consumer have performed the startup sequence, the system is ready be begin issuing and executing commands.

4.1.4 Command Start Sequence

The initial start sequence to execute a single command follows this pattern:

1. The User of the Service Consumer issues a request for a command to be executed.
2. The Service Consumer publishes the `FunctionCommandType` with a unique session ID, the source ID of the Service Consumer and the destination ID of the desired Service Provider.
3. The Service Provider, upon notification of the new `FunctionCommandType`, publishes a new `FunctionCommandStatusType` with the same session ID as the new `FunctionCommandType` and the status of `ISSUED` and reason of `SUCCEEDED` to notify the Service Consumer it has received the new command.

The Command Start Sequence is shown in Figure 9. This pattern will be repeated each time a new command is requested. After the Command Start Sequence, the sequence can take different paths depending on the actual execution of the command. Some possible paths are detailed in the following sections, but they do not enumerate all of the possible execution paths. Other paths (e.g., an objective failing) will follow a similiar pattern to other failures; all are required to follow the state diagram shown in Figure 4 and eventually end with the Command Cleanup Sequence (as shown in Figure 15).

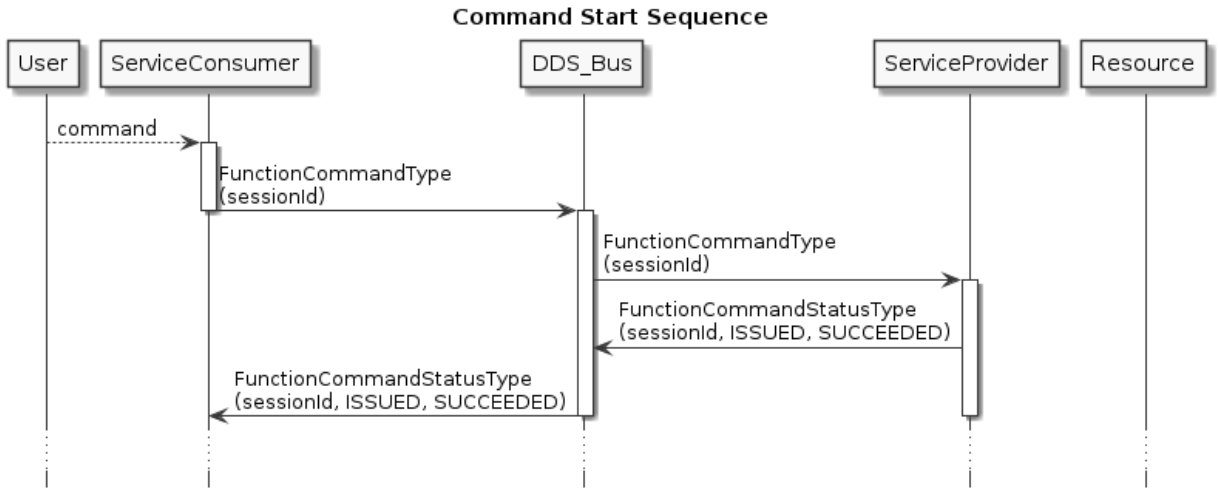

Figure 9: The sequence diagram for the start of a command execution.

4.1.4.1 Command Execution Once a Service Provider starts to process a command, the Command Execution sequence is:

1. The Service Provider publishes a `FunctionCommandAckReportType` with matching session ID and parameters as the `FunctionCommandType` it is starting to process.
2. The Service Provider performs any validation and negotiation with backing resources as necessary. Once the command is ready to be executed the Service Provider publishes a `FunctionCommandStatusType` with a status `COMMANDED` and reason `SUCCEEDED` to notify the Service Consumer that the command has been validated and commanded to start execution.
3. Once the command has begun executing the Service Provider publishes a `FunctionCommandStatusType` with a status `EXECUTED` and reason `SUCCEEDED` to notify the Service Consumer that the command has been validated and commanded to start.
4. If the Function has a defined `FunctionExecutionStatusReportType`, the Service Provider must publish a new instance with matching session ID as the associated `FunctionCommandType`. The `FunctionExecutionStatusReportType` must be updated by the Service Provider throughout the execution as dictated by the defintions of the command-specific attributes in the execution status report.

The command execution sequence is shown in Figure 10. This sequence holds until the command completes execution.

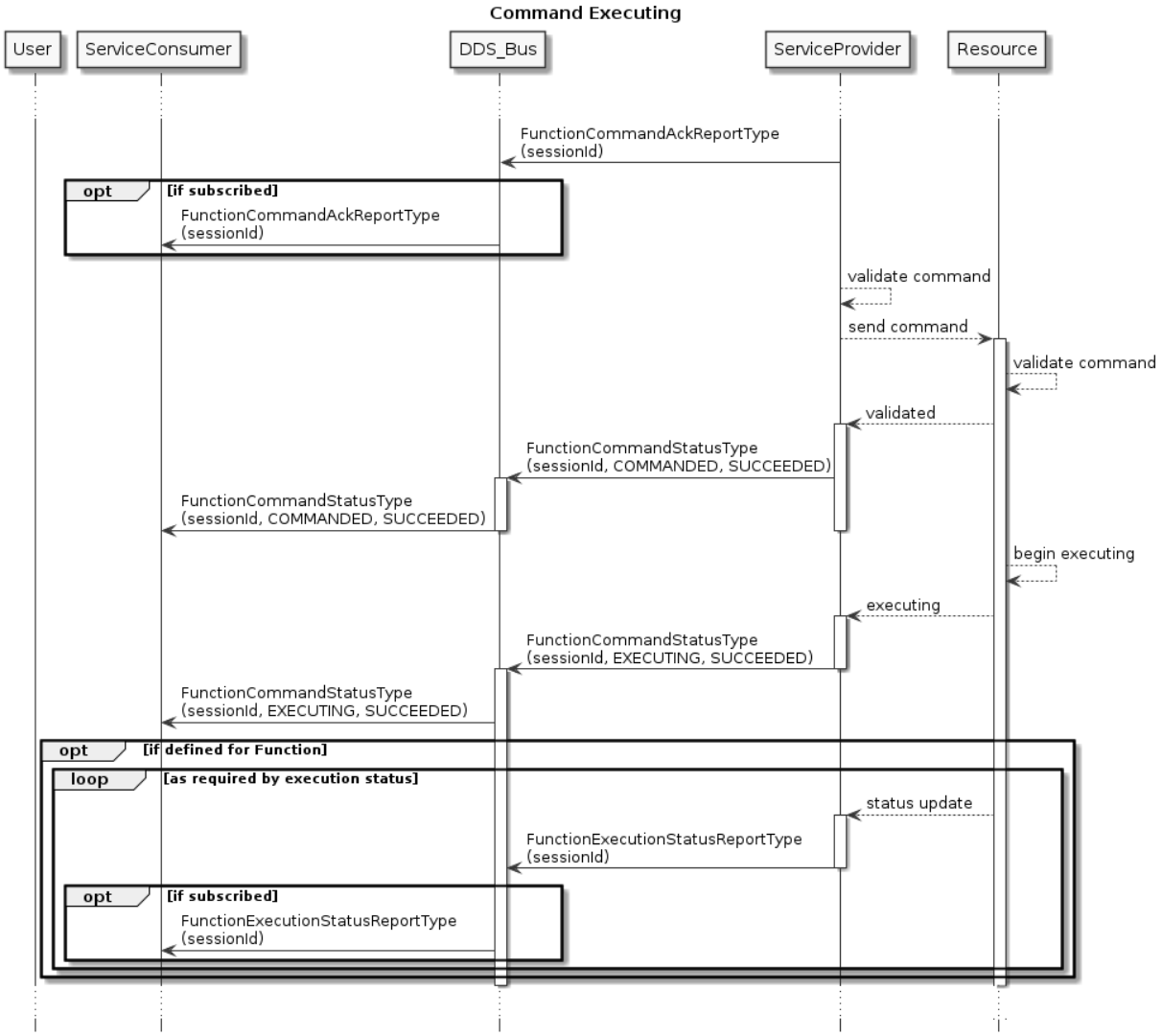

Figure 10: The beginning sequence diagram for a command execution.

The normal successful conclusion of a command being executed in some cases is initiated by the Service Consumer (an endless GlobalVector command concluded by canceling it) and in other cases is initiated by the Service Provider (a GlobalWaypoint commanded concluded by reaching the last waypoint). Unless otherwise explicitly stated, it is assumed the Service Provider will be able to identify the successful conclusion of a command. In the cases where commands are defined to be indeterminate the Service Consumer must cancel the command when the Service Consumer no longer desires the command to be executed.

4.1.4.2 Command Execution Success When the Service Provider determines a command has succesfully completed, it must update the associated `FunctionCommandStatusType` with as status of `COMPLETED` and reason of `SUCCEEDED`. This signals to the Service Consumer the command has completed successfully.

The Command Execution Success sequence is shown in Figure 11.

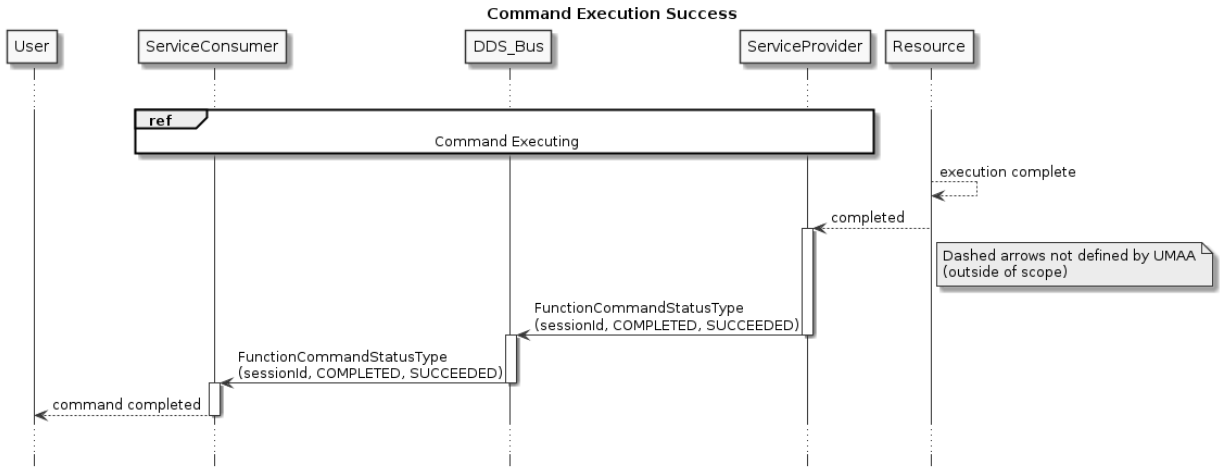

Figure 11: The sequence diagram for a command that completes successfully.

4.1.4.3 Command Execution Failure The command may fail to complete for any number of reasons including software errors, hardware failures, or unfavorable environmental conditions. The Service Provider may also reject a command for a number of reasons including inability to perform the task, malformed or out of range requests, or a command being interrupted by a higher priority process. In all cases the Service Provider must publish a `FunctionCommandStatusType` with an identical `sessionID` as the originating `FunctionCommandType` with a status of `FAILED` and the reason that reflects the cause of the failure (`VALIDATION_FAILED, SERVICE_FAILED, OBJECTIVE_FAILED,` etc).

The following figures provide examples of cases where a command has failed.

In the first example, the backing Resource has failed and the Service Provider in unable to communicate with it. In this case the Service Provider will report a `FunctionCommandStatusType` with a status of `FAILED` and a reason of `RESOURCE_FAILED`. This is shown in Figure 12.

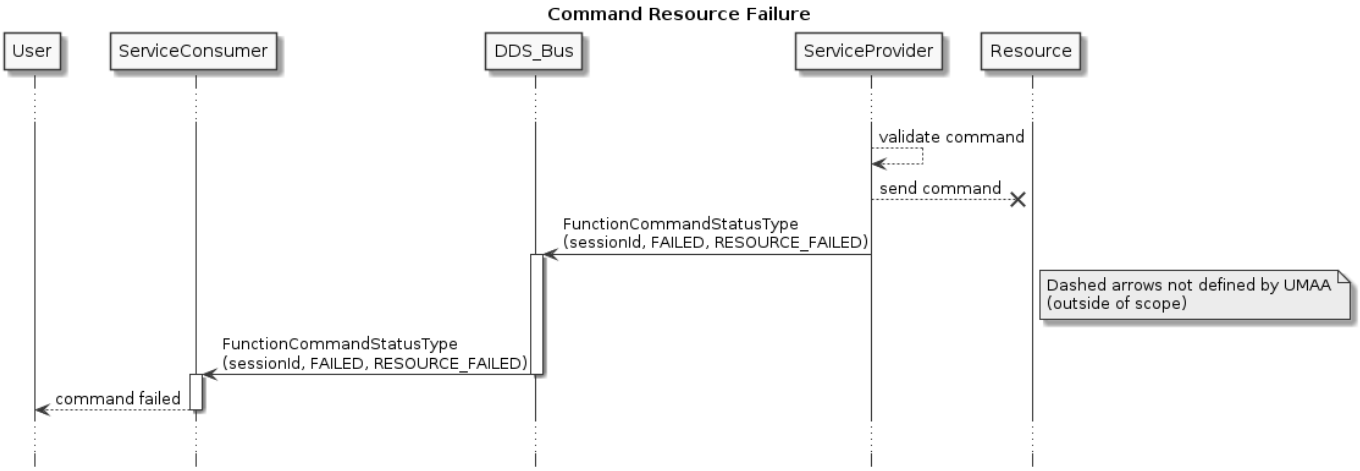

Figure 12: The sequence diagram for a command that fails due to Resource failure.

In the second example, the Resource takes too long to response, so the Service Provider cancels the request and reports a `FunctionCommandStatusType` with a status of `FAILED` and a reason of `TIMEOUT`. This is shown in Figure 13.

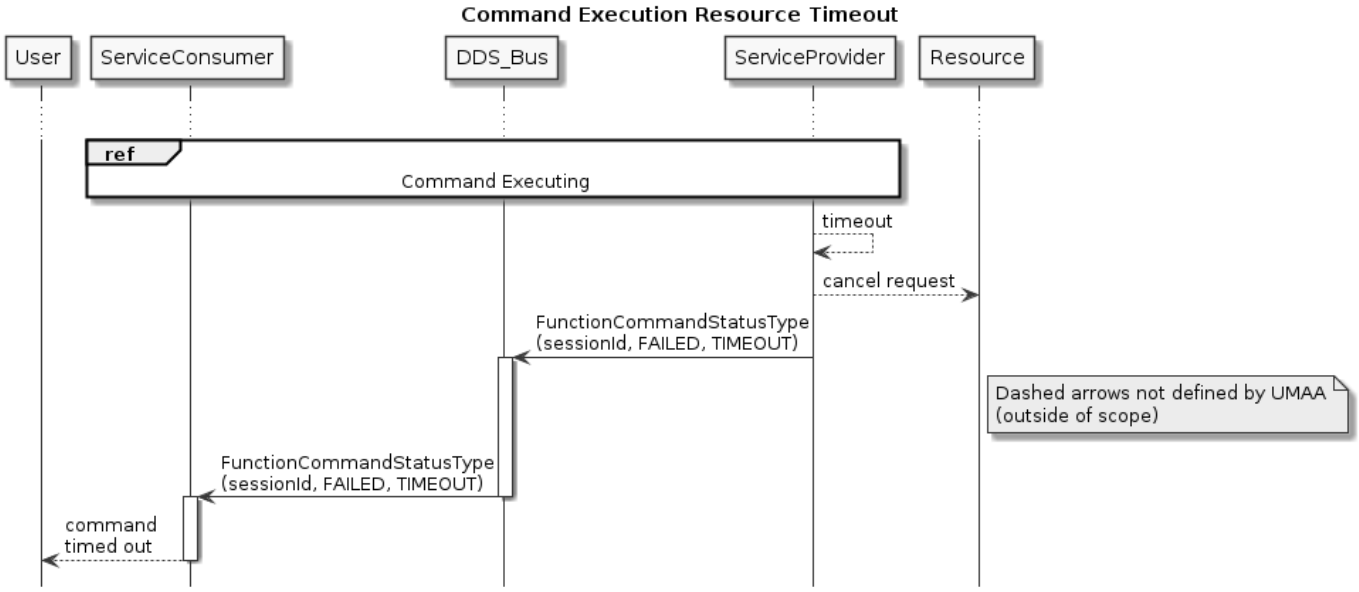

Figure 13: The sequence diagram for a command that times out before completing.

Other failure conditions will follow a similar pattern: when the failure is recognized, the Service Provider will publish a `FunctionCommandStatusType` with a status of `FAILED` and a reason that reflect the cause of the failure.

4.1.4.4 Command Canceled The Service Consumer may decide to cancel the command before processing is finished. To signal a desire to cancel a command, the Service Consumer disposes the existing `FunctionCommandType` from the DDS bus before the execution is complete. When notified of the command disposal, if the Service Provider is able to cancel the command it should respond to the Service Consumer with a `FunctionCommandStatusType` with both the status and reason as `CANCELED` and then dispose the `FunctionCommandStatusType` and `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` from the bus. This is shown in Figure 14. If the command cannot be canceled the Service Provider can continue to update the command status until the execution is completed, reporting `FunctionCommandStatusType` with a status of `COMPLETED` and a reason of `SUCCEEDED`, and then dispose the `FunctionCommandStatusType` and `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` from the DDS bus.

There is no new unique specific status message response to a cancel command from the Service Provider. The cancel command status can be inferred through the corresponding `FunctionCommandStatusType` status and reason updates.

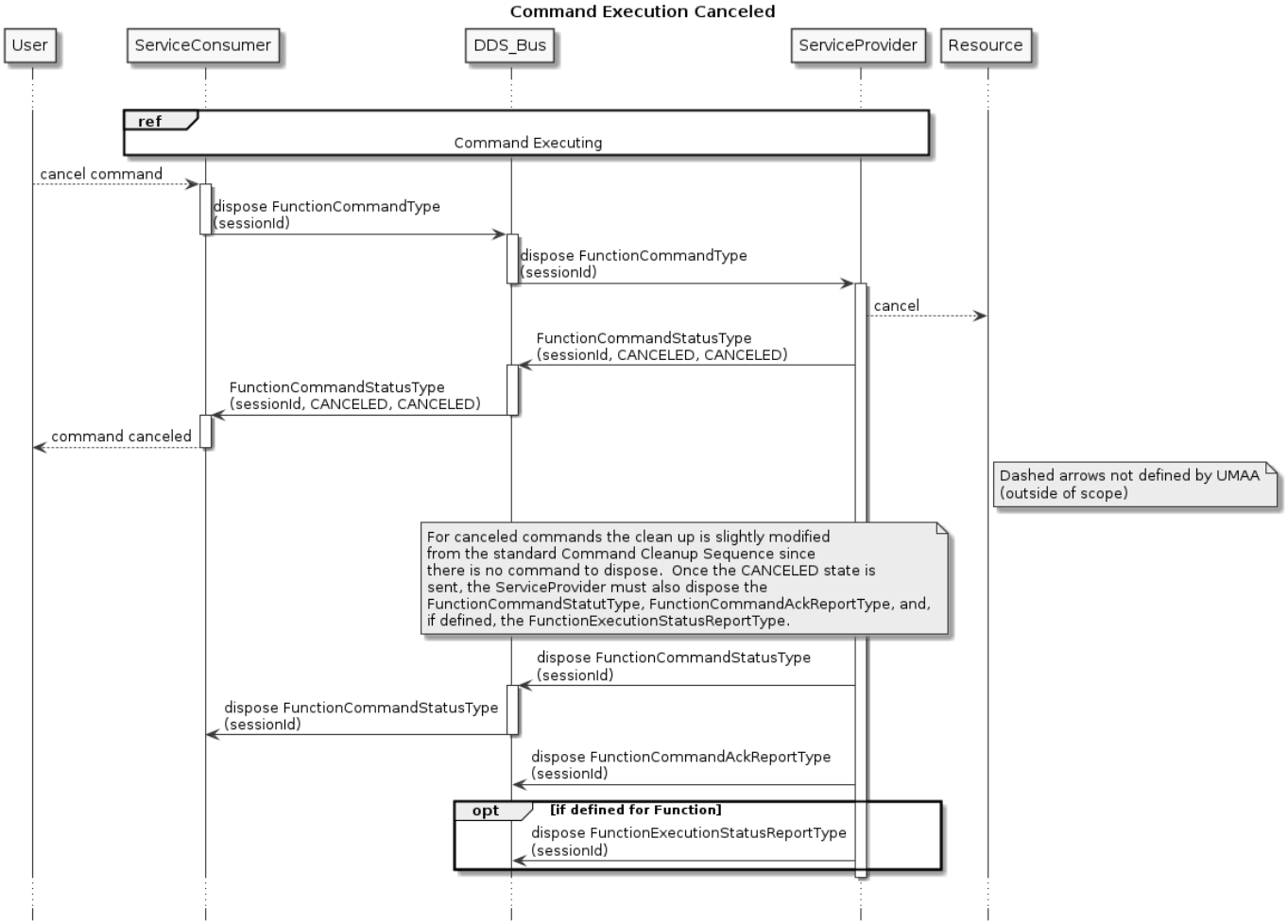

Figure 14: The sequence diagram for a command that is canceled by the Service Consumer before the Service Provider is able to complete it.

4.1.5 Command Cleanup

The Service Consumer and Service Provider are responsible for disposing corresponding data published to the DDS bus when the command is no longer active. With the exception of a canceled command, the signal that a `FunctionCommandType` can be disposed is when the `FunctionCommandStatusType` reports a terminal state (`COMPLETED` or `FAILED`)[3]. In turn, the signal that a `FunctionCommandStatusType`, `FunctionCommandAckReportType`, and if defined for the Function service, the `FunctionExecutionStatusReportType` can be disposed is when the corresponding `FunctionCommandType` has been disposed. This is shown in Figure 15.

[3]While `CANCELED` is also a terminal state, `CANCELED` command cleanup is handled specially as part of the cancelling sequence and, as such, does not need to be handled here.

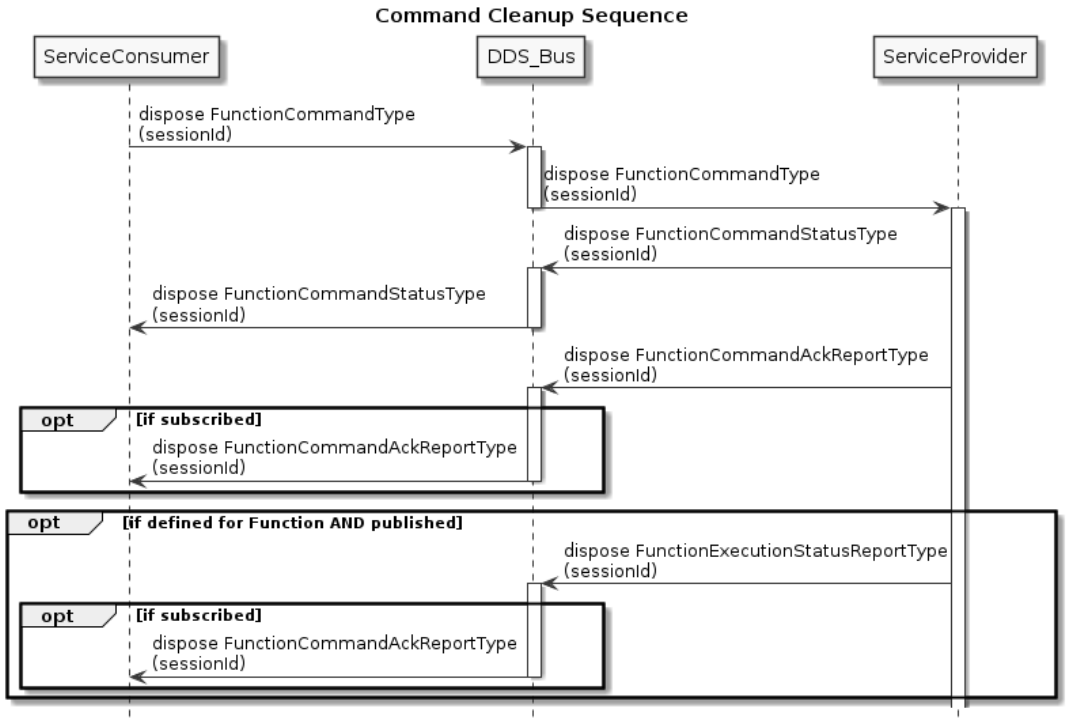

Figure 15: The sequence diagram showing cleanup of the bus when a command has been completed and the Service Consumer no longer wishes to maintain the commanded state.

4.1.6 Command Shutdown Sequence

As part of shutdown both the Service Provider and Service Consumer are required to perform a shutdown sequence. This shutdown cleans up resources on the DDS bus and informs the system that the Service Provider and Service Consumer are no longer available.

The Service Provider and Service Consumer can shutdown in any order. The sequence diagram is shown in Figure 16.

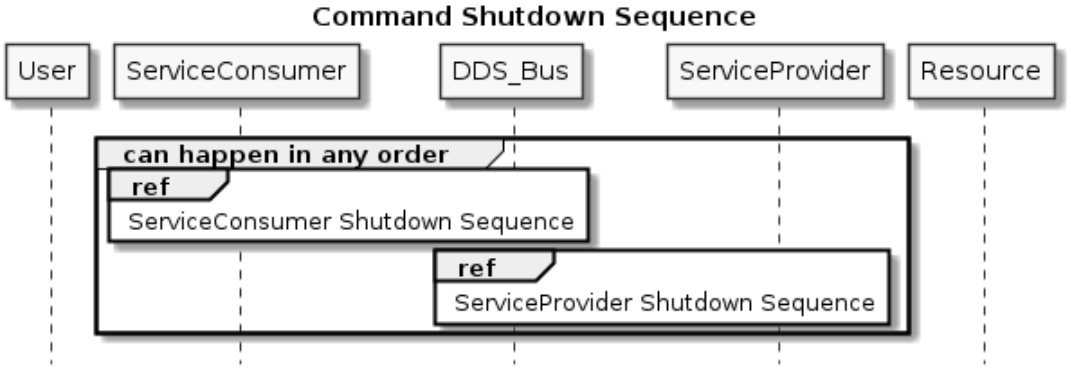

Figure 16: The sequence diagram for command shutdown.

4.1.6.1 Service Provider Shutdown Sequence During shutdown the Service Provider is required to fail any incomplete requests and then unregisters as a publisher of the `FunctionCommandStatusType`, `FunctionCommandAckReportType`, and, if defined for the Function service, the `FunctionExecutionStatusReportType`.

The Service Provider is also required to unsubscribe from the `FunctionCommandType`.

The Service Provider Shutdown sequence is shown in Figure 17.

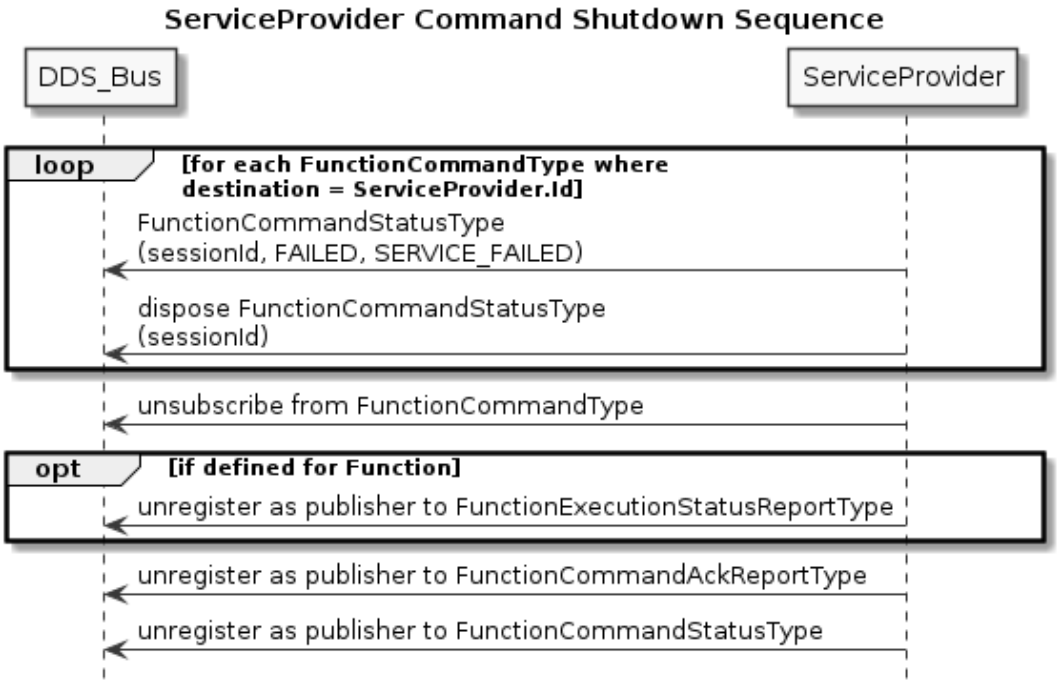

Figure 17: The sequence diagram for command shutdown for Service Providers.

4.1.6.2 Service Consumer Shutdown Sequence During shutdown the Service Consumer is required to cancel any incomplete requests and then unregister as a publisher of the `FunctionCommandType`.

The Service Consumer is also required to unsubscribe from the `FunctionCommandStatusType`, the `FunctionCommandAckReportType` if subscribed, and the `FunctionExecutionStatusReportType` if defined for the Function service and subscribed.

The Service Consumer Shutdown sequence is shown in Figure 18.

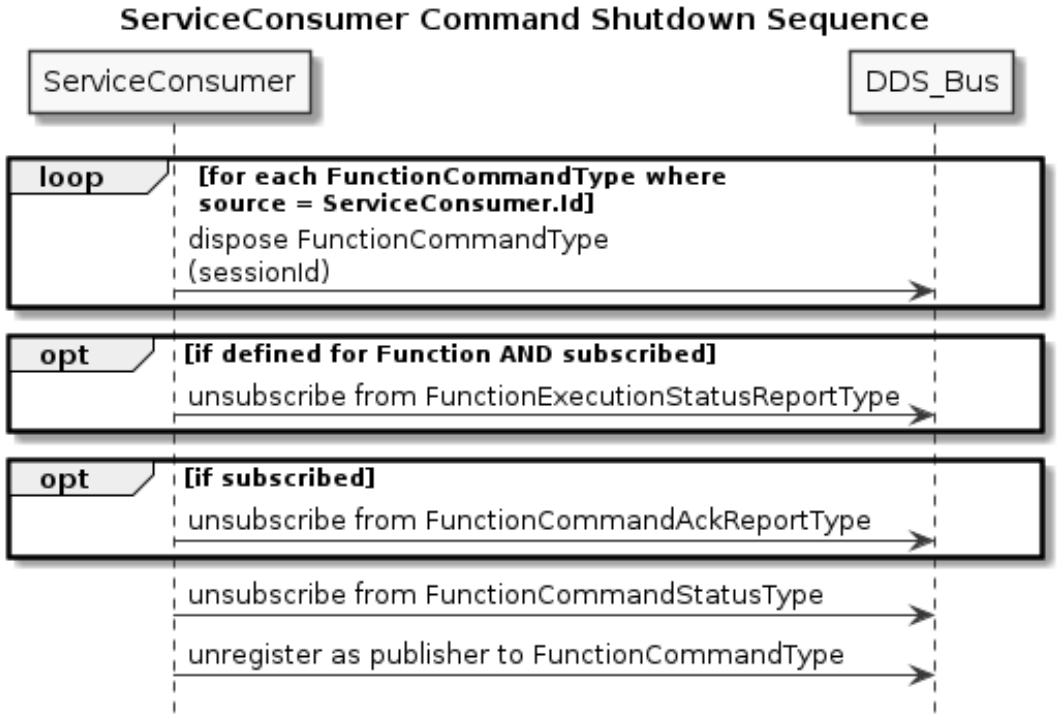

Figure 18: The sequence diagram for command shutdown for Service Consumers.

4.2 Request / Reply

This section defines the flow of control for request/reply over the DDS bus. A request/reply is used to obtain data or status from a specific Service Provider.

A Service Provider is required to reply to all requests it receives. In the case of requests with no query data, this is accomplished via a DDS subscribe. In the case of a request with associated query data, a message with the query data must be published by the requester. To direct a request at a specific Service Provider or set of services UMAA defines a `destination` GUID as part of requests.

In the following sections, the sequence diagrams demonstrate different exchanges between a Service Consumer and Service Provider. Within the diagrams, the dashed arrows represent implementation-specific communications that are outside of UMAA's scope. Additionally, these sequence diagrams are just an example of one possible implementation. Other implementations may have different communication patterns between the Service Provider and the Resource or be implemented completely within the Service Provider process itself (no external Resource). In all implementations, however, UMAA-defined exchanges with the DDS bus between the Service Consumer and Service Provider must happen in the order shown within the sequence diagrams.

4.2.1 Request/Reply without Query Data

In the case where there is no specific query data (i.e., the service is always just providing the current data to the bus) the sequence of exchanges is show in Figure 19.

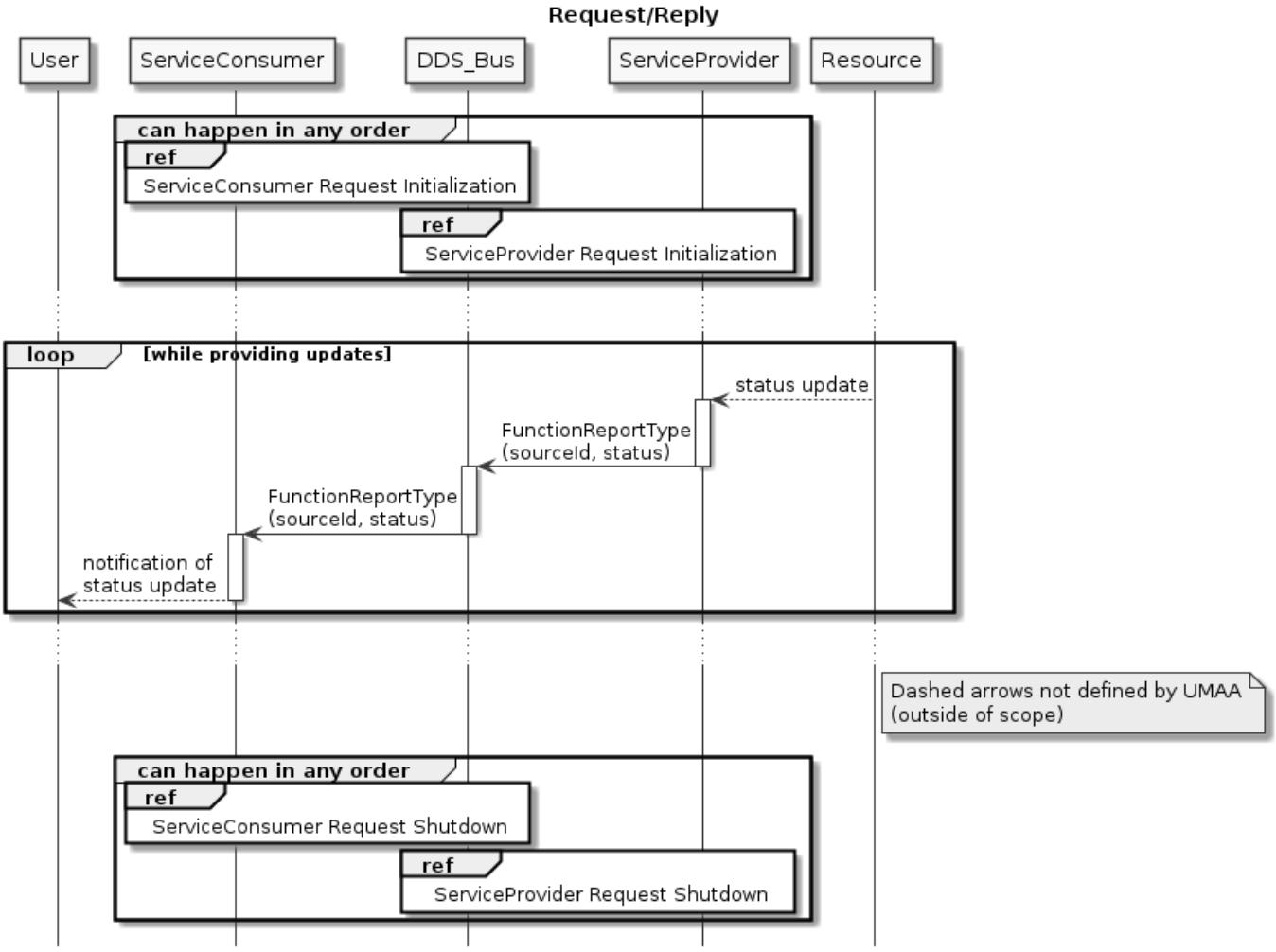

Figure 19: The sequence diagram for a request/reply for report data that does not require any specific query data.

4.2.1.1 Service Provider Startup Sequence The Service Provider registers as a publisher of `FunctionReportTypes` to be able to respond to requests. The Service Provider must also handle reports that exist on the bus from a previous instantiation, either by providing an immediate update or, if the status is unrecoverable, disposing of the old `FunctionReportType`. This is shown in Figure 20.

As `FunctionReportType` updates are required (either through event-driven changes or periodic updates), the Service Provider publishes the updated data. The DDS bus will deliver the updates to the Service Consumer.

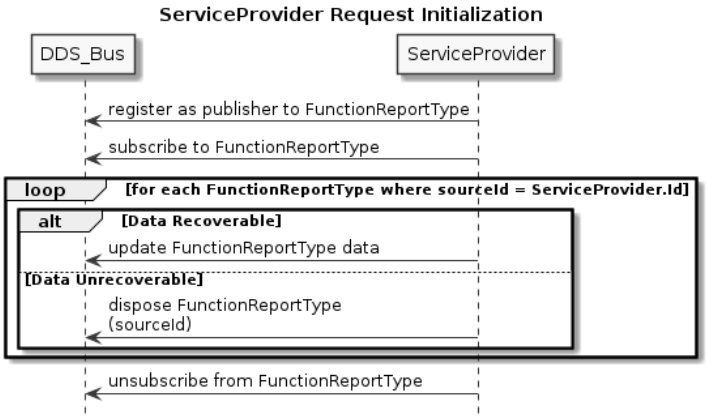

Figure 20: The sequence diagram for initialization of a Service Provider to provide FunctionReportTypes.

4.2.1.2 Service Consumer Startup Sequence The Service Consumer subscribes to the `FunctionReportType` to signal an outstanding request for updates. This is shown in Figure 21.

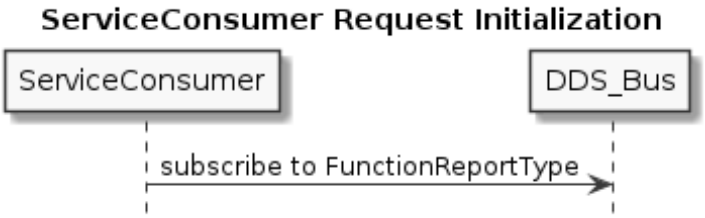

Figure 21: The sequence diagram for initialization of a Service Consumer to request FunctionReportTypes.

4.2.1.3 Service Provider Shutdown To no longer provide `FunctionReportTypes`, the Service Provider disposes the `FunctionReportType` and unregisters as a publisher of the data as shown in Figure 22.

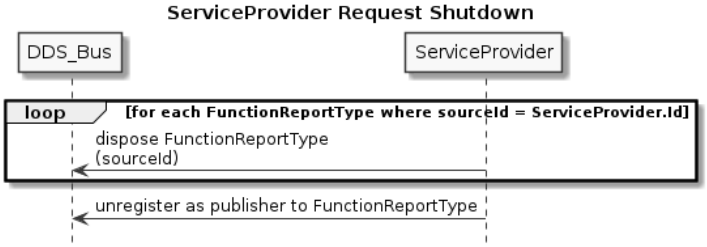

Figure 22: The sequence diagram for shutdown of a Service Provider.

4.2.1.4 Service Consumer Shutdown To no longer request `FunctionReportTypes`, the Service Consumer unsubscribes from `FunctionReportType` as shown in Figure 23.

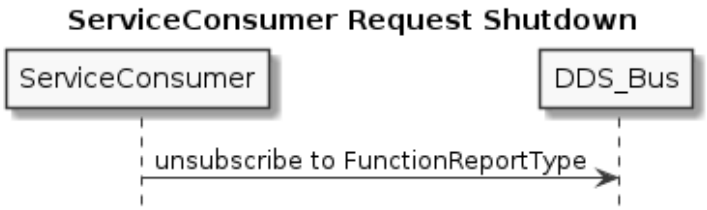

Figure 23: The sequence diagram for shutdown of a Service Consumer.

4.2.2 Request/Reply with Query Data

Currently UMAA does not define any request/reply interactions with query data, but it is expected some will be defined. When defined, this section will be expanded to describe how they must be used.

5 Sensor and Effector Management (SEM) Services and Interfaces

5.1 Services and Interfaces

The interfaces in the following subsections describe how each UCS-UMAA topic is defined by listing the name, namespace, and member attributes. The "name" corresponds with the message name of a given service interface. The "namespace" defines the scope of the "name" where similar commands are grouped together. The "member attributes" are fields that can be populated with differing data types, e.g. a generic "depth" attribute could be populated with a double data value. Note that using a UCS-UMAA "Topic Name" requires using the fully-qualified namespace plus the topic name.

Each interface topic is referenced by a UMAA service and is defined as either an input or output interface.
Attributes ending in one or more asterisk(s) denote the following:
* = Key (annotated with @key in IDL file, vendors may use different notation to indicate a key field)
†= Optional (annotated with @optional in IDL file, vendors may use different notation to indicate an optional field)

Optional fields should be handled as described in the UMAA Compliance Specification.

Commands issued on the DDS bus must be treated as if they are immutable in UMAA and therefore if updated (treated incorrectly as mutable), the resulting service actions are indeterminate and flow control protocols are no longer guaranteed.

Operations without DDS Topics

The following operations are all handled directly by DDS. They are marked in the operations tables with a ⊕.

query<...> - all query operations are used to retrieve the correlated report message. For UMAA, this operation is accomplished through subscribing to the appropriate DDS topic.

cancel<...> - all cancel operations are used to nullify the current command. For UMAA, this operation is accomplished through the DDS dispose action on the publisher.

report<...>CancelCommandStatus - all cancel reports are included here to show completeness of the MDE model mapping to UMAA. For UMAA, this operation is not used.

Instead, the cancel status is inferred from the associated command status. If the cancel command is successful, the corresponding command will fail with a command status and reason of CANCELED. If the corresponding command status reports COMPLETED, then this cancel command has failed.

5.1.1 GPSFixControl

The purpose of this service is to command the unmanned underwater platforms to surface for updating its GPS position.

Table 6: GPSFixControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setGPSFix	reportGPSFixCommandStatus
queryGPSFixCommandAck⊕	reportGPSFixCommandAck
cancelGPSFixCommand⊕	reportGPSFixCancelCommandStatus⊕

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.1.1 reportGPSFixCommandAck

Description: This operation is used to report the current GPS Fix command.

Namespace: UMAA::SEM::GPSFixControl

Topic: GPSFixCommandAckReport

Data Type: GPSFixCommandAckReportType

Table 7: GPSFixCommandAckReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		

5.1.1.2 reportGPSFixCommandStatus

Description: This operation is used to report status of the GPS Fix command.

Namespace: UMAA::SEM::GPSFixControl

Topic: GPSFixCommandStatus

Data Type: GPSFixCommandStatusType

Table 8: GPSFixCommandStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatus		

5.1.1.3 setGPSFix

Description: This operation is used to command the unmanned underwater platform to surface to update its GPS position.

Namespace: UMAA::SEM::GPSFixControl

Topic: GPSFixCommand

Data Type: GPSFixCommandType

Table 9: GPSFixCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		

5.1.2 GPSFixStatus

The purpose of this service is to provide Global Positioning System (GPS) signal status.

Table 10: GPSFixStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryGPSFix⊕	reportGPSFix
queryGPSFixStatus⊕	reportGPSFixStatus

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.2.1 reportGPSFix

Description: This operation is used to report the current GPS signal status from the unmanned platform.

Namespace: UMAA::SEM::GPSFixStatus

Topic: GPSFixReport

Data Type: GPSFixReportType

Table 11: GPSFixReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
altitudeMSL	Altitude_MSL	The current altitude in mean sea level
GPSFixCommandStatus	GPSFixEnumType	The command status of GPS Fix
GPSLatLongValid	BooleanEnumType	Whether GPS position is valid, if not INS position is used.
magneticVariation	MagneticVariation	The current magnetic variation
position2DWithTime	Position2DTime	The UTC Epoch time of the last position update
SOG	GroundSpeed	The speed over ground from the GPS
trueCourse	Course_TrueNorth	The current true course
velocity	Velocity3D_PlatformNED	The current velocity

5.1.2.2 reportGPSFixStatus

Description: This operation is used to report the current GPS data from unmanned vehicle.

Namespace: UMAA::SEM::GPSFixStatus

Topic: GPSFixStatusReport

Data Type: GPSFixStatusReportType

Table 12: GPSFixStatusReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
carrierToNoise	sequence<Size_Numeral>	The current carrier to noise level
countDown	Count	The current countdown
fixValid	BooleanEnumType	The current invalid/valid fix
lowBkupBattery	BooleanEnumType	The current invalid/valid backup battery
navSolution	GPSNavigationSolutionEnumType	The current GPS selected for navigation
originPosition	Position3D_WGS84	The origin position
pCode	BooleanEnumType	The current invalid/valid p-code
timeOut	BooleanEnumType	The current invalid/invalid time-out

5.1.3 GPSStatus

The purpose of this service is to provide access to the capabilities and configuration of the GPS system, allowing the controlling component to query the GPS system for a particular operational profile. The actual transmission of the GPS data stream is outside the scope of this service.

Table 13: GPSStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryGPS⊕	reportGPS

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.3.1 reportGPS

Description: This operation is used to report the parameters of the gps system.

Namespace: UMAA::SEM::GPSStatus

Topic: GPSReport

Data Type: GPSReportType

Table 14: GPSReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
clock	GPSClockType	the gps clock
numberSatellitesInView	Count	the number of satellites in use
satellites	sequence<GPSSatelliteType>	The list of available satellites.

5.1.4 InertialSensorControl

The purpose of this service is to provide the control of the inertial sensor such as Inertial Navigation Unit (INU) or Inertial Measurement Unit (IMU) of the unmanned platform.

Table 15: InertialSensorControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setInertialSensor	reportInertialSensorCommandStatus
queryInertialSensorCommandAck⊕	reportInertialSensorCommandAck
cancelInertialSensorCommand⊕	reportInertialSensorCancelCommandStatus⊕

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.4.1 reportInertialSensorCommandAck

Description: This operation is used to report the commanded state of the inertial sensor of the unmanned platform.

Namespace: UMAA::SEM::InertialSensorControl

Topic: InertialSensorCommandAckReport

Data Type: InertialSensorCommandAckReportType

Table 16: InertialSensorCommandAckReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
state	InertialSensorStateEnumType	The desired state of the inertial sensor

5.1.4.2 reportInertialSensorCommandStatus

Description: This operation is used to report the status of the inertial sensor command.

Namespace: UMAA::SEM::InertialSensorControl

Topic: InertialSensorCommandStatus

Data Type: InertialSensorCommandStatusType

Table 17: InertialSensorCommandStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatus		

5.1.4.3 setInertialSensor

Description: This operation is used to set the state of the inertial sensor of the unmanned platform.

Namespace: UMAA::SEM::InertialSensorControl

Topic: InertialSensorCommand

Data Type: InertialSensorCommandType

Table 18: InertialSensorCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		
state	InertialSensorStateEnumType	The desired state of the inertial sensor

5.1.5 InertialSensorStatus

The purpose of this service is to report the operational status of the inertial sensor such as Inertial Navigation Unit (INU) or Inertial Measurement Unit (IMU) of the unmanned platform.

Table 19: InertialSensorStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryInertialSensor⊕	reportInertialSensor

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.5.1 reportInertialSensor

Description: This operation is used to report the current operational status of the inertial sensor of the unmanned platform.

Namespace: UMAA::SEM::InertialSensorStatus

Topic: InertialSensorReport

Data Type: InertialSensorReportType

Table 20: InertialSensorReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
state	InertialSensorStateEnumType	The state of the inertial sensor.
status	InertialSensorOpStatusEnumType	The operational status of the inertial sensor.

5.2 Common Data Types

Common data types define DDS types that are referenced throughout the UMAA model. These DDS types are considered common because they can be re-used as the data type for many attributes defined in service interface topics, interface topics, and other common data types. These data types are not intended to be directly published to/subscribed as DDS topics.

5.2.1 UCSMDEInterfaceSet

Namespace: UMAA::UCSMDEInterfaceSet

Description: Defines the common UCSMDE Interface Set Message Fields.

Table 21: UCSMDEInterfaceSet Structure Definition

Attribute Name	Attribute Type	Attribute Description
timeStamp	DateTime	The time at which the data was derived.

5.2.2 UMAACommand

Namespace: UMAA::UMAACommand

Description: Defines the common UMAA Command Message Fields.

Table 22: UMAACommand Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	NumericGUID	The unique identifier of the originating source of the command interface.
destination*	NumericGUID	The unique identifier of the destination of the command interface.
sessionID*	NumericGUID	The identifier of the session.

5.2.3 UMAAStatus

Namespace: UMAA::UMAAStatus

Description: Defines the common UMAA Status Message Fields.

Table 23: UMAAStatus Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	NumericGUID	The unique identifier of the originating source of the status interface.

5.2.4 UMAACommandStatusBase

Namespace: UMAA::UMAACommandStatusBase

Description: Defines the common UMAA Command Status Base Message Fields.

Table 24: UMAACommandStatusBase Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	NumericGUID	The unique identifier of the originating source of the command status interface.
sessionID*	NumericGUID	The identifier of the session.

5.2.5 UMAACommandStatus

Namespace: UMAA::UMAACommandStatus

Description: Defines the common UMAA Command Status Message Fields.

Table 25: UMAACommandStatus Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
commandStatus	CommandStatusEnumType	The status of the command
commandStatusReason	CommandStatusReasonEnumType	The reason for the status of the command
logMessage	StringLongDescription	Human-readable description related to response. Systems should not parse or use any information from this for processing purposes.

5.2.6 DateTime

Namespace: UMAA::Measurement::DateTime

Description: Describes an absolute time. Conforms with POSIX time standard (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.

Table 26: DateTime Structure Definition

Attribute Name	Attribute Type	Attribute Description
seconds	DateTimeSeconds	The number of seconds offset from the standard POSIX (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.
nanoseconds	DateTimeNanoSeconds	The number of nanoseconds elapsed within the current DateTimeSecond

5.2.7 Altitude_HAE

Namespace: UMAA::Common::Measurement::Altitude_HAE

Description: Altitude_HAE specifies the entity's height above the reference ellipsoid.

Table 27: Altitude_HAE Structure Definition

Attribute Name	Attribute Type	Attribute Description
altitude	EllipsoidalHeight	Specifies the entityâ€™s height above the reference ellipsoid.

5.2.8 Altitude_MSL

Namespace: UMAA::Common::Measurement::Altitude_MSL

Description: Altitude_MSL specifies the entity's height above the geoid.

Table 28: Altitude_MSL Structure Definition

Attribute Name	Attribute Type	Attribute Description
altitude	MSLHeight	Specifies the entity's height above the geoid.

5.2.9 GPSClockType

Namespace: UMAA::SEM::GPSStatus::GPSClockType

Description: Specifies the characteristics of a gps clock.

Table 29: GPSClockType Structure Definition

Attribute Name	Attribute Type	Attribute Description
bias†	NanosecondsCount	The clock's time bias.
biasUncertianty†	NanosecondsCount	The clock's bias uncertainty (1-Sigma).
drift†	NanosecondsDrift	The clock's drift.
driftUncertainty†	NanosecondsDrift	The clock's drift uncertainty (1-Sigma).
elapsedRealtime†	NanosecondsCount	The clock's elapsed real-time of this clock since system boot.
elapsedRealtimeUncertainty†	NanosecondsCount	The estimate of the relative precision of the alignment of the elapsedRealtime timestamp (68% confidence).
fullBias†	NanosecondsCount	The difference between hardware clock inside GPS receiver and the true GPS time since 0000Z, January 6, 1980.
hardwareClockDiscontinuityCount†	NanosecondsCount	The count of hardware clock discontinuities.
leapSecond†	Count	The leap second associated with the clock's time.
referenceCarrierFrequency†	RadioFrequency_Hertz	The reference carrier frequency in Hz.
referenceCodeTypeFor†	StringShortDescription	The reference code type.

Attribute Name	Attribute Type	Attribute Description
referenceConstellationType†	GPSConstellationEnumType	The reference constellation type.
time†	Count	The receiver internal hardware clock value.
timeUncertainty†	Size_Numeral	The clock's time uncertainty (1-Sigma).

5.2.10 GPSSatelliteType

Namespace: UMAA::SEM::GPSStatus::GPSSatelliteType

Description: Specifies an element that specifies the characteristics of a gps satellite.

Table 30: GPSSatelliteType Structure Definition

Attribute Name	Attribute Type	Attribute Description
antennaCarrierNoiseDensity†	CarrierToNoiseDensityRatio	The carrier-to-noise density at the antenna of the satellite.
azimuth†	Angle	The azimuth of the satellite.
basebandCarrierNoiseDensity†	CarrierToNoiseDensityRatio	The baseband carrier-to-noise density of the satellite.
carrierFrequency†	RadioFrequency_Hertz	The carrier frequency of the signal tracked.
constellationType†	GPSConstellationEnumType	The constellation type of the satellite.
containsAlmanacData†	BooleanEnumType	Whether the satellite has almanac data.
elevation†	Angle	The elevation of the satellite.
ephemerisData†	BooleanEnumType	Whether the satellite has ephemeris data
satelliteId†	NumericGUID	The identification number for the satellite.
usedInFix†	BooleanEnumType	Whether the satellite was used in the calculation of the most recent position fix.

5.2.11 GeodeticLatitude

Namespace: UMAA::Common::Measurement::GeodeticLatitude

Description: GeodeticLatitude specifies the angle between the normal and the equatorial plane of the ellipsoid. The Latitude specifies the north-south position of a point.

Table 31: GeodeticLatitude Structure Definition

Attribute Name	Attribute Type	Attribute Description
latitude	GeodeticLatitude	GeodeticLatitude specifies the angle between the normal and the equatorial plane of the ellipsoid. The Latitude specifies the north-south position of a point.

5.2.12 GeodeticLongitude

Namespace: UMAA::Common::Measurement::GeodeticLongitude

Description: GeodeticLongitude specifies the angular measurement of a location east or west of the prime meridian of the reference ellipsoid.

Table 32: GeodeticLongitude Structure Definition

Attribute Name	Attribute Type	Attribute Description
longitude	GeodeticLongitude	GeodeticLongitude specifies the angular measurement of a location east or west of the prime meridian of the reference ellipsoid.

5.2.13 Position2D

Namespace: UMAA::Common::Measurement::Position2D

Description: Position2D specifies a location on the surface of the Earth.

Table 33: Position2D Structure Definition

Attribute Name	Attribute Type	Attribute Description
geodeticLatitude	GeodeticLatitude	geodeticLatitude specifies the north-south coordinate of the position.
geodeticLongitude	GeodeticLongitude	geodeticLongitude specifies the east-west coordinate of the position.

5.2.14 Position2DTime

Namespace: UMAA::Common::Measurement::Position2DTime

Description: Position2DTime specifies a location on the surface of the Earth at a given point in time.

Table 34: Position2DTime Structure Definition

Attribute Name	Attribute Type	Attribute Description
geodeticPosition	Position2D	geodeticPosition specifies a location on the surface of the Earth.
timeAtPosition†	DateTime	timeAtPosition specifies the date and time when this position is considered valid or an associated measurement was made.

5.2.15 Position3D_WGS84

Namespace: UMAA::Common::Measurement::Position3D_WGS84

Description: Position3D_WGS84 specifies a location relative to the WGS-84 ellipsoid.

Table 35: Position3D_WGS84 Structure Definition

Attribute Name	Attribute Type	Attribute Description
geodeticPosition	Position2D	geodeticPosition specifies a location on the surface of the Earth.
heightAboveEllipsoid	Altitude_HAE	heightAboveEllipsoid specifies the height above the WGS-84 ellipsoid.

5.2.16 Quaternion

Namespace: BasicTypes::Quaternion

Description: Defines a four-element vector that can be used to encode any rotation in a 3D coordinate system.

Table 36: Quaternion Structure Definition

Attribute Name	Attribute Type	Attribute Description
a	double	Real number a.
b	double	Real number b.
c	double	Real number c.
d	double	Real number d.

5.2.17 Velocity3D_PlatformNED

Namespace: UMAA::Common::Measurement::Velocity3D_PlatformNED

Description: Velocity3D_PlatformNED specifies the velocity given by northing, easting and down vectors in a North-East-Down coordinate system centered on the platform.

Table 37: Velocity3D_PlatformNED Structure Definition

Attribute Name	Attribute Type	Attribute Description
downSpeed	DownSpeed	downSpeed specifies the down velocity vector in a North-East-Down coordinate system centered on the platform.
eastSpeed	EastSpeed	eastSpeed specifies the easting velocity vector in a North-East-Down coordinate system centered on the platform.
northSpeed	NorthSpeed	northSpeed specifies the northing velocity vector in a North-East-Down coordinate system centered on the platform.

5.3 Enumerations

Enumerations are used extensively throughout UMAA. This section lists the values associated with each enumeration defined in UCS-UMAA.

5.3.1 CommandStatusReasonEnumType

Namespace: UMAA::Common::MaritimeEnumeration::CommandStatusReasonEnumType

Description: Defines a mutually exclusive set of reasons why a command status state transition has occurred.

Table 38: CommandStatusReasonEnumType Enumeration

Enumeration Value	Description
CANCELED	Indicates a transition to the CANCELED state when the command is canceled successfully.
VALIDATION_FAILED	Indicates a transition to the FAILED state when the command contains missing, out-of-bounds, or otherwise invalid parameters.
OBJECTIVE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to external factors.
SERVICE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to processing failure.
RESOURCE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to resource or platform failure.
RESOURCE_REJECTED	Indicates a transition to the FAILED state when the commanded resource rejects the command for some reason.
INTERRUPTED	Indicates a transition to the FAILED state when the command has been interrupted by a higher priority process.
TIMEOUT	Indicates a transition to the FAILED state when the command is not acknowledged within some defined time bound.
SUCCEEDED	Indicates the conditions to proceed to this state have been met and a normal state transition has occurred.

5.3.2 GPSConstellationEnumType

Namespace: UMAA::Common::MaritimeEnumeration::GPSConstellationEnumType

Description: A mutually exclusive set of values that defines the constellation type of the satellite.

Table 39: GPSConstellationEnumType Enumeration

Enumeration Value	Description
UNKNOWN	An unknown constellation.
BEIDOU	The Beidou constellation.
GALILEO	The Galileo constellation.
GLONASS	The Glonass constellation.
GPS	The GPS constellation.
IRNSS	The IRNSS constellation.
QZSS	The QZSS constellation.
SBAS	The SBAS constellation.

5.3.3 GPSFixEnumType

Namespace: UMAA::Common::MaritimeEnumeration::GPSFixEnumType

Description: A mutually exclusive set of values that defines the command status of the GPSFix.

Table 40: GPSFixEnumType Enumeration

Enumeration Value	Description
INITIATING	Initiating GPS Fix
PERFORMING	Performing GPS Fix
STABLE	Stable GPS Fix

5.3.4 GPSNavigationSolutionEnumType

Namespace: UMAA::Common::MaritimeEnumeration::GPSNavigationSolutionEnumType

Description: A mutually exclusive set of values that defines navigation solution of the unmanned platform.

Table 41: GPSNavigationSolutionEnumType Enumeration

Enumeration Value	Description
GPS_1	GPS 1
GPS_2	GPS 2
GPS_2D	GPS 2D
GPS_3	GPS 3
GPS_3D	GPS 3D
GPS_4	GPS 4
GPS_DEAD_RECK	GPS dead reckoning
NO_NAV	No navigation information

5.3.5 InertialSensorOpStatusEnumType

Namespace: UMAA::Common::MaritimeEnumeration::InertialSensorOpStatusEnumType

Description: An enumeration that is used to report the operational status of the inertial sensor.

Table 42: InertialSensorOpStatusEnumType Enumeration

Enumeration Value	Description
COURSE_GPS_ALIGNMENT	Course GPS alignment
COURSE_STATIONARY_ALIGNMENT	Course stationary alignment
FINE_GPS_ALIGNMENT_COMPLETE	Fine GPS alignment complete
FINE_GPS_ALIGNMENT_STARTED	Fine GPS alignment started

Enumeration Value	Description
FINE_STATIONARY_ALIGNMENT_COMPLETE	Fine stationary alignment complete
FINE_STATIONARY_ALIGNMENT_STARTED	Fine stationary alignment started
INERTIAL_SENSOR_FAILURE	Inertial sensor failure
INIT	Initializing
OPERATIONAL	Operational

5.3.6 InertialSensorStateEnumType

Namespace: UMAA::Common::MaritimeEnumeration::InertialSensorStateEnumType

Description: A mutually exclusive set of values that defines the operational state of inertial sensor unit.

Table 43: InertialSensorStateEnumType Enumeration

Enumeration Value	Description
OFF	Off
ON	On
RESTARTING	Restarting

5.3.7 CommandStatusEnumType

Namespace: UMAA::Common::MaritimeEnumeration::CommandStatusEnumType

Description: Defines a mutually exclusive set of values that defines the states of a command as it progresses towards completion.

Table 44: CommandStatusEnumType Enumeration

Enumeration Value	Description
FAILED	The command has been attempted, but was not successful.
COMPLETED	The command has been completed successfully.
ISSUED	The command has been issued to the resource (typically a sensor or streaming device), but processing has not yet commenced.
COMMANDED	The command has been placed in the resource's command queue but has not yet been accepted.
EXECUTING	The command is being performed by the resource and has not yet been completed.
CANCELED	The command was canceled by the requestor before the command completed successfully.

5.4 Type Definitions

This section describes the type definitions for UMAA. The table below lists how UMAA defined types are mapped to the DDS primitive types.

Table 45: Type Definitions

Type Name	Primitive Type	Range of Values	Description
Angle	double	fractionDigits=3 maxInclusive=3.141592653589793238462643383279 5 minInclusive=-3.141592653589793126462643383279 5 units=Radian referenceFrame=Counting	Angle specifies the amount of turning necessary to bring one ray, line or plane into coincidence with or parallel to another. The measurement is stated in radians between -pi and pi.
BooleanEnumType	boolean	units=N/A minInclusive=N/A maxInclusive=N/A fractionDigits=N/A length=N/A	BooleanEnumTypeLDM is a Realization of BooleanEnumType which is a mutually exclusive set of values that defines the truth values of logical algebra.
CarrierToNoiseDensityRatio	double	maxInclusive=1000000 minInclusive=0 units=dB-Hz	Describes a ratio of the carrier power to the noise power density.
Count	long	units=N/A minInclusive=-2147483648 maxInclusive=2147483647 fractionDigits=0	Represents a whole (non-fractional) number that can be positive, negative or zero.
Course_TrueNorth	double	fractionDigits=3 maxInclusive=6.283185307179586364925286766559 minInclusive=0 units=Radian referenceFrame=TrueNorth	Course_TrueNorth specifies the direction of the platform's motion relative to true north. The measurement is stated in radians between 0 and 2 pi.
DateTimeNanoseconds	long	units=Nanoseconds minInclusive=0 maxInclusive=999999999 fractionDigits=0	number of nanoseconds elapsed within the current second.
DateTimeSeconds	longlong	units=Seconds minInclusive=0 maxInclusive=18446744073709500000 fractionDigits=0	seconds offset from the standard POSIX (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.
DownSpeed	double	axisDirection=down axisUnit=MeterPerSecond maximumValue=-10000 minimumValue=10000 rangeMeaning=exact resolution=0.001	The DownSpeed axis is used for measuring speed and increases in magnitude as speed toward the center of the Earth increases. DownSpeed measurements are expressed in meters per second.
EastSpeed	double	axisDirection=east axisUnit=MeterPerSecond maximumValue=-10000 minimumValue=10000 rangeMeaning=exact resolution=0.001	The EastSpeed axis is used for measuring speed and increases in magnitude as speed in the easterly direction increases. EastSpeed measurements are expressed in meters per second.

Type Name	Primitive Type	Range of Values	Description
EllipsoidalHeight	double	axisAbbrev=h axisDirection=up axisUnit=Meter maximumValue=700000 minimumValue=-10000 rangeMeaning=exact resolution=0.001	The EllipsoidalHeight axis is used for measuring position and increases in magnitude as values extend away from the center of the Earth. EllipsoidalHeight measurements are expressed in meters.
GeodeticLatitude	double	axisAbbrev=Latitude axisDirection=north/south axisUnit=Degrees maximumValue=90.0 minimumValue=-90.0 rangeMeaning=exact resolution=0.0000000001	The Latitude axis is used for measuring position and increases in magnitude as position extends from the south pole to the north pole. Latitude measurements are expressed in degrees.
GeodeticLongitude	double	axisAbbrev=Longitude axisDirection=east axisUnit=Degrees maximumValue=180.0 minimumValue=-180.0 rangeMeaning=wraparound resolution=0.0000000001	The Longitude axis is used for measuring position and increases in magnitude as position extends eastward. Longitude measurements are expressed in degrees. Longitude measurements are periodic and whose limits (min and max), while mathematically discontinuous, represent a continuous range.
GroundSpeed	double	units=MeterPerSecond minInclusive=0 maxInclusive=200 fractionDigits=6	This type stores speed in meters/s.
MagneticVariation	double	fractionDigits=3 maxInclusive=3.141592653589793238462643383279 5 minInclusive=-3.141592653589793126462643383279 5 units=Radian referenceFrame=TrueNorth	MagneticVariation specifies the angle on the horizontal plane between true north and magnetic north (the direction the north end of the compass needle points). The measurement is stated in radians between -pi and pi.
MSLHeight	double	axisDirection=up axisUnit=Meter maximumValue=700000 minimumValue=-10000 rangeMeaning=exact resolution=0.001	The MSLHeight axis is used for measuring position and increases in magnitude as values extend away from the center of the Earth. MSLHeight measurements are expressed in meters.
NanosecondsCount	long	maxInclusive=9223372036854775807 minInclusive=-9223372036854775808 units=Nanosecond	Describes a count of nanoseconds.
NanosecondsDrift	long	maxInclusive=2147483647 minInclusive=-2147483648 units=Nanoseconds/Second	Describes a clock deviation rate in regards to nanoseconds drift per second.
NorthSpeed	double	axisDirection=north axisUnit=MeterPerSecond maximumValue=-10000 minimumValue=10000 rangeMeaning=exact resolution=0.001	The NorthSpeed axis is used for measuring speed and increases in magnitude as speed in the northerly direction increases. NorthSpeed measurements are expressed in meters per second.

Type Name	Primitive Type	Range of Values	Description
NumericGUID	octet[16]	units=N/A minInclusive=0 maxInclusive=(2^128)-1 fractionDigits=0	Represents a 128-bit number according to RFC 4122 variant 2
RadioFrequency_-Hertz	double	units=Hertz minInclusive=0.0 maxInclusive=10000000000 fractionDigits=6	This type stores Frequency in Hz.
Size_Numeral	double	units=N/A minInclusive=-1000000000000 maxInclusive=1000000000000 fractionDigits=3	Represents nonnegative integers.
StringLongDescription	string	fractionDigits=N/A length=4095 maxExclusive=N/A maxInclusive=N/A minExclusive=N/A minInclusive=N/A units=N/A	Represents a long format description.
StringShortDescription	string	fractionDigits=N/A length=1023 maxExclusive=N/A maxInclusive=N/A minExclusive=N/A minInclusive=N/A units=N/A	Represents a short format description.

A Appendices

A.1 Acronyms

Note: This acronym list is included in every ICD and covers the complete UMAA specification. Not every acronym appears in every ICD.

ADD	Architecture Design Description
AGL	Above Sea Level
ASF	Above Sea Floor
BSL	Below Sea Level
BWL	Beam at Waterline
C2	Command and Control
CMD	Command
CO	Comms Operations
CPA	Closest Point of Approach
CTD	Conductivity, Temperature and Depth
DDS	Data Distribution Service
EO	Engineering Operations
FB	Feedback
GUID	Globally Unique Identifier
HM&E	Hull, Mechanical, & Electrical
ICD	Interface Control Document
ID	Identifier
IDL	Interface Definition Language Specification
IMO	International Maritime Organization
INU	Inertial Navigation Unit
LDM	Logical Data Model
LOA	Length Over All
LRC	Long Range Cruise
LWL	Length at Waterline
MDE	Maritime Domain Extensions
MEC	Maximum Endurance Cruise
MM	Mission Management
MMSI	Maritime Mobile Service Identity
MO	Maneuver Operations
MRC	Maximum Range Cruise
MSL	Mean Sea Level
OMG	Object Management Group
PIM	Platform Independent Model
PMC	Primary Mission Control
PNT	Precision Navigation and Timing
PO	Processing Operations
PSM	Platform Specific Model
RMS	Root-Mean-Square
RPM	Revolutions per minute
RTPS	Real Time Publish Subscribe
RTSP	Real Time Streaming Protocol

SA	Situational Awareness
SEM	Sensor and Effector Management
SO	Support Operations
SoaML	Service-oriented architecture Modeling Language
STP	Standard Temperature and Pressure
UCS	Unmanned Systems Control Segment
UMAA	Unmanned Maritime Autonomy Architecture
UML	Unified Modeling Language
UMS	Unmanned Maritime System
UMV	Unmanned Maritime Vehicle
UxS	Unmanned System
WGS84	Global Coordinate System
WMO	World Meteorological Organization