AEROSPACE STANDARD

AS6522™ REV. B

Issued 2016-12
Revised 2020-08
Stabilized 2024-02

Superseding AS6522A

Unmanned Systems (UxS) Control Segment (UCS) Architecture: Architecture Technical Governance

RATIONALE

This document is no longer part of UCS Architecture as of Rev A.

STABILIZED NOTICE

This document has been declared "STABILIZED" by SAE AS-4UCS Unmanned Systems Control Segment Architecture Committee and will no longer be subjected to periodic reviews for currency. Users are responsible for verifying references and continued suitability of technical requirements. Newer technology may exist.

INTRODUCTION

This Architecture Technical Governance for the SAE UCS Architecture Model (AS6518) starting at Revision A describes how the Model is structured and how it is to be interpreted, manipulated and extended by users. The UCS Architecture Model is expressed by the Unified Modeling Language® (UML®) and various UML Profiles. Technical Governance describes the purpose of these Profiles and how they are to be applied to the Packages and Diagrams in the Model.

In general terms, the UCS Architecture Model and its user extensions are required to conform to this Technical Governance. UCS Products are in turn required to conform to the UCS Architecture Model and its user extensions. A conformant extension of the UCS Architecture Model can be manipulated by UCS tools. UCS Product conformance is defined in the UCS Architecture Conformance Specification (AS6513).

Technical Governance is expressed through a collection of policies, where each policy describes its motivation, its required usage, constraints and conformance, the policy itself, and application examples of the policy.

APPLICABILITY

This document applies to the UCS Architecture Model (AS6518) and to user extensions of AS6518 for development of conformant UCS Products. This Revision A applies to Revision A (and possibly later revisions) of AS6518.

TABLE OF CONTENTS

1. SCOPE

This Technical Governance is part of the SAE UCS Architecture Library and is primarily concerned with the UCS Architecture Model (AS6518) starting at Revision A and its user extensions. Users of the Model may extend it in accordance with AS6513 to meet the needs of their UCS Products. UCS Products include software components, software configurations and systems that provide or consume UCS services. For further information, refer to AS6513 Revision A or later.

Technical Governance is part of the UCS Architecture Framework. This framework governs the UCS views expressed as Packages and Diagrams in the UCS Architecture Model.

2. TERMINOLOGY

In this document, all strict textual references to Unified Modeling Language (UML) Metaclasses and Stereotypes will begin with a capital letter and follow the naming conventions in the applicable standard. All strict textual references to instances of Metaclasses and Stereotypes (including notional examples) will be indicated in bold text.

For example, the UCS Architecture Model comprises the **Notices** Package, **Data Model Framework** Package, **UCS Architectural Model Package** and **UCS Architecture Technical Goverance** Package. The Stereotype **ScalarQuantity** Extends DataType. An Instance of this Stereotype is **Mass[kg]** whose Attribute **number** denotes the magnitude of mass in kilograms.

3. REFERENCES

3.1 Applicable Documents

The following publications form a part of this document to the extent specified herein. The latest issue of SAE publications shall apply. The applicable issue of other publications shall be the issue in effect on the date of the purchase order. In the event of conflict between the text of this document and references cited herein, the text of this document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific examption has been obtained.

3.1.1 SAE Publications

Available from SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or +1 724-776-4970 (outside USA), www.sae.org.

AS6512A	SAE Unmanned Systems (UxS) Control Segment (UCS) Architecture: Architecture Description
AS6513A	SAE Unmanned Systems (UxS) Control Segment (UCS) Architecture: Conformance Specification
AS6518A	SAE Unmanned Systems (UxS) Control Segment (UCS) Architecture: Architecture Model
AS6969A	SAE Data Dictionary for Quantities Used in Cyber Physical Systems

3.1.2 Object Management Group (OMG) Publications

OMG OMG Unified Modeling Language® (UML®). Version 2.5.1. December 2011. Available online at: https://www.omg.org/spec/UML/2.5.1.

OMG Service Oriented Architecture Modeling Language (SoaML). Version 1.0.1. May 2012. Available online at: https://www.omg.org/spec/SoaML/1.0.1.

OMG System Modeling Language (SysML®). Version 1.6. December 2019. Available online at: https://www.omg.org/spec/SysML/1.6.

3.1.3 ISO Publications

Copies of these documents are available online at http://webstore.ansi.org/.

ISO/IEC 80000	Quantities and Units

3.1.4 ECMA International Publications

Standard EMCA-262	ECMAScript® 2019 Language Specification. 10th edition (June 2019).

3.1.5 U.S. Government Publications

Available from U.S. Army Combat Capabilities Development Command (CCDC), Ground Vehicle Systems Center, 6305 E 11 Mile Rd, Warren, MI 48092.

U.S Government. Data Model Framework (DMF). Version 1.1. April 2020.

3.2 Definitions

This technical governance conforms to the definitions provided in the Data Model Framework and OMG standards cited in 3.1.

3.3 Acronyms

CDM	Conceptual Data Model
DMF	Data Model Framework
GUID	Global Unique Identifier
ISO	International Organization for Standardization
LDM	Logical Data Model
NFP	Non-Functional Property
OMG	Object Management Group
QoS	Quality of Service
SOA	Service Oriented Architecture
SoaML	Service Oriented Architecture Modeling Language
UAF	Unified Architecture Framework
UCS	UxS Control Segment
UML	Unified Modeling Language
UxS	Unmanned System
XML	Extensible Markup Language

4. GENERAL POLICIES

4.1 Policy on Namespace

4.1.1 Policy Usage and Conformance

This policy applies to the UCS Architecture Model and any derived SAE-managed artifacts delivered in XML format.

4.1.2 Policy Description and Motivation

This policy defines the UCS convention for declaring the Namespace of XML Artifacts. The policy is intended to avoid clashes between names from different markup vocabularies.

4.1.3 Policy Constraints

The definition Objects defined by SAE AS6969 (beginning in Revision A) use the Namespace name **https://www.sae.org/AS6969/**.

4.1.4 Policy

All UCS XML architecture products will declare the UCS XML Namespace using the UCS Namespace identifier **https://www.sae.org**/AS6512/ where AS6512 is the designation for the UCS Architecture Library, including the UCS Architecture Model. This is represented as an xmlns XML attribute:

xmlns: ucs="https://www.sae.org/AS6512/"

In addition to the Namespace declaration, each release of the UCS Architecture XML products must contain the release identifier as a local name using a version attribute. For example, for Revision A:

xmlns: ucs="https://www.sae.org/AS6512/" version="Revision A"

4.1.5 Policy Example(s)

None.

4.1.6 Policy history

Updated in Revision A from AS6522 UCS-TECHGOV-POLICY Namespace.

4.2 Policy on Model Languages

4.2.1 Policy Usage and Conformance

This policy applies to the UCS Architecture Model, AS6518, starting at Revision A. The model will conform to this policy.

4.2.2 Policy Description and Motivation

This policy identifies the Unified Modeling Language (UML) specifications and industry Profiles of UML to be Applied in the UCS Architcture Model.

4.2.3 Policy Constraints

Contraints on this policy are established by other policies in this technical governance.

4.2.4 Policy

The UCS Architecture Model shall be expressed in UML 2.5.1 sing industry profiles as determined by the subject matter. These industry profiles are:

1. OMG Service Oriented Architecture Modeling Language (SoaML) version 1.0.1

2. OMG System Modeling Langauge (SysML) version 1.6

3. OMG Unified Archirtecture Framework (UAF) Profile (UAFP) version 1.0

4. U.S. Government profiles defined by the Data Model Framework (DMF) version 1.1

In addition, the UCS Architecture Model defines profiles of UML as established in policies in this technical governance. The UCS Profile Diagrams are provided in the **UCS Architecture UML Profiles** Package of the **UCS Architecture Technical Governance** Package.

4.2.5 Policy Examples

None.

4.2.6 Policy History

Original at Revision A.

4.3 Policy on UCS Architecture Model Package Structure

4.3.1 Policy Usage and Conformance

The policy applies to the UCS Architecture Model, AS6518, beginning at Revision A. The model will conform to this policy.

4.3.2 Policy Description and Motivation

This policy describes the high level Package structure of the UCS Architecture Model. Further policies will expand upon these Packages.

4.3.3 Policy Constraints

None.

4.3.4 Policy

The UCS Architecture Model comprises four model views as shown in Figure 1. These model views are the **UCS Architectural Model** Package, the **UCS Architecture Technical Goverance** Package, the **Data Model Framework** Package, and **NOTICES** Package. It will be noted in this terminology that the **UCS Architectural Model** is a Package within the UCS Architecture Model.

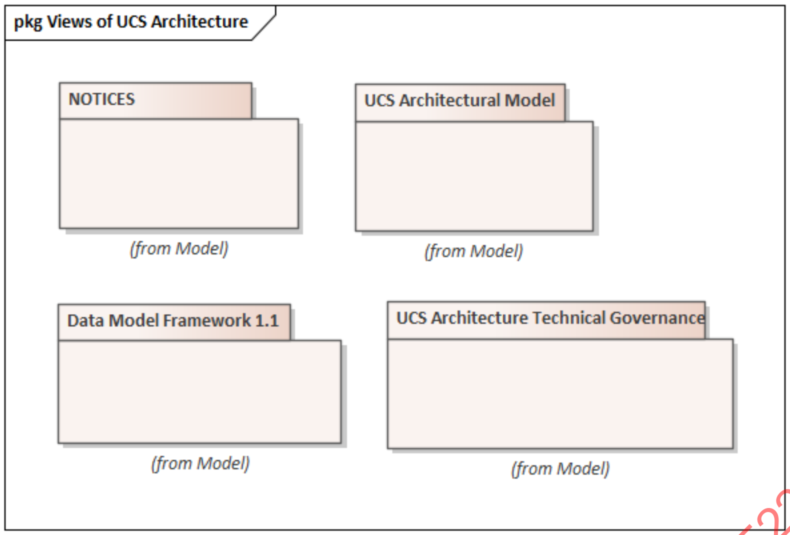


Figure 1 - UCS Architecture Model

The **Notices** Package contains publication metadata about the UCS Architecture Model.

The **UCS Architectural Model** is a Package that contains the UCS Architecture information model at two levels of abstraction called the **Conceptual Model** and the **Logical Model**. These abstractions are based on the abstraction of their DataTypes as defined in the **Data Model Framework 1.1** Package (DMF). The DMF includes the definitions provided in AS6969. The **Conceptual Model** and **Logical Model** have a related but not identical Package structure. Each contains a service model, a message model, and a datamodel. Policies concerning these models are provided in Section 5.

Supporting the information model is the **Non-Functional Property Model** Package, the **System Use Case Model** Package, and the **Model Overview** Package. These Packages are provided for reference only and are not normative. Policies concerning the **Non-Functional Property Model** are provided in Section 6. Policies concerning the **System Use Case Model** are provided in Section 7. See Figure 2.

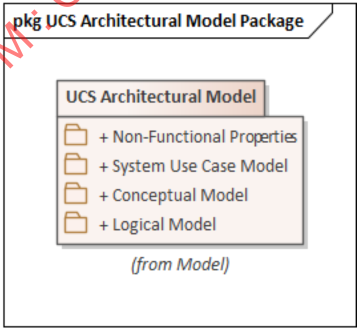


Figure 2 - UCS Architectural Model Packages

The architecture framework for the **UCS Architectural Model** is provided by the **UCS Architecture Technical Governance Package** and **Data Model Framework 1.1** Package. The **UCS Architecture Technical Governance** Package contains the UCS Architecture Profiles of UML together with this document. The **DMF** Package is a Government-owned plug-in to the UCS Architecture Model. It contains Profiles of UML, Technical Governance, and the AS6969 Library of definition Objects upon which the DMF is based. The **DataTypes: Common** Package in the DMF contains an example library of conceptual and logical DataTypes. These DataTypes at the Global Unique Identifier (GUID) level are not used in the **UCS Architectural Model** Package. The **UCS Architectural Model** Package contains its own DataTypes (with unique GUIDs).

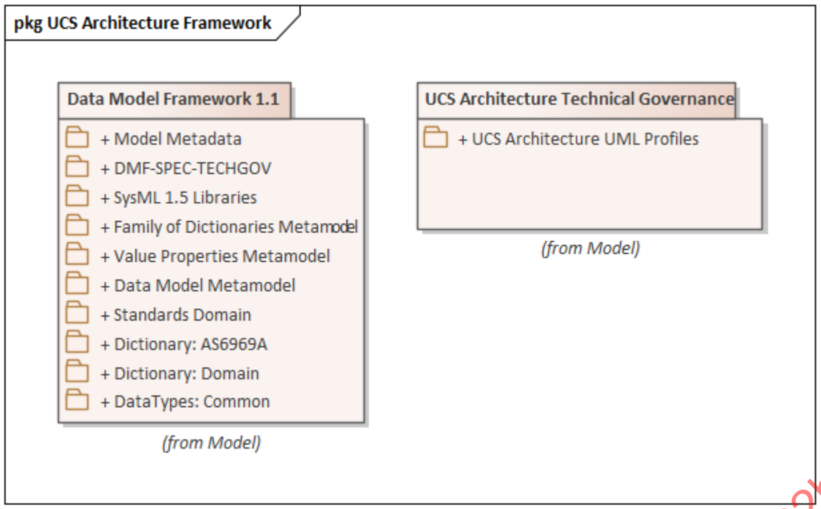


Figure 3 - UCS Architecture Framework

4.3.5 Policy Example(s)

None.

4.3.6 Policy History

Updated in Revision A from AS6522 UCS-TECHGOV-POLICY Architecture Package Structure.

4.4 General Policy on Naming Conventions

4.4.1 Policy Usage and Conformance

This policy applies to NamedElements in the UCS Architecture Model. It should be noted that UML-based modeling languages, including SysML and UAF, do not formally impose naming conventions.

4.4.2 Policy Description and Motivation

This policy establishes general model naming conventions. These conventions will promote a uniform modeling style and make it easier to navigate and understand the UCS Architecture Model and associated work products that reference those models. Additional naming policies are provided for specific Packages in the UCS Architecture Model.

4.4.3 Policy Constraints

This policy does not apply to separately governed third-party content that has been imported into UCS Architecture Model.

4.4.4 Policy

NamedElements in the model shall have names that satisfy the regular expression specified below for each category of name. The regular expressions specified below satisfy JavaScript conventions (ECMA-262) and are used verbatim in name checks.

Package and Type name conventions permit an underscore "_" prefix during model development that indicates that an element is work-in-progress and not part of the formal model.

4.4.4.1 Package and Use Case Names

Allow spaces, periods, hyphens, parentheses.

```
    var legalPackageOrUseCaseName = new RegExp("^_?[A-Z][a-zA-Z0-9_.() -]*$");
```

4.4.4.2 General Type Names (Classes, Enumerations, Components, Actors)

Type names follow the general "camel case" convention of beginning with an uppercase letter.

```
    var legalTypeName = new RegExp("^_?[A-Z][a-zA-Z0-9_-]*$");
```

Service Interfaces allow a single preceding "~" character.

```
    var legalTypeNameConjugate = new RegExp("^~[A-Z][a-zA-Z0-9_]*$");
```

4.4.4.3 General Instance Names (Attributes, Operations)

Instance names follow the general "camel case" convention of beginning with a lowercase letter.

```
    var legalInstanceName = new RegExp("^[a-z][A-Za-z0-9_]*$");
```

Instance names also permit an initial acronym where an acronym consists of at least two uppercase letters, or an uppercase letter and a digit.

```
    var legalInstanceNameAcroPrefix = new RegExp("^[A-Z][A-Z0-9][A-Za-z0-9_]*$");
```

Enumeration literals follow the convention of all uppercase letters with digits and underscores.

```
    var legalEnumerationAttributeName = new RegExp("^[A-Z][A-Z0-9_]*$");
```

4.4.4.4 Specific Instance Names (Attributes, Operations)

Time stamps:

Except as allowed by this set of rules, all Attributes relating to time or time stamps shall use the term "**timeStamp**." The term **timeStamp** shall be considered as the concatenation of the two words "time" and "stamp." This distinction is specified for camel case guidelines.

The Attribute name may include an additional identifier, but the name shall still use the term **timeStamp** with the appropriate camel case (e.g., **previousTimeStamp**). If the Attribute belongs to an Element realizing a NamedElement in the **Conceptual Model** that already has a time-related Attribute (e.g**., TimeCapabilityType**), the Attribute name shall use the name present in the **Conceptual Model** Element (e.g., **timeSetPoint** would be named **timeSetPoint** and not **timeStampSetPoint**).

4.4.5 Policy Example(s)

See the policy examples throughout this document.

4.4.6 Policy History

Updated in Revision A from AS6522 UCS-TECHGOV-POLICY Naming Conventions. Naming conventions for content in specific Packages is now provided in the appropriate policy.

5. INFORMATION MODEL POLICIES

These policies apply to the **Conceptual Model** Package and **Logical Model** Package, which provide the core views of the UCS Architecture Model at different levels of abstraction**.** The defining difference between these models is the level of abstraction of their DataTypes. Both models contain a service model, a message model, and a datamodel.

The principal Package in the **Conceptual Model** is the **Conceptual Data Model** (CDM). This is a definitional model of the entities in the UCS Domain. In this definitional model, the DataTypes serve only to scope the semantics of an entity's innate Properties. The main purpose of the **Conceptual Message Model** and **Conceptual Service Model** is to show the trace from the **Logical Message Model** to the CDM. The conformance of UCS Products to the UCS Architecture is based on the **Logical Service Model** whose Interface Signals are parameterized by logical Messages defined in the **Logical Message Model**. In the **Logical Message Model**, DataTypes specify the Value structure so that Values can be interpreted.

For example, the conceptual DataType **Mass** would be used only to Type a particular Property as a mass property (e.g., **Aircraft:: fuelAmount: Mass**). The Features of this Datatype are insufficient to interpret a Value of **Mass**. The logical DataType **Mass[kg]** is a refinement of **Mass**. Its Value is **number: Real** based on the arbitrarily chosen kilogram measurement unit.

5.1 Policy on Conformance to the Data Model Framework

5.1.1 Policy Usage and Conformance

This policy applies to the UCS Architecture Model, most specifically the **Conceptual Data Model** Package and **the Logical Data Model** Package. The UCS Architecture Model shall conform to this policy starting at Revision A.

5.1.2 Policy Description and Motivation

The **Data Model Framework 1.1** is a Package in the UCS Architecture Model.

The Data Model Framework (DMF) is a government owned and managed architecture framework for datamodels. It is founded on the mathematical principles that were established in the SAE AS6969 document, Data Dictionary for Quantities Used in Cyber Physical Systems. The Profiles of UML provided by the DMF are the **FOD** Profile, **DataType** Profile, **Value Property** Profile, and **Data Model Packages** Profile. The SAE AS6969 Model, starting at Revision A, is based on the DMF **FOD Profile**.

The Package structure for the imported elements of the Data Model Framework is shown in Figure 4. It includes the Package **Dictionary: AS6969A**, which is the definition library from AS6969 starting at Revision A. Adoption of the DMF provides the opportunity for a consistent approach to data dictionaries and datamodels across multiple government-sponsored architectures.

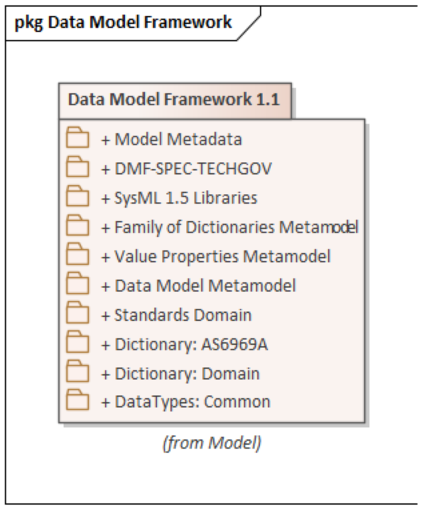


Figure 4 - Imported DMF Package structure

5.1.3 Constraints

The UCS Architectue Model is in the process of transitioning to towards full alignment with the DMF. For Revision A of the UCS Architecture Model, conformance does not include nominal properties, property modals or Enumerations. Not all DataTypes fully align with DMF in Revision A.

5.1.4 Policy

The UCS Architecture Model shall conform to the polices and UML profiles defined in the **Data Model Framework** Package. The document Artifact **DMF-SPEC-TECHGOV** contained with the **Data Model Framework** Package is a part of this technical governance by reference.

The **Data Model Framework** package, created through collaboration with the U.S. Army Autonomous Ground Vehicle Reference Architecture (AGVRA) effort, has been adopted and imported wholly into the UCS Architecture Model. Its models will be referenced and extended as needed by UCS, but the DMF package contents will not be edited by UCS.

The DMF Package **DataTypes: Common** provides example DataTypes. This Package is not directly used in the **UCS Architectural Model** Package. The **UCS Architectural Model** Package contains its own library of DataTypes. Many of these differ from their DMF equivalent only in the DataType's GUID.

5.1.5 Policy Example(s)

Begin Example

Figure 5 is from an example in Policy 7.03 of DMF-SPEC-TECHGOV. It shows the conceptual DataType **ThermodynamicTemperature** and the derived logical DataType **ThermometerTemperature[C]**. The semantics of these DataTypes is supported by definitions provided by AS6969. These are the definitions of **Thermodynamic Temperature Quantity**, **Celsius Temperature Quantity**, and **Degree Celsius**.

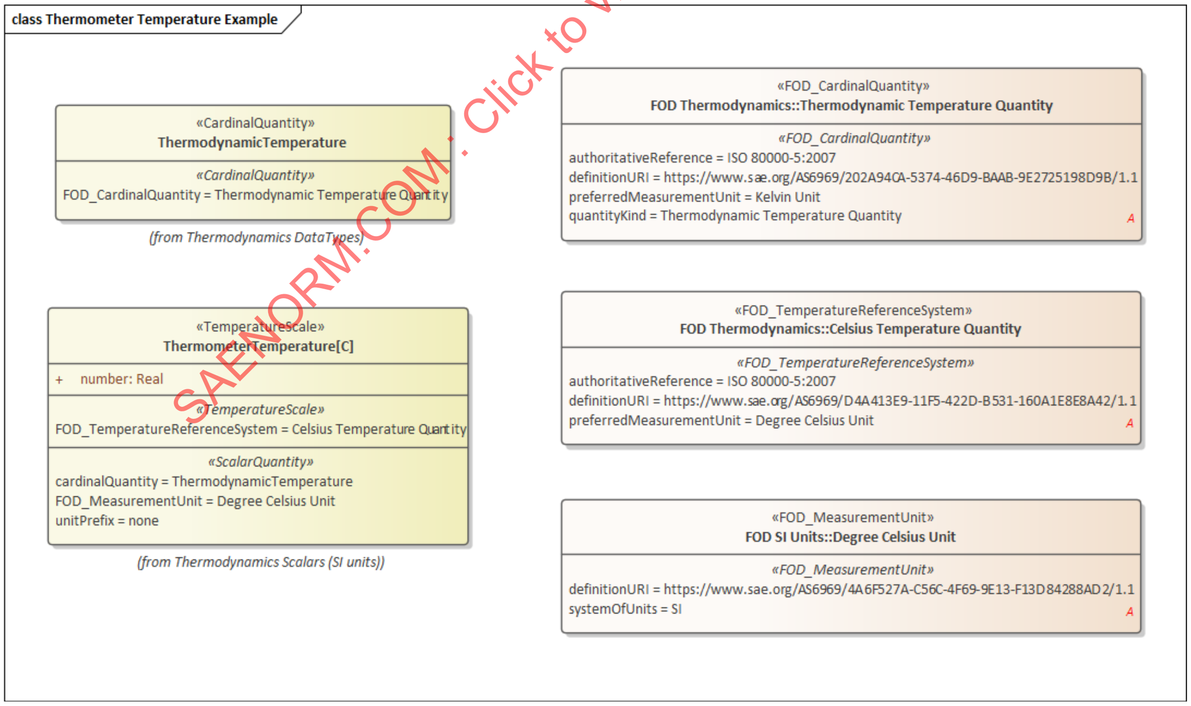

class Thermometer Temperature Example
«CardinalQuantity»
ThermodynamicTemperature
«CardinalQuantity»
FOD_CardinalQuantity = Thermodynamic Temperature Quantity
(from Thermodynamics DataTypes)
«TemperatureScale»
ThermometerTemperature[C]
+ number: Real
«TemperatureScale»
FOD_TemperatureReferenceSystem = Celsius Temperature Quantity
«ScalarQuantity»
cardinalQuantity = ThermodynamicTemperature
FOD_MeasurementUnit = Degree Celsius Unit
unitPrefix = none
(from Thermodynamics Scalars (SI units))
«FOD_CardinalQuantity»
FOD Thermodynamics::Thermodynamic Temperature Quantity
«FOD_CardinalQuantity»
authoritativeReference = ISO 80000-5:2007
definitionURI = https://www.sae.org/AS6969/202A94CA-5374-46D9-BAAB-9E2725198D9B/1.1
preferredMeasurementUnit = Kelvin Unit
quantityKind = Thermodynamic Temperature Quantity
«FOD_TemperatureReferenceSystem»
FOD Thermodynamics::Celsius Temperature Quantity
«FOD_TemperatureReferenceSystem»
authoritativeReference = ISO 80000-5:2007
definitionURI = https://www.sae.org/AS6969/D4A413E9-11F5-422D-B531-160A1E8E8A42/1.1
preferredMeasurementUnit = Degree Celsius Unit
«FOD_MeasurementUnit»
FOD SI Units::Degree Celsius Unit
«FOD_MeasurementUnit»
definitionURI = https://www.sae.org/AS6969/4A6F527A-C56C-4F69-9E13-F13D84288AD2/1.1
systemOfUnits = SI

Figure 5 - Example DataTypes that conform to the DMF and AS6969

End Example

5.1.6 Policy History

New at Revision A.

5.2 Policy on Levels of Architecture Abstraction

5.2.1 Policy Usage and Conformance

This policy applies to the **Conceptual Data Model** Package and **Logical Data Model** Package within the **UCS Architectural Model** Package. These Packages shall conform to this policy.

5.2.2 Policy Description and Motivation

The defining difference between the **Conceptual Data Model** and **Logical Data Model** is the level of abstraction of the Datatypes used in each model. Therefore a clear policy is required to distiguish between a conceptual DataType and a logical DataType. The distinction is based on DMF Technical Goverance Policy 8.01 (DataType Abstractions). The DMF is based on AS6969, which defines basic quantities, their modalities, and their morphisms. The morphism of a quantity is considered a logical specification.

It shall be noted that a conceptual specification is not intended to be vague. Force is precisely and completey defined in ISO 80000-4 as the rate of change of momentum. The logical value of a force vector may have many morphisms based on different vector spaces, measurement units, and unit prefixes. For example, the components of force may use kN in a North-East-Down vector space. The same value might also be expressed in pounds-force in an East-North-Up vector space.

5.2.3 Constraints

None.

5.2.4 Policy

The **Conceptual Model** shall use DataTypes as defined in DMF Technical Governance Policy 8.04 (Conceptual DataTypes) and DMF Technical Goverance Policy 8.05 (Conceptual Enumerations). DataTypes used in the **Conceptual Model** shall apply the Stereotypes defined in the DMF **DataType Profile** Package.

The **Logical Model** shall use DataTypes as defined in DMF Technical Governance Policy 8.07 (Logical DataTypes) and DMF Technical Goverance Policy 8.08 (Logical Enumerations). DataTypes used in the **Logical Model** shall apply the Stereotypes defined in the DMF **DataType Profile** Package.

It will be noted that logical DataTypes for basic quantities have a **number** Attribute whose Type is the SysML Primitive **Number** or its Specializations **Integer**, **Real** or **Complex**. The SysML **Number**, **Real** and **Complex** Primitives are to be interpreted as continuous variables whose range is only limited by the semantics of the quantity magnitude they express. It is only at the software implementation level that the Value Primitive is to be interpreted as a discrete variable that would occupy finite memory.

5.2.5 Policy Example(s)

Begin Example

Figure 6 is based on DMF-SPEC-TECHGOV Policy 8.01 (DataType Abstractions). It shows a conceptual DataType called **Mass** and two logical refinements of **Mass** called **Mass[kg]** and **Mass[mg].** The conceptual DataType is a Primitive with no Value Attribute. Its purpose is only to scope the semantics of a Property (measurand) in a conceptual data model. For example, the measurand **Aircraft:: fuelAmount: Mass** has a different meaning to **Aircraft:: fuelAmount: Volume**. A conceptual data model is a definitional model of entities and their innate properties. The definition of **Aircraft:: fuelAmount: Mass** does not depend on the chosen measurement unit.

In contrast to a conceptual data model, a logical data model is concerned with how the state of an entity is to be expressed. The two logical DataTypes in the figure provide alternate value structures for **Mass**. Consider the value 0.001 kg = 1000 mg. In the DataType **Mass [kg]**, the **number: Real** part of the value is 0.001. In the DataType **Mass [mg]**, the **number: Real** part of the value is 1000. These two DataTypes therefore provide different morphisms for the same value.

At the physical implementation level, the logical Attribute **number: Real** could be redefined, for example, by substituting the SysML Primitive **Real** with an unsigned 32-bit integer, and establishing a bit reference value. This is outside the scope of the UCS Architecture, however.

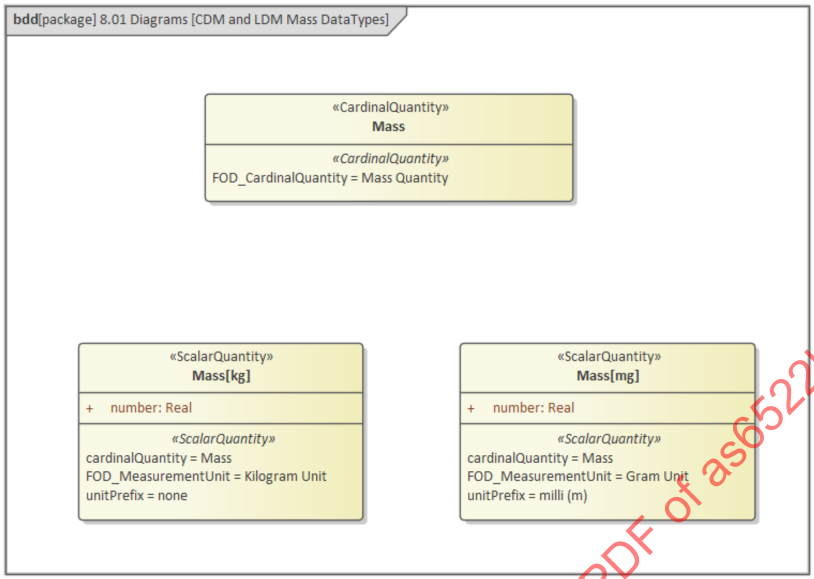

bdd[package] 8.01 Diagrams [CDM and LDM Mass DataTypes]
«CardinalQuantity» Mass
«CardinalQuantity» FOD_CardinalQuantity = Mass Quantity
«ScalarQuantity» Mass[kg]
+ number: Real
«ScalarQuantity» cardinalQuantity = Mass
FOD_MeasurementUnit = Kilogram Unit
unitPrefix = none
«ScalarQuantity» Mass[mg]
+ number: Real
«ScalarQuantity» cardinalQuantity = Mass
FOD_MeasurementUnit = Gram Unit
unitPrefix = milli (m)

Figure 6 - Conceptual and example logical DataType refinements for mass

End Example

Begin Example

Figure 7 is based on DMF-SPEC-TECHGOV Policy 8.07 (Logical DataTypes - N_Tensors). The conceptual DataType is **Position Vector**. The logical DataType is **WGS84_EllipsoidPosition[rad,m]**, which establishes a vector space for the value of position vector. The value has three coordinates, one of which is **WGS_EarthEllipsoidalHeight[m].**

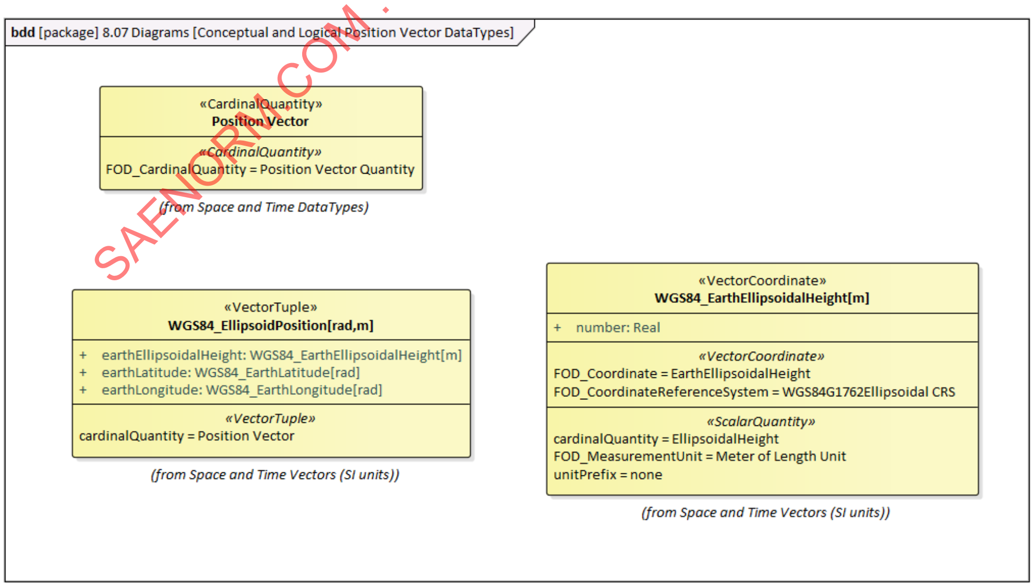

bdd [package] 8.07 Diagrams [Conceptual and Logical Position Vector DataTypes]
«CardinalQuantity» Position Vector
«CardinalQuantity» FOD_CardinalQuantity = Position Vector Quantity
(from Space and Time DataTypes)
«VectorTuple» WGS84_EllipsoidPosition[rad,m]
+ earthEllipsoidalHeight: WGS84_EarthEllipsoidalHeight[m]
+ earthLatitude: WGS84_EarthLatitude[rad]
+ earthLongitude: WGS84_EarthLongitude[rad]
«VectorTuple» cardinalQuantity = Position Vector
(from Space and Time Vectors (SI units))
«VectorCoordinate» WGS84_EarthEllipsoidalHeight[m]
+ number: Real
«VectorCoordinate» FOD_Coordinate = EarthEllipsoidalHeight
FOD_CoordinateReferenceSystem = WGS84G1762Ellipsoidal CRS
«ScalarQuantity» cardinalQuantity = EllipsoidalHeight
FOD_MeasurementUnit = Meter of Length Unit
unitPrefix = none
(from Space and Time Vectors (SI units))

Figure 7 - WGS84_EllipsoidPosition[rad,m]

End Example

Begin Example

Figure 8 is an agreed extension to DMF policies. It concerns the modeling of decibel representations of quantities. In the example, the conceptual DataType **RadiantFlux** is Refined into three alternate logical DataTypes. **RadiantFlux[mW]** is the standard linear value structure based on the milliwatt unit. **RadiantFlux[dBm]** differs in that the Attribute is **decibelNumber: Real**, where **decibelNumber** is to be interpreted as 10 log (Number). In the third logical DataType, **RadiantFlux[dBW],** the reference unit is watts rather than milliwatts.

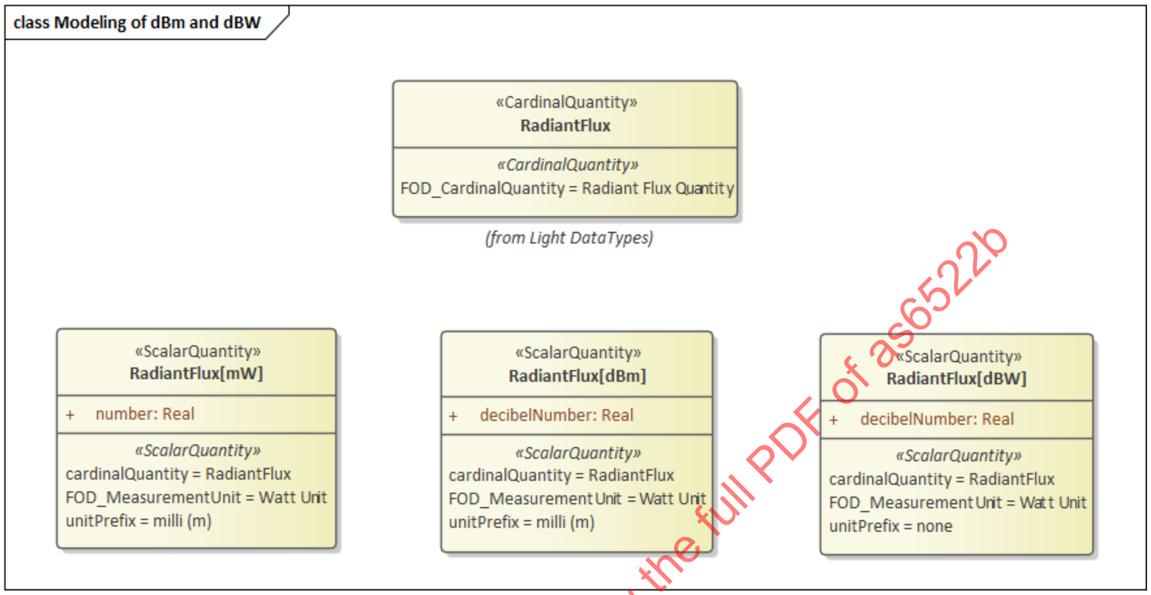


Figure 8 - Example logical DataTypes for radiant flux

End Example

5.2.6 Policy History

Orignal at Revision A.

5.3 Policy on Conceptual Model Package Structure

5.3.1 Policy Usage and Conformance

This policy applies to the **Conceptual Model** Package in the **UCS Architectural Model**. The Package shall comply with the policy.

5.3.2 Policy Description and Motivation

The Conceptual Model uses conceptual DataTypes as defined in the DMF. See policies 5.1 and 5.2. The **Conceptual Model** Package contains three Packages. These are the **Conceptual Data Model**, the **Conceptual Message Model**, and the **Conceptual Service Model**. This policy describes each Package and the dependencies between Packages.

5.3.3 Policy Constraints

None.

5.3.4 Policy

The Package structure of the **Conceptual Model** is shown in Figure 9.

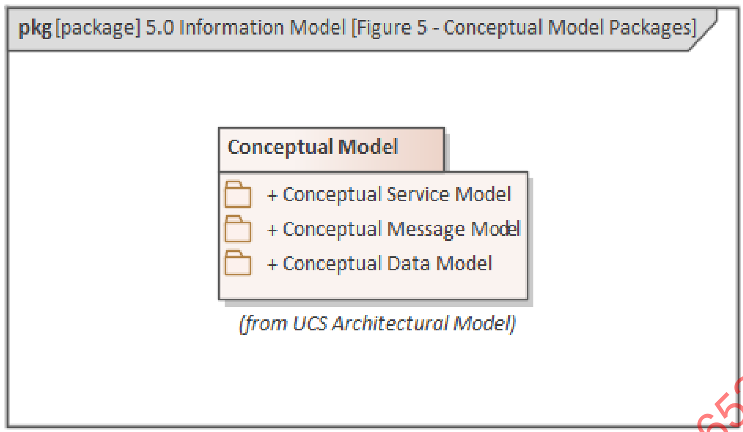

pkg [package] 5.0 Information Model [Figure 5 - Conceptual Model Packages]
Conceptual Model
+ Conceptual Service Model
+ Conceptual Message Model
+ Conceptual Data Model
(from UCS Architectural Model)

Figure 9 - Conceptual Model Packages

The **Conceptual Data Model** Package (CDM) is a definitional model of entities in the UCS domain. It identifies each entity together with its innate Properties including states and entity-entity relationships. The library of CDM DataTypes is provided in the **Conceptual Data Types** Package, whose basic structure conforms to the DMF. The **Conceptual Entity Types** Package contains a **Common** Package plus **Air**, **Ground**, and **Maritime** Packages. See Policy 5.4

The **Conceptual Message Model** Package is an abstract model that identifies conceptual messages where each message is a view of the CDM that exposes content of interest to one or more services in the **Conceptual Service Model**. Each Attribute in the **Conceptual Message Model** is a Projection of an Attribute in the CDM. See Policy 5.5

The **Conceptual Service Model** Package is an abstract model that identifies service Participants, where each Participant has one or more Ports that are Typed by a ServiceInterface. Each Signal is parameterized by a message in the **Conceptual Message Model**. See Policy 5.6.

5.3.5 Policy Example(s)

Begin Example

Figure 10 is an example of a conceptual entity in the CDM called **FuelSystemType**. It does not show inherited Features from **SubsystemType**, etc. This definitional model identifies the innate Properties of a fuel system. Note that the DataType **MassCapabilityType** is composite DataType whose Attributes are ultimately Typed **Mass**. Each Attribute represents a different modality of **FuelSystemType: fuelLoad: Mass** such as its observable value (**mass**: **Mass**), its range of possible values based on the capacity of the fuel system, etc.

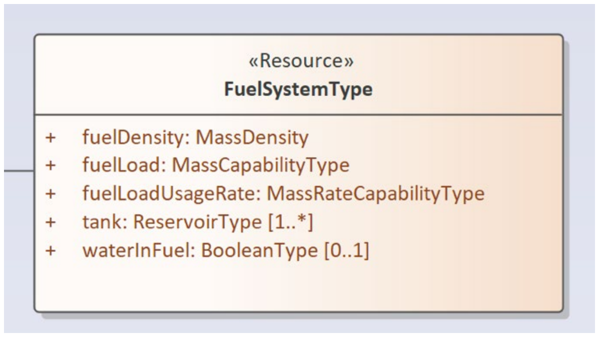

«Resource»
FuelSystemType
+ fuelDensity: MassDensity
+ fuelLoad: MassCapabilityType
+ fuelLoadUsageRate: MassRateCapabilityType
+ tank: ReservoirType [1..*]
+ waterInFuel: BooleanType [0..1]

Figure 10 - Example Conceptual Entity

End Example

Begin Example

Figure 11 is an example of conceptual message in the CDM called **VehicleStatusType**. This view of the CDM exposes part of the CDM via Projection Connectors. The Attribute (message payload) **propulsionFuelLoad: Mass** projects to the entity **AirVehicleSystemType** and has the projection path **propulsion.fuelSystem.fuelLoad.mass**.

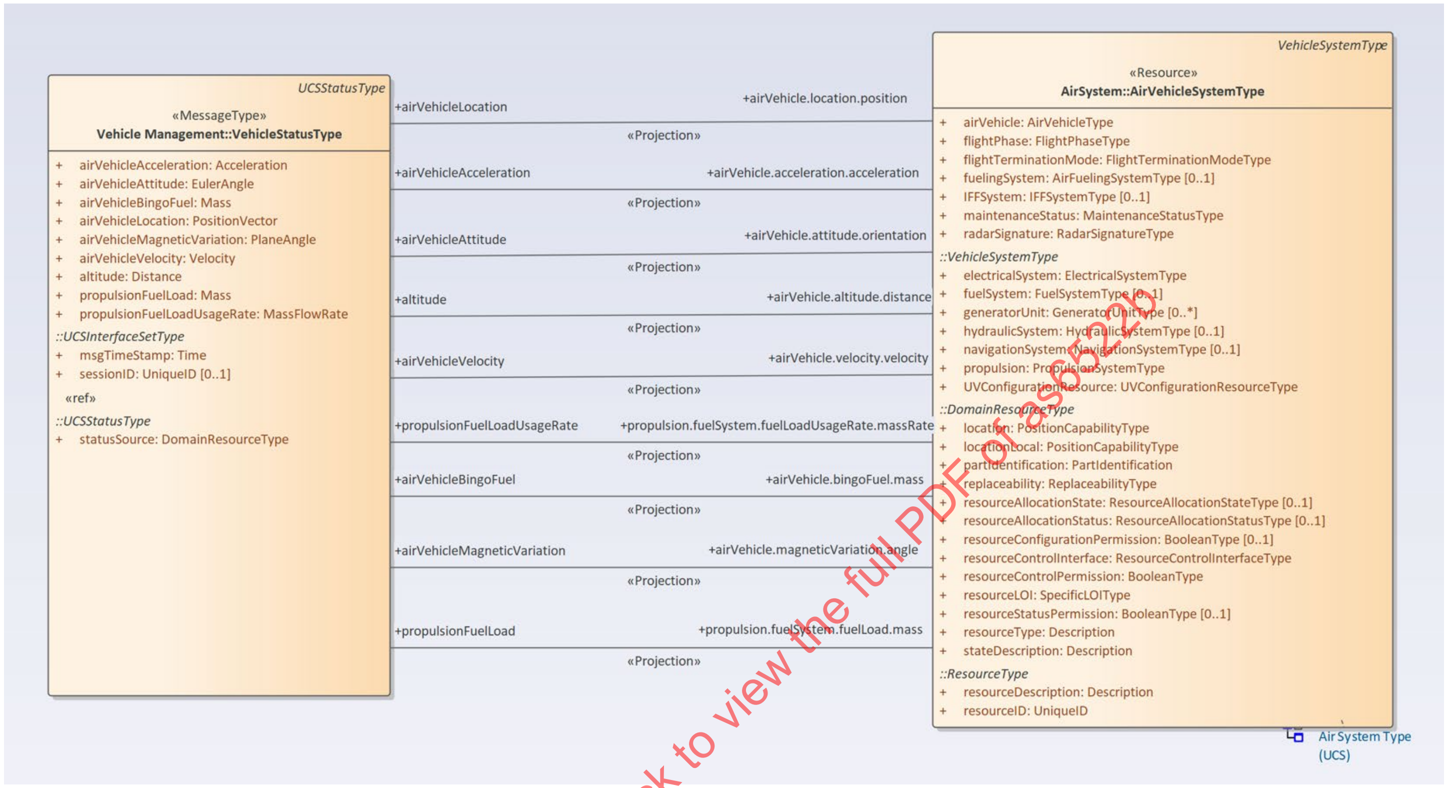

UCSStatusType
«MessageType»
Vehicle Management::VehicleStatusType
+ airVehicleAcceleration: Acceleration
+ airVehicleAttitude: EulerAngle
+ airVehicleBingoFuel: Mass
+ airVehicleLocation: PositionVector
+ airVehicleMagneticVariation: PlaneAngle
+ airVehicleVelocity: Velocity
+ altitude: Distance
+ propulsionFuelLoad: Mass
+ propulsionFuelLoadUsageRate: MassFlowRate
::UCSInterfaceSetType
+ msgTimeStamp: Time
+ sessionID: UniqueID [0..1]
«ref»
::UCSStatusType
+ statusSource: DomainResourceType
+airVehicleLocation
+airVehicle.location.position
«Projection»
+airVehicleAcceleration
+airVehicle.acceleration.acceleration
«Projection»
+airVehicleAttitude
+airVehicle.attitude.orientation
«Projection»
+altitude
+airVehicle.altitude.distance
«Projection»
+airVehicleVelocity
+airVehicle.velocity.velocity
«Projection»
+propulsionFuelLoadUsageRate
+propulsion.fuelSystem.fuelLoadUsageRate.massRate
«Projection»
+airVehicleBingoFuel
+airVehicle.bingoFuel.mass
«Projection»
+airVehicleMagneticVariation
+airVehicle.magneticVariation.angle
«Projection»
+propulsionFuelLoad
+propulsion.fuelSystem.fuelLoad.mass
«Projection»
VehicleSystemType
«Resource»
AirSystem::AirVehicleSystemType
+ airVehicle: AirVehicleType
+ flightPhase: FlightPhaseType
+ flightTerminationMode: FlightTerminationModeType
+ fuelingSystem: AirFuelingSystemType [0..1]
+ IFFSystem: IFFSystemType [0..1]
+ maintenanceStatus: MaintenanceStatusType
+ radarSignature: RadarSignatureType
::VehicleSystemType
+ electricalSystem: ElectricalSystemType
+ fuelSystem: FuelSystemType [0..1]
+ generatorUnit: GeneratorUnitType [0..*]
+ hydraulicSystem: HydraulicSystemType [0..1]
+ navigationSystem: NavigationSystemType [0..1]
+ propulsion: PropulsionSystemType
+ UVConfigurationResource: UVConfigurationResourceType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
Air System Type (UCS)

Figure 11 - VehicleStatusType:: propulsionFuelLoad: Mass

End Example

Begin Example

Figure 12 is an example of a conceptual ServiceInterface called **VehicleFlightStatus**. This Types a Service Port owned by a Participant also called **VehicleFlightStatus**. The responses Interface has various Signals that are parameterized by the message **VehicleStatusType**. For example, the Signal **VehicleFlightStatusRprt**.

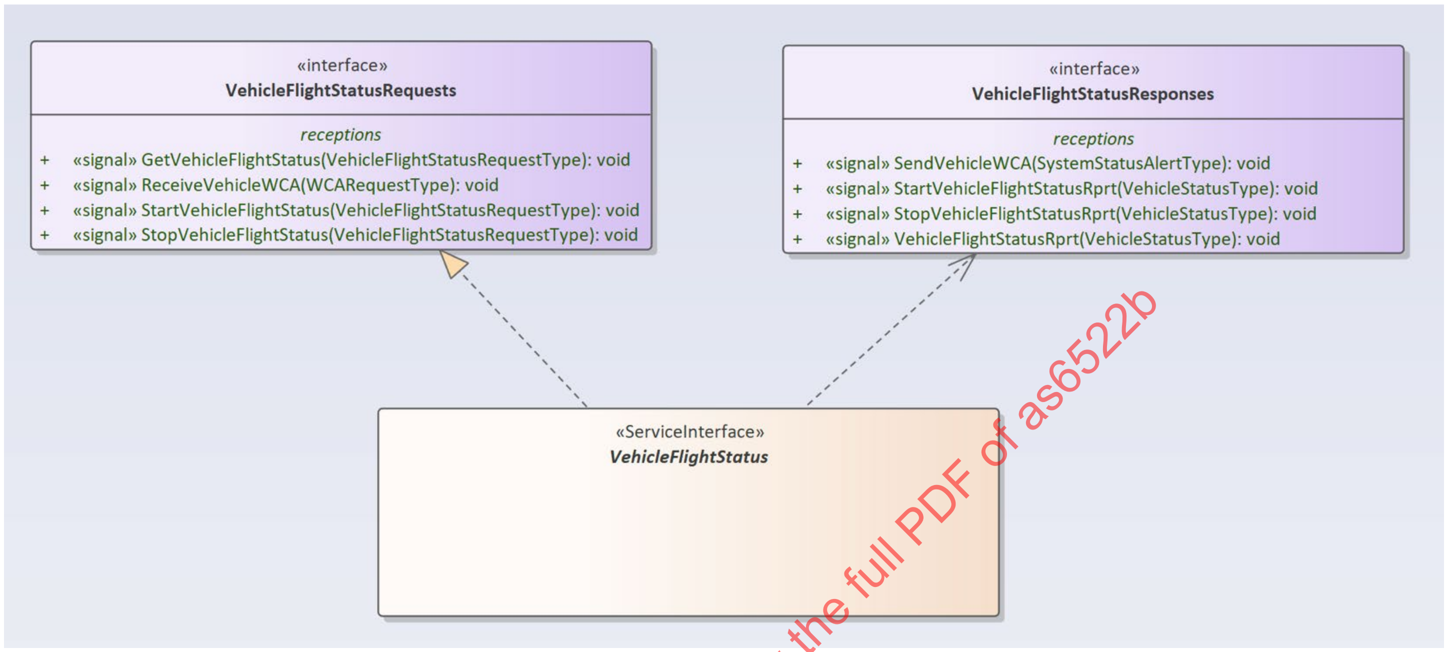

«interface»
VehicleFlightStatusRequests
receptions
+ «signal» GetVehicleFlightStatus(VehicleFlightStatusRequestType): void
+ «signal» ReceiveVehicleWCA(WCARequestType): void
+ «signal» StartVehicleFlightStatus(VehicleFlightStatusRequestType): void
+ «signal» StopVehicleFlightStatus(VehicleFlightStatusRequestType): void
«interface»
VehicleFlightStatusResponses
receptions
+ «signal» SendVehicleWCA(SystemStatusAlertType): void
+ «signal» StartVehicleFlightStatusRprt(VehicleStatusType): void
+ «signal» StopVehicleFlightStatusRprt(VehicleStatusType): void
+ «signal» VehicleFlightStatusRprt(VehicleStatusType): void
«ServiceInterface»
VehicleFlightStatus

Figure 12 - VehicleFlightStatus ServiceInterface

End Example

5.3.6 Policy History

Original at Revision A.

5.4 Policy on Conceptual Data Model (CDM)

5.4.1 Policy Usage and Conformance

This policy applies to the **Conceptual Data Model** Package in the **Conceptual Model** Package in the **UCS Architectural Model**. The Package shall conform to the policy.

5.4.2 Policy Description and Motivation

The **Conceptual Data Model** Package (CDM) is a definitional model of entities in the UCS domain. It identifies each entity together with its innate Properties including states and entity-entity relationships. It contains a **Conceptual Entity Model** Package and a **Conceptual Data Types** Package. In the **Conceptual Model**, the purpose of conceptual DataType is to scope the semantics of a Property. The DataType's value structure is provided in the **Logical Model**.

5.4.3 Policy Constraints

None.

5.4.4 Policy

The **Conceptual Entity Types** Package contains a **Common** Package plus domain-specific **Air**, **Ground**, and **Maritime** Packages. The structure of the **Common** Package is replicated in the domain-specific Packages as needed to provide domain-specific content. The Packages in the Common Package are shown in Figure 13.

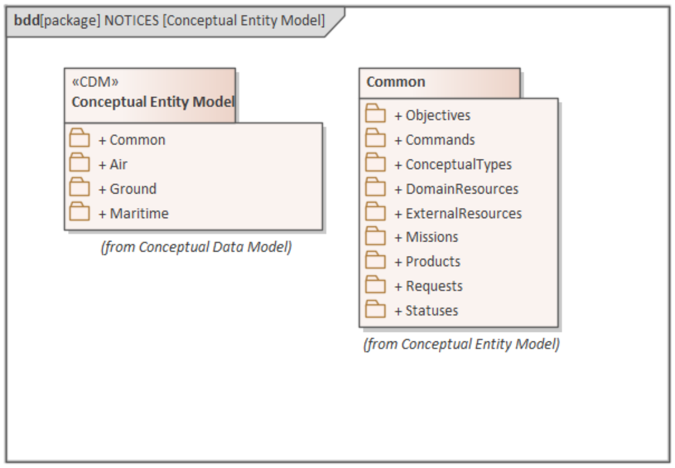


Figure 13 - Packages in the Common Package of Conceptual Entity Types

These Packages are supported by the Class Stereotypes **Objective**, **Mission**, and **Resource** as shown in Figure 14.

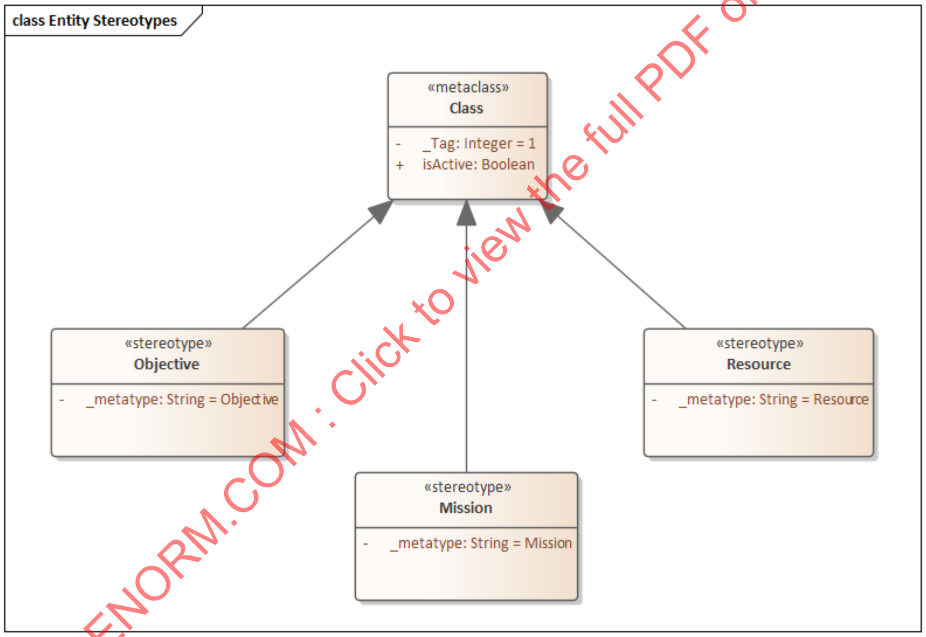


Figure 14 - UCS Stereotypes Used in Conceptual Entity Types

The **Resource** Stereotype principally applies to the **DomainResources** Package and **ExternalResources** Package as well as to the **Products**, **Commands**, **Requests**, and **Statuses** Packages.

Domain resources are entities that describe persistent resources within the span of control of the UCS domain. All domain resources Specialize from **DomainResourceType** as shown in Figure 15. The **UCSSystemType** has the **UVSystemType** (unmanned vehicle system) and **ControlSystemType** as Parts. External resources are entities describing persistent resources outside the control of the UCS domain, such as externally created information products. All external resources Specialize from **ExternalResourceType**.

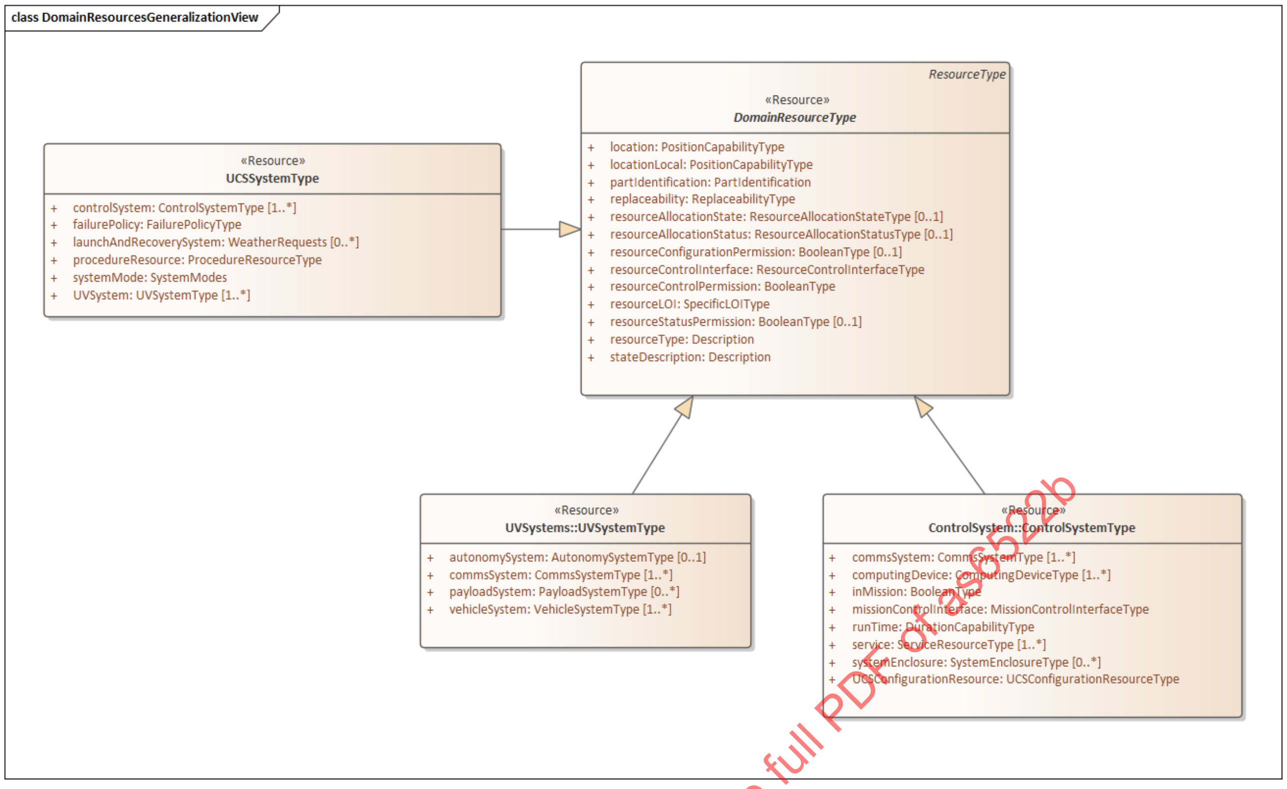

class DomainResourcesGeneralizationView
ResourceType
«Resource»
DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
«Resource»
UCSSystemType
+ controlSystem: ControlSystemType [1..*]
+ failurePolicy: FailurePolicyType
+ launchAndRecoverySystem: WeatherRequests [0..*]
+ procedureResource: ProcedureResourceType
+ systemMode: SystemModes
+ UVSystem: UVSystemType [1..*]
«Resource»
UVSystems::UVSystemType
+ autonomySystem: AutonomySystemType [0..1]
+ commsSystem: CommsSystemType [1..*]
+ payloadSystem: PayloadSystemType [0..*]
+ vehicleSystem: VehicleSystemType [1..*]
«Resource»
ControlSystem::ControlSystemType
+ commsSystem: CommsSystemType [1..*]
+ computingDevice: ComputingDeviceType [1..*]
+ inMission: BooleanType
+ missionControlInterface: MissionControlInterfaceType
+ runTime: DurationCapabilityType
+ service: ServiceResourceType [1..*]
+ systemEnclosure: SystemEnclosureType [0..*]
+ UCSConfigurationResource: UCSConfigurationResourceType

Figure 15 - DomainResourceType and Principal Parts of UCS System

The **Products** Package contains persistent information products that are produced by domain resources. They Specialize from **DataProductType** as shown in Figure 16.

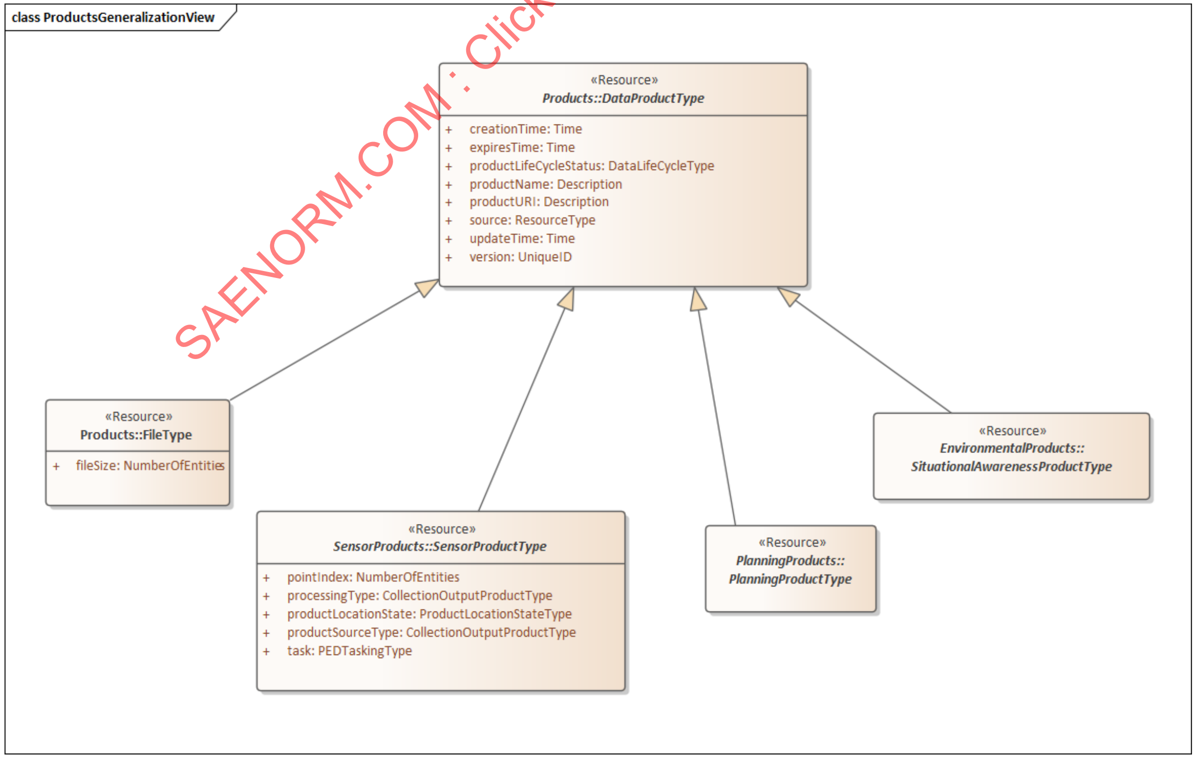

class ProductsGeneralizationView
«Resource»
Products::DataProductType
+ creationTime: Time
+ expiresTime: Time
+ productLifeCycleStatus: DataLifeCycleType
+ productName: Description
+ productURI: Description
+ source: ResourceType
+ updateTime: Time
+ version: UniqueID
«Resource»
Products::FileType
+ fileSize: NumberOfEntities
«Resource»
SensorProducts::SensorProductType
+ pointIndex: NumberOfEntities
+ processingType: CollectionOutputProductType
+ productLocationState: ProductLocationStateType
+ productSourceType: CollectionOutputProductType
+ task: PEDTaskingType
«Resource»
PlanningProducts::
PlanningProductType
«Resource»
EnvironmentalProducts::
SituationalAwarenessProductType

Figure 16 - DataProductsType and its Principal Specializations

The **Commands, Requests and Statuses** Packages contain persistent information items that ultimately Specialize from **UCSPersistentView**. Commands Specialize from **UCSPersistentCommand,** requests Specialize from **UCSPersistentRequest**, and statuses Specialize from **UCSPersistentStatus**.

The **Objective** Stereotype is applied to a entities that define domain resource objectives and actions. In the CDM, objective Types Specialize from **UCSObjectiveType** and are clearly defined, decisive, and attainable goals toward which every operation is directed. Action Types Specialize from **ActionType** and are clearly defined data necessary to command a resource. See Figure 17.

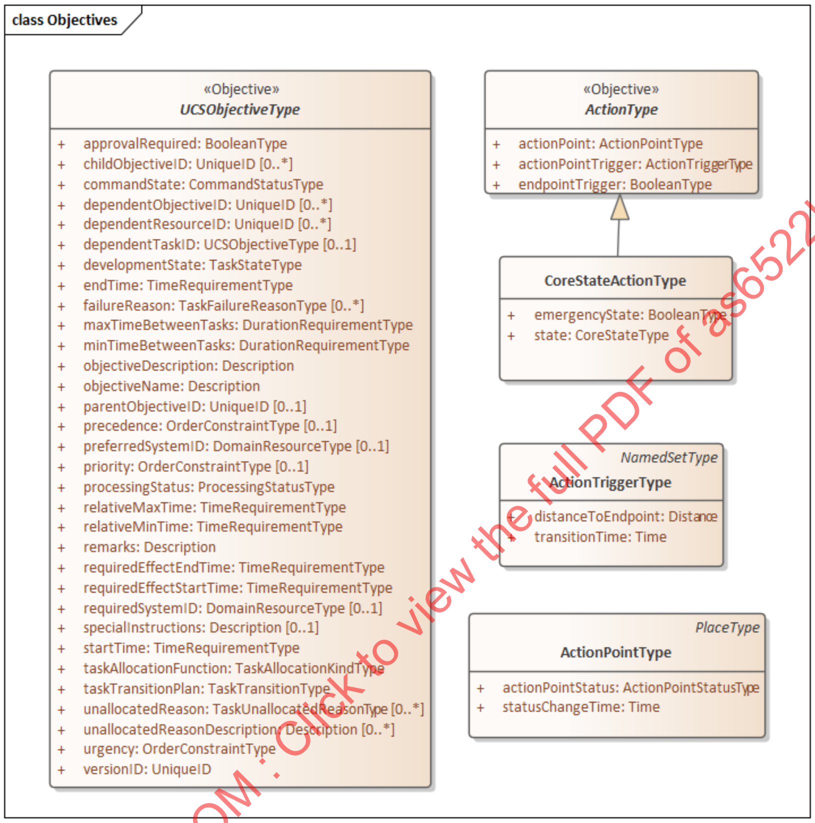

class Objectives
«Objective»
UCSObjectiveType
+ approvalRequired: BooleanType
+ childObjectiveID: UniqueID [0..*]
+ commandState: CommandStatusType
+ dependentObjectiveID: UniqueID [0..*]
+ dependentResourceID: UniqueID [0..*]
+ dependentTaskID: UCSObjectiveType [0..1]
+ developmentState: TaskStateType
+ endTime: TimeRequirementType
+ failureReason: TaskFailureReasonType [0..*]
+ maxTimeBetweenTasks: DurationRequirementType
+ minTimeBetweenTasks: DurationRequirementType
+ objectiveDescription: Description
+ objectiveName: Description
+ parentObjectiveID: UniqueID [0..1]
+ precedence: OrderConstraintType [0..1]
+ preferredSystemID: DomainResourceType [0..1]
+ priority: OrderConstraintType [0..1]
+ processingStatus: ProcessingStatusType
+ relativeMaxTime: TimeRequirementType
+ relativeMinTime: TimeRequirementType
+ remarks: Description
+ requiredEffectEndTime: TimeRequirementType
+ requiredEffectStartTime: TimeRequirementType
+ requiredSystemID: DomainResourceType [0..1]
+ specialInstructions: Description [0..1]
+ startTime: TimeRequirementType
+ taskAllocationFunction: TaskAllocationKindType
+ taskTransitionPlan: TaskTransitionType
+ unallocatedReason: TaskUnallocatedReasonType [0..*]
+ unallocatedReasonDescription: Description [0..*]
+ urgency: OrderConstraintType
+ versionID: UniqueID
«Objective»
ActionType
+ actionPoint: ActionPointType
+ actionPointTrigger: ActionTriggerType
+ endpointTrigger: BooleanType
CoreStateActionType
+ emergencyState: BooleanType
+ state: CoreStateType
NamedSetType
ActionTriggerType
+ distanceToEndpoint: Distance
+ transitionTime: Time
PlaceType
ActionPointType
+ actionPointStatus: ActionPointStatusType
+ statusChangeTime: Time

Figure 17 - UCSObjectiveType and ActionType

The **Mission** Stereotype applies to a set of entities that define missions. A Mission is an entity associating an objective to a domain resource. All missions Specialize from **UCSMissionType**. See Figure 18.

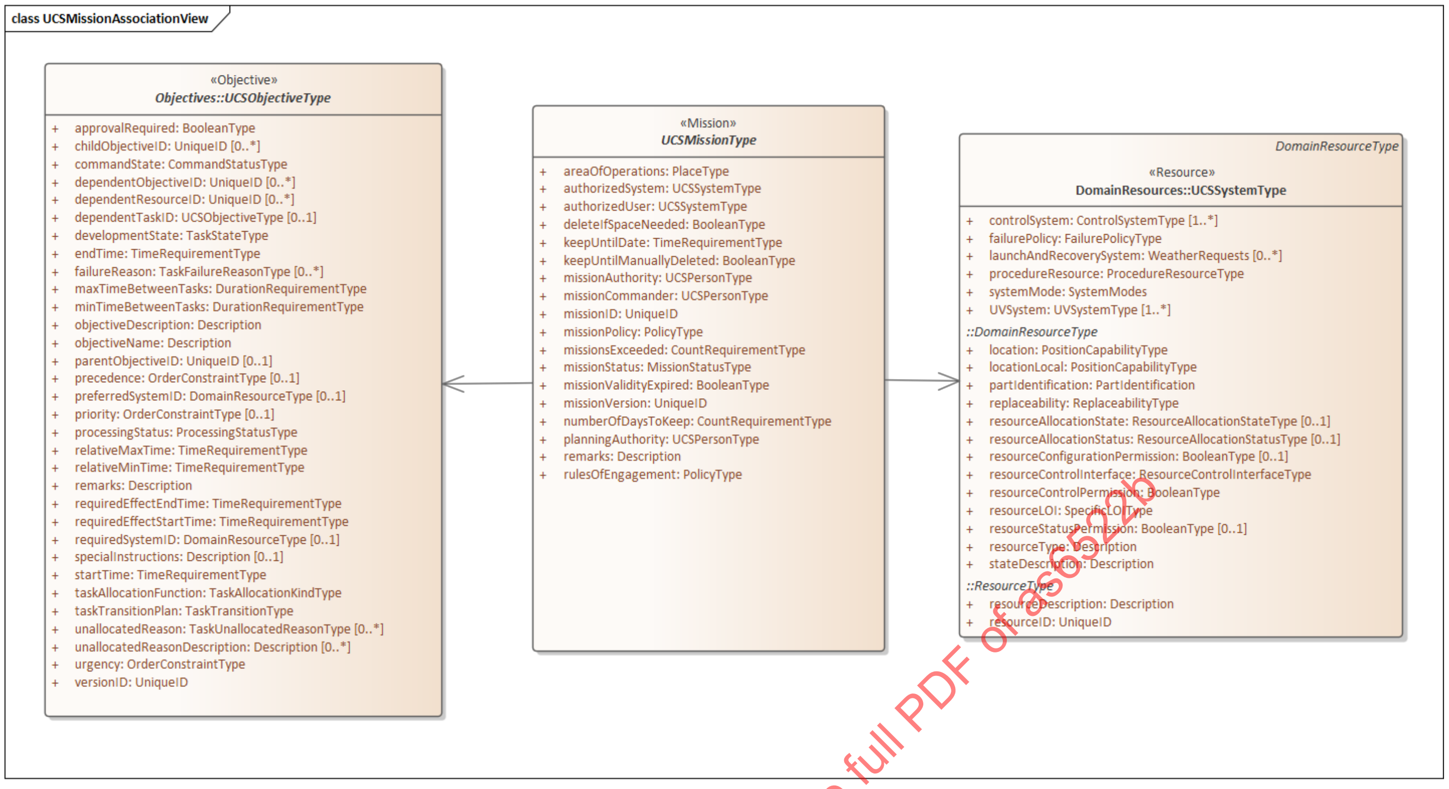

class UCSMissionAssociationView
«Objective»
Objectives::UCSObjectiveType
+ approvalRequired: BooleanType
+ childObjectiveID: UniqueID [0..*]
+ commandState: CommandStatusType
+ dependentObjectiveID: UniqueID [0..*]
+ dependentResourceID: UniqueID [0..*]
+ dependentTaskID: UCSObjectiveType [0..1]
+ developmentState: TaskStateType
+ endTime: TimeRequirementType
+ failureReason: TaskFailureReasonType [0..*]
+ maxTimeBetweenTasks: DurationRequirementType
+ minTimeBetweenTasks: DurationRequirementType
+ objectiveDescription: Description
+ objectiveName: Description
+ parentObjectiveID: UniqueID [0..1]
+ precedence: OrderConstraintType [0..1]
+ preferredSystemID: DomainResourceType [0..1]
+ priority: OrderConstraintType [0..1]
+ processingStatus: ProcessingStatusType
+ relativeMaxTime: TimeRequirementType
+ relativeMinTime: TimeRequirementType
+ remarks: Description
+ requiredEffectEndTime: TimeRequirementType
+ requiredEffectStartTime: TimeRequirementType
+ requiredSystemID: DomainResourceType [0..1]
+ specialInstructions: Description [0..1]
+ startTime: TimeRequirementType
+ taskAllocationFunction: TaskAllocationKindType
+ taskTransitionPlan: TaskTransitionType
+ unallocatedReason: TaskUnallocatedReasonType [0..*]
+ unallocatedReasonDescription: Description [0..*]
+ urgency: OrderConstraintType
+ versionID: UniqueID
«Mission»
UCSMissionType
+ areaOfOperations: PlaceType
+ authorizedSystem: UCSSystemType
+ authorizedUser: UCSSystemType
+ deleteIfSpaceNeeded: BooleanType
+ keepUntilDate: TimeRequirementType
+ keepUntilManuallyDeleted: BooleanType
+ missionAuthority: UCSPersonType
+ missionCommander: UCSPersonType
+ missionID: UniqueID
+ missionPolicy: PolicyType
+ missionsExceeded: CountRequirementType
+ missionStatus: MissionStatusType
+ missionValidityExpired: BooleanType
+ missionVersion: UniqueID
+ numberOfDaysToKeep: CountRequirementType
+ planningAuthority: UCSPersonType
+ remarks: Description
+ rulesOfEngagement: PolicyType
DomainResourceType
«Resource»
DomainResources::UCSSystemType
+ controlSystem: ControlSystemType [1..*]
+ failurePolicy: FailurePolicyType
+ launchAndRecoverySystem: WeatherRequests [0..*]
+ procedureResource: ProcedureResourceType
+ systemMode: SystemModes
+ UVSystem: UVSystemType [1..*]
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID

Figure 17 - UCSMissionType

The **Conceptual Data Types** Package contains the Packages shown in Figure 19.

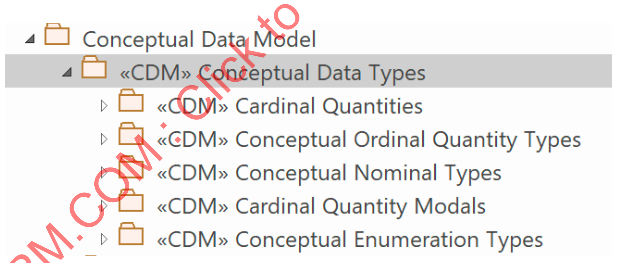

Conceptual Data Model
«CDM» Conceptual Data Types
«CDM» Cardinal Quantities
«CDM» Conceptual Ordinal Quantity Types
«CDM» Conceptual Nominal Types
«CDM» Cardinal Quantity Modals
«CDM» Conceptual Enumeration Types

Figure 19 - Packages in Conceptual Data Types Package

These Packages are migrating towards applying the **DataType** Profile of the Data Model Framework starting in Revision A. Refer to DMF Technical Governance, policies 8.03 (CDM DataTypes Package), 8.04 (Conceptual DataTypes), and 8.05 (Conceptual Enumerations) for the development of new content. As stated elsewhere, the purpose of a conceptual DataType is only to scope the semantics of a Property in the CDM and its Projection into the **Conceptual Message Model**. The Value structure of the Property is provided in the logical refinement.

The **Cardinal Quantities** Package is a library of DataTypes for cardinal quantities as defined in AS6969 and DMF Technical Governance Policy 9.03 (Cardinal Quantities). These DataTypes are used to Type categorical Measurands and their modalities. The **Cardinal Quantity Modals** Package is a library of cardinal quantity DataTypes whose Attributes are modals. The concept of modalities and modals is defined in AS6969 and in DMF Technical Goverance Policy 8.11 (Property Modals). The current structure of this Package, however, is not fully aligned with the DMF. The **Conceptual Ordinal Quantity Types** Package is a library DataTypes for ordinal quantities as defined in AS6969A and DMF Technical Goverance Policy 9.04 (Ordinal Quantities).

The **Conceptual Nominals Types** Package is a library of DataTypes for unenumerated nominal properties as defined in AS6969 and DMF Technical Governance Policy 9.05 (Nominal Properties). The **Conceptual Enumeration** Types Package is a library of conceptual Enumerations. See DMF Technical Governance Policy 8.05 (Conceptual Enumerations). The current structure of this Package is not fully aligned with the DMF.

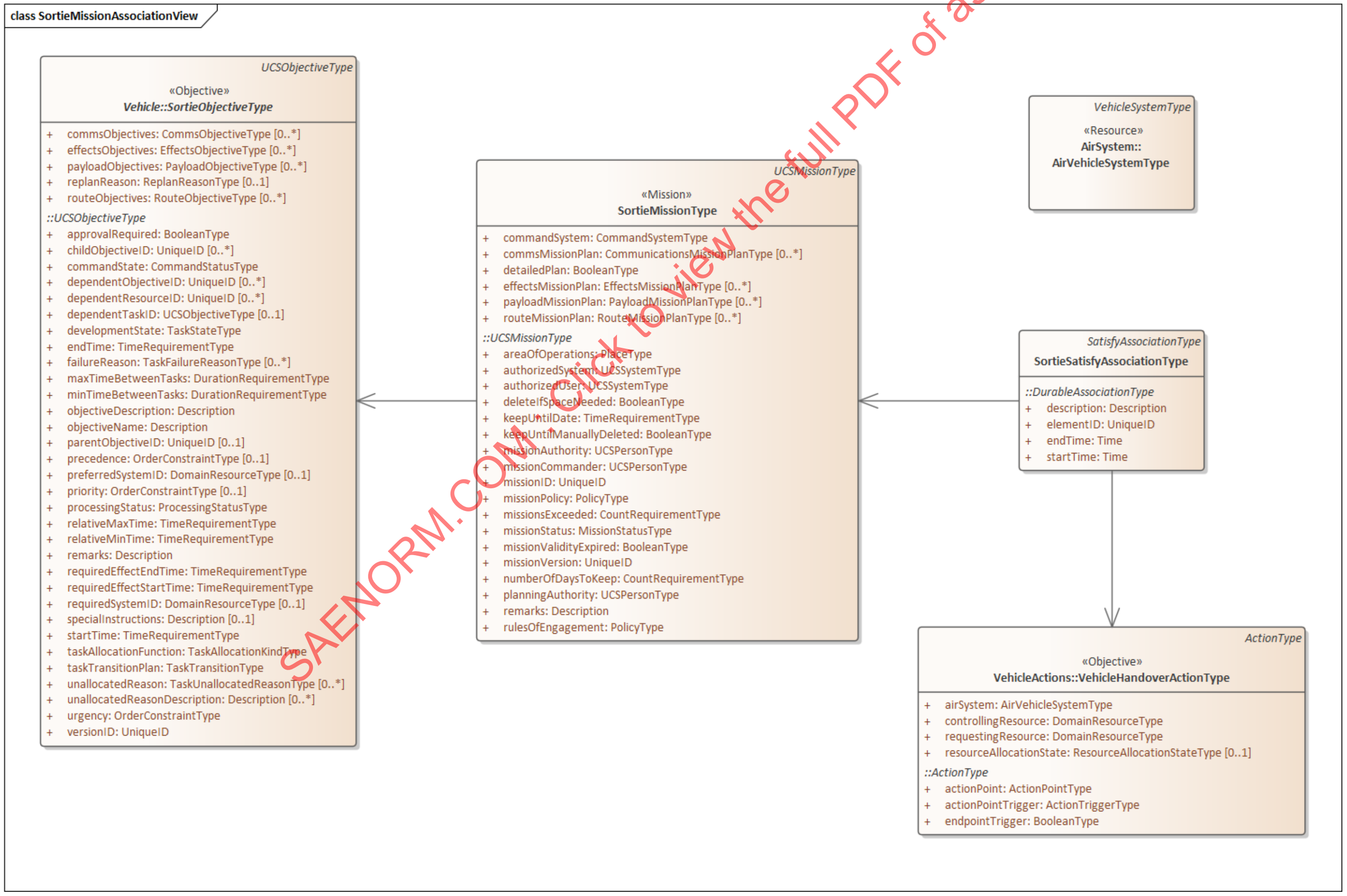
Naming Conventions in CDM

- Only uppercase (A to Z), lowercase (a to z), and digits (0 to 9) shall be used for names, packages, classes. CamelCase with upper-case initial character shall be used for Types. The lower-case initial character (camelCase) shall be used for Property (i.e. Attribute and Association) names only. Acronyms and abbreviations will be capitalized using their natural language form (e.g., GPS, COTS). Enumerated value names use upper case characters only. Nouns shall be singular. Verbs shall be present tense (unless tense is essential to meaning). Articles, prepositions, and conjunctions shall only be used where essential to meaning.

5.4.5 Policy Example(s)

Begin Example

Figure 20 shows the **SortieMissionType** in the **Air** Package of the CDM, including its Inherited Attributes. It is Associated with **SortieObjectiveType**. It Aggregates **CommunicationMissionPlanType[0..*]**, **EffectsMissionPlanType[0..*]**, **PayloadMissionPlan[0..*]**, and **RouteMissionPlan[0..*]**, each of which has an associated objective. The sortie mission can be handed over from its controlling **DomainResourceType** to a requesting **DomainResourceType** as identified in the **VehicleHandoverActionType**.



class SortieMissionAssociationView
UCSObjectiveType
«Objective»
Vehicle::SortieObjectiveType
+ commsObjectives: CommsObjectiveType [0..*]
+ effectsObjectives: EffectsObjectiveType [0..*]
+ payloadObjectives: PayloadObjectiveType [0..*]
+ replanReason: ReplanReasonType [0..1]
+ routeObjectives: RouteObjectiveType [0..*]
::UCSObjectiveType
+ approvalRequired: BooleanType
+ childObjectiveID: UniqueID [0..*]
+ commandState: CommandStatusType
+ dependentObjectiveID: UniqueID [0..*]
+ dependentResourceID: UniqueID [0..*]
+ dependentTaskID: UCSObjectiveType [0..1]
+ developmentState: TaskStateType
+ endTime: TimeRequirementType
+ failureReason: TaskFailureReasonType [0..*]
+ maxTimeBetweenTasks: DurationRequirementType
+ minTimeBetweenTasks: DurationRequirementType
+ objectiveDescription: Description
+ objectiveName: Description
+ parentObjectiveID: UniqueID [0..1]
+ precedence: OrderConstraintType [0..1]
+ preferredSystemID: DomainResourceType [0..1]
+ priority: OrderConstraintType [0..1]
+ processingStatus: ProcessingStatusType
+ relativeMaxTime: TimeRequirementType
+ relativeMinTime: TimeRequirementType
+ remarks: Description
+ requiredEffectEndTime: TimeRequirementType
+ requiredEffectStartTime: TimeRequirementType
+ requiredSystemID: DomainResourceType [0..1]
+ specialInstructions: Description [0..1]
+ startTime: TimeRequirementType
+ taskAllocationFunction: TaskAllocationKindType
+ taskTransitionPlan: TaskTransitionType
+ unallocatedReason: TaskUnallocatedReasonType [0..*]
+ unallocatedReasonDescription: Description [0..*]
+ urgency: OrderConstraintType
+ versionID: UniqueID
UCSMissionType
«Mission»
SortieMissionType
+ commandSystem: CommandSystemType
+ commsMissionPlan: CommunicationsMissionPlanType [0..*]
+ detailedPlan: BooleanType
+ effectsMissionPlan: EffectsMissionPlanType [0..*]
+ payloadMissionPlan: PayloadMissionPlanType [0..*]
+ routeMissionPlan: RouteMissionPlanType [0..*]
::UCSMissionType
+ areaOfOperations: PlaceType
+ authorizedSystem: UCSSystemType
+ authorizedUser: UCSSystemType
+ deleteIfSpaceNeeded: BooleanType
+ keepUntilDate: TimeRequirementType
+ keepUntilManuallyDeleted: BooleanType
+ missionAuthority: UCSPersonType
+ missionCommander: UCSPersonType
+ missionID: UniqueID
+ missionPolicy: PolicyType
+ missionsExceeded: CountRequirementType
+ missionStatus: MissionStatusType
+ missionValidityExpired: BooleanType
+ missionVersion: UniqueID
+ numberOfDaysToKeep: CountRequirementType
+ planningAuthority: UCSPersonType
+ remarks: Description
+ rulesOfEngagement: PolicyType
VehicleSystemType
«Resource»
AirSystem::
AirVehicleSystemType
SatisfyAssociationType
SortieSatisfyAssociationType
::DurableAssociationType
+ description: Description
+ elementID: UniqueID
+ endTime: Time
+ startTime: Time
ActionType
«Objective»
VehicleActions::VehicleHandoverActionType
+ airSystem: AirVehicleSystemType
+ controllingResource: DomainResourceType
+ requestingResource: DomainResourceType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
::ActionType
+ actionPoint: ActionPointType
+ actionPointTrigger: ActionTriggerType
+ endpointTrigger: BooleanType

Figure 20 - Air Sortie Mission Associations

End Example

Begin Example

Figure 21 shows the **WaypointMissionType**, which is Part of the **RouteMissionPlanType** identified in the previous example. It is Associated with the **WaypointObjectiveType** and is Associated with **VehicleWaypointActionType** via the **WaypointSatisfyAssociationType**.

The Measurands of the objectives and actions typically have a deontic modality (see AS6969). That is, they declare the required mission Value or contraints on the mission Value. The Measurands for resouces typically declare the categorical (i.e. observable) Value or some modal or set of modals.

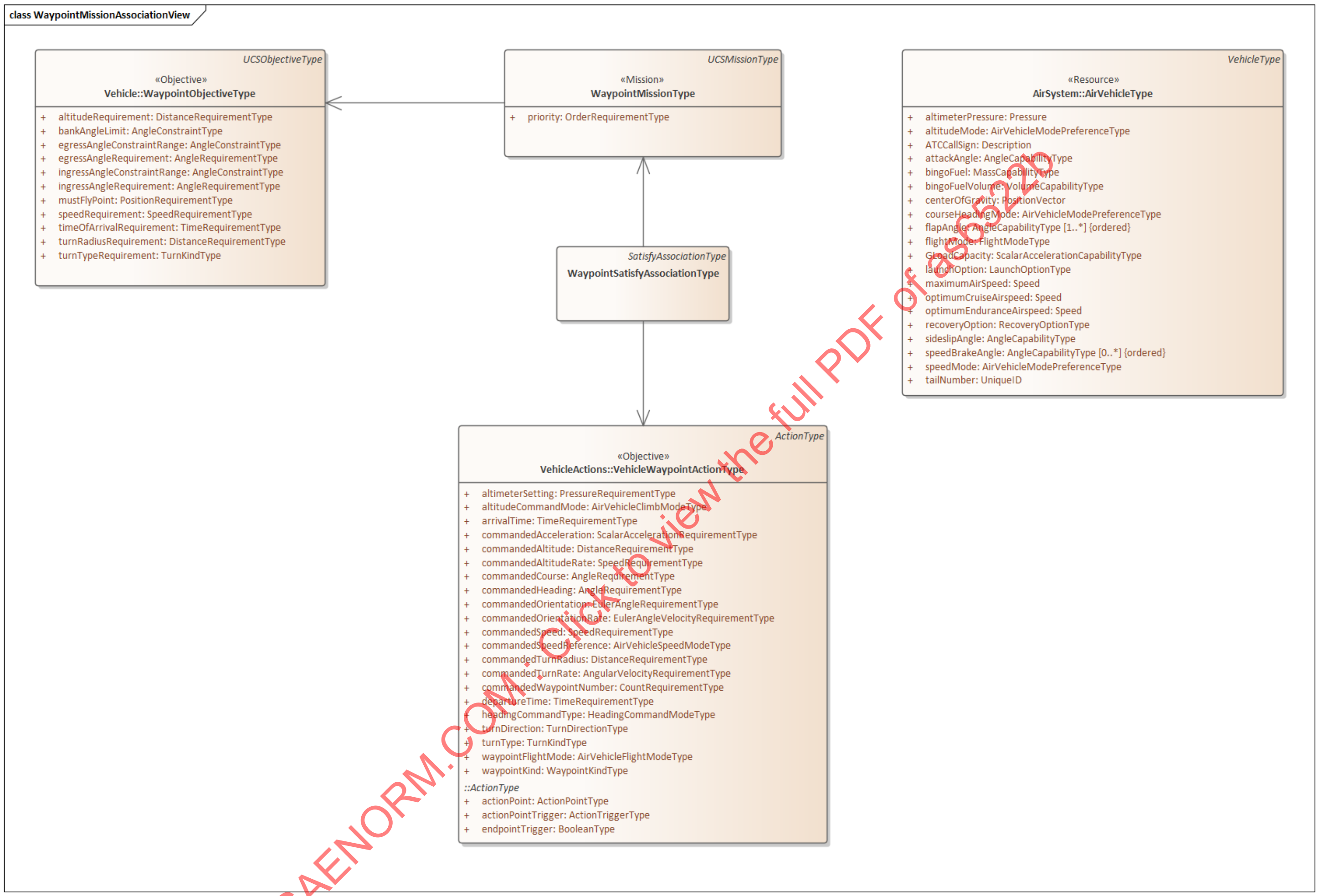

class WaypointMissionAssociationView
UCSObjectiveType
«Objective»
Vehicle::WaypointObjectiveType
+ altitudeRequirement: DistanceRequirementType
+ bankAngleLimit: AngleConstraintType
+ egressAngleConstraintRange: AngleConstraintType
+ egressAngleRequirement: AngleRequirementType
+ ingressAngleConstraintRange: AngleConstraintType
+ ingressAngleRequirement: AngleRequirementType
+ mustFlyPoint: PositionRequirementType
+ speedRequirement: SpeedRequirementType
+ timeOfArrivalRequirement: TimeRequirementType
+ turnRadiusRequirement: DistanceRequirementType
+ turnTypeRequirement: TurnKindType
UCSMissionType
«Mission»
WaypointMissionType
+ priority: OrderRequirementType
SatisfyAssociationType
WaypointSatisfyAssociationType
VehicleType
«Resource»
AirSystem::AirVehicleType
+ altimeterPressure: Pressure
+ altitudeMode: AirVehicleModePreferenceType
+ ATCCallSign: Description
+ attackAngle: AngleCapabilityType
+ bingoFuel: MassCapabilityType
+ bingoFuelVolume: VolumeCapabilityType
+ centerOfGravity: PositionVector
+ courseHeadingMode: AirVehicleModePreferenceType
+ flapAngle: AngleCapabilityType [1..*] {ordered}
+ flightMode: FlightModeType
+ GLoadCapacity: ScalarAccelerationCapabilityType
+ launchOption: LaunchOptionType
+ maximumAirSpeed: Speed
+ optimumCruiseAirspeed: Speed
+ optimumEnduranceAirspeed: Speed
+ recoveryOption: RecoveryOptionType
+ sideslipAngle: AngleCapabilityType
+ speedBrakeAngle: AngleCapabilityType [0..*] {ordered}
+ speedMode: AirVehicleModePreferenceType
+ tailNumber: UniqueID
ActionType
«Objective»
VehicleActions::VehicleWaypointActionType
+ altimeterSetting: PressureRequirementType
+ altitudeCommandMode: AirVehicleClimbModeType
+ arrivalTime: TimeRequirementType
+ commandedAcceleration: ScalarAccelerationRequirementType
+ commandedAltitude: DistanceRequirementType
+ commandedAltitudeRate: SpeedRequirementType
+ commandedCourse: AngleRequirementType
+ commandedHeading: AngleRequirementType
+ commandedOrientation: EulerAngleRequirementType
+ commandedOrientationRate: EulerAngleVelocityRequirementType
+ commandedSpeed: SpeedRequirementType
+ commandedSpeedReference: AirVehicleSpeedModeType
+ commandedTurnRadius: DistanceRequirementType
+ commandedTurnRate: AngularVelocityRequirementType
+ commandedWaypointNumber: CountRequirementType
+ departureTime: TimeRequirementType
+ headingCommandType: HeadingCommandModeType
+ turnDirection: TurnDirectionType
+ turnType: TurnKindType
+ waypointFlightMode: AirVehicleFlightModeType
+ waypointKind: WaypointKindType
::ActionType
+ actionPoint: ActionPointType
+ actionPointTrigger: ActionTriggerType
+ endpointTrigger: BooleanType

Figure 21 - Waypoint Mission Associations

End Example

5.4.6 Policy History

Original at Revision A.

5.5 Policy on Conceptual Message Model

5.5.1 Policy Usage and Conformance

This policy applies to the **Conceptual Message Model** Package in the **Conceptual Model** Package in the **UCS Architectural Model**. The Package shall conform to the policy.

5.5.2 Policy Description and Motivation

The **Conceptual Message Model** Package is an abstract model that identifies conceptual messages where each message is a view of the CDM that exposes content of interest to one or more services in the **Conceptual Service Model**. Each Attribute in the **Conceptual Message Model** is a Projection of an Attribute in the CDM.

5.5.3 Policy Constraints

None.

5.5.4 Policy

The Packages in the **Conceptual Message Model** are shown in Figure 22. The **Message Model** Package is organized according to the taxonomy of services in the **Conceptual Service Model** (see Policy 5.6) plus a **Common** Package for common messages. The **Message Model** defines each UCS message at the conceptual level.

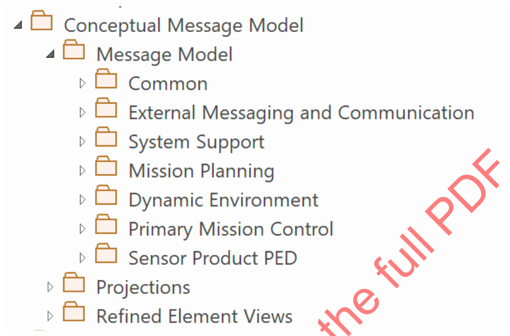


Figure 22 - Packages in Conceptual Message Model Package

The **Projections** Package shows how each message Attribute in the **Message Model** is a Projection of an Attribute in the **Conceptual Data Model**.

The Projection connector provides a direct correspondence between each Message Attribute and CDM Attribute (and its Type). This correspondence is called a Projection and is represented by a **Projection** Stereotype on a non-directed association. The source role is labeled with the name of the Message Attribute. The target role is labeled with a period-separated path terminating with the name of the Attribute in the CDM being Projected from. The path must be a valid list of AttributeNames, where each successive AttributeName is the AttributeName in the Type of the preceding Attribute.

There are guidelines on what should project to what. Table 1 describes the MessageType Classes that should project from various types of core elements. For example, the Attributes in status Messages (StatusTypes) generally report the state of some entity in the **Conceptual Data Model**. Attributes in command Messages (CommandTypes) usually include information used to update some commanded action (ActionType).

Table 1 - Projection connector guidelines

«MessageType» Classes	Projected From	Purpose
StatusTypes	ResourceTypes	StatusTypes provide state information about one or more resources or specializations of resources (i.e., ProductTypes).
CommandTypes	ActionTypes, MissionType	CommandTypes change the end state of resources. This is performed through a projection to an ActionType in order to satisfy an Objective.
RequestTypes	ResourceTypes	RequestTypes solicit state information from Resources (i.e., they do not change state of a Resource).

UCS ConversionProjection Connectors are like Projection Connectors, except that the Type of the “projected-from” Attribute is different from the Type of the Message Attribute. Some type and value conversion will be required. The UCS Refinement Connector, maps the CDM Types to the UCS LDM Types to indicate what types should be substituted for the UCS Realization of the abstract CDM. Because a Message Attribute with a ConversionProjection is not the same Type as the “projected-from” Attribute, the Attribute also requires a UCS Refinement Connector targeting the desired LDM Type to indicate what the “converted-to” LDM Type should be. Currently, the conversion functions that define the value conversions in a conversion projection are not defined.

The **Refined Element Views** Package Refines each conceptual AttributeType in the **Conceptual Data Model** to a logical Attribute Type. The process allows CDM AttributeTypes to have different AttributeTypes substituted from the **Logical Data Model** (LDM). The selection options are defined by the **Refine** Stereotype combined with the **UCS** Stereotype to produce a «refine,UCS» Dependency Connector.

5.5.5 Policy Example(s)

Begin Example

Figure 23 shows a Generalized Message, **UCSMissionStatusType,** whose Message Slot **UCSMission** is simply a Projection of **UCSMissionType**. This Type in the CDM is therefore an information exchange item contained within the Message.

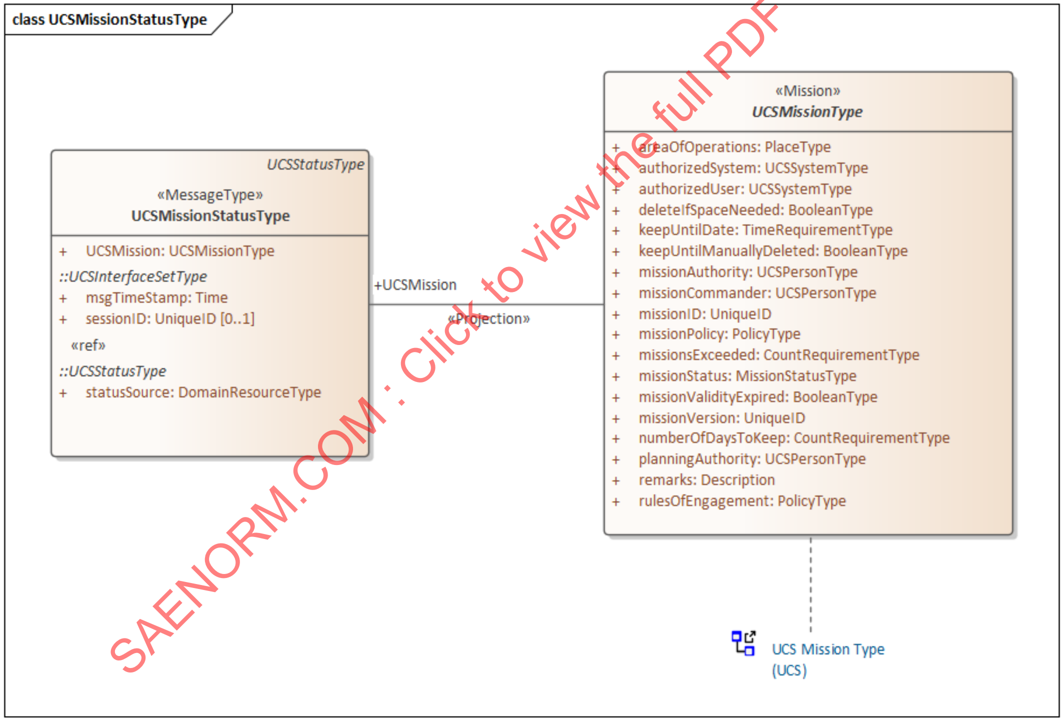

class UCSMissionStatusType
UCSStatusType
«MessageType»
UCSMissionStatusType
+ UCSMission: UCSMissionType
::UCSInterfaceSetType
+ msgTimeStamp: Time
+ sessionID: UniqueID [0..1]
«ref»
::UCSStatusType
+ statusSource: DomainResourceType
+UCSMission
«Projection»
«Mission»
UCSMissionType
+ areaOfOperations: PlaceType
+ authorizedSystem: UCSSystemType
+ authorizedUser: UCSSystemType
+ deleteIfSpaceNeeded: BooleanType
+ keepUntilDate: TimeRequirementType
+ keepUntilManuallyDeleted: BooleanType
+ missionAuthority: UCSPersonType
+ missionCommander: UCSPersonType
+ missionID: UniqueID
+ missionPolicy: PolicyType
+ missionsExceeded: CountRequirementType
+ missionStatus: MissionStatusType
+ missionValidityExpired: BooleanType
+ missionVersion: UniqueID
+ numberOfDaysToKeep: CountRequirementType
+ planningAuthority: UCSPersonType
+ remarks: Description
+ rulesOfEngagement: PolicyType
UCS Mission Type
(UCS)

Figure 23 - UCSMissionStatusType Projection

End Example

Figure 24 shows a Message, **SubsystemStatusReportType**. The Message Slot **UCSAirComponentPartIdentification** is a Projection to **SubsystemType:: partIdentification: PartIdentification** in the CDM. In this example, the Message SlotName is different to the CDM AttributeName.

Similarly, the Slot **UCSAirComponentSubsystemReport** is a Projection to **SubsystemType:: subsystemReport: SubsystemReportType[1..*].**

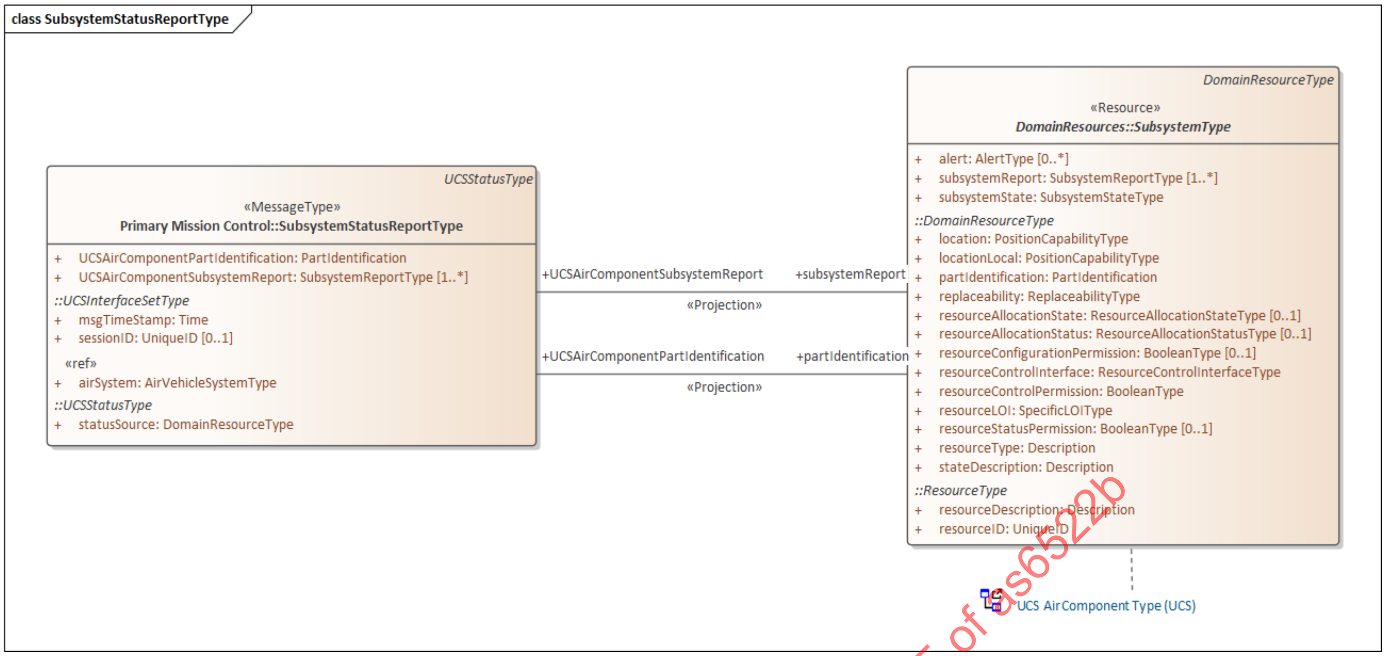

class SubsystemStatusReportType
UCSStatusType
«MessageType»
Primary Mission Control::SubsystemStatusReportType
+ UCSAirComponentPartIdentification: PartIdentification
+ UCSAirComponentSubsystemReport: SubsystemReportType [1..*]
::UCSInterfaceSetType
+ msgTimeStamp: Time
+ sessionID: UniqueID [0..1]
«ref»
+ airSystem: AirVehicleSystemType
::UCSStatusType
+ statusSource: DomainResourceType
+UCSAirComponentSubsystemReport
+subsystemReport
«Projection»
+UCSAirComponentPartIdentification
+partIdentification
«Projection»
DomainResourceType
«Resource»
DomainResources::SubsystemType
+ alert: AlertType [0..*]
+ subsystemReport: SubsystemReportType [1..*]
+ subsystemState: SubsystemStateType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
UCS AirComponent Type (UCS)

Figure 24 - SubsystemStatusReportType Projections

End Example

Begin Example

Figure 25 shows a Message with more complicated Projection paths. The Message Slot **airVehicleSpeedSetPoint** has the Projection path **airVehicle.speed.speedSetPoint**. The “airVehicle” part Projects to the Attribute **AirVehicleSystemType:: airVehicle: AirVehicleType**. The ‘speed’ part then Projects to the Attribute **AirVehicleType:: speed: SpeedCapabilityType**. The “speedSetPoint” part then Projects to the Attribute **SpeedCapabilityType:: speedSetPoint: Speed.**

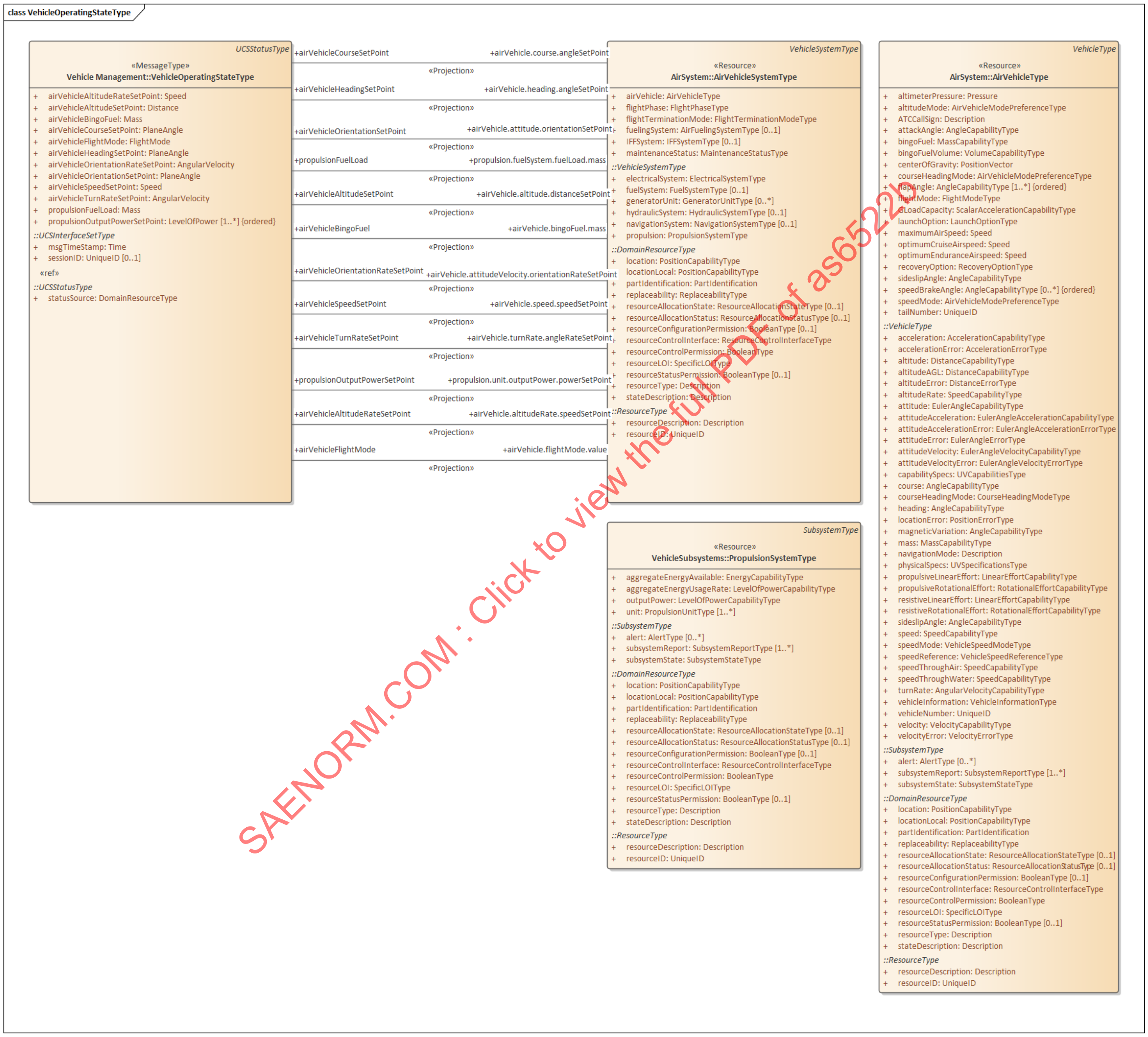

class VehicleOperatingStateType
UCSStatusType
«MessageType»
Vehicle Management::VehicleOperatingStateType
+ airVehicleAltitudeRateSetPoint: Speed
+ airVehicleAltitudeSetPoint: Distance
+ airVehicleBingoFuel: Mass
+ airVehicleCourseSetPoint: PlaneAngle
+ airVehicleFlightMode: FlightMode
+ airVehicleHeadingSetPoint: PlaneAngle
+ airVehicleOrientationRateSetPoint: AngularVelocity
+ airVehicleOrientationSetPoint: PlaneAngle
+ airVehicleSpeedSetPoint: Speed
+ airVehicleTurnRateSetPoint: AngularVelocity
+ propulsionFuelLoad: Mass
+ propulsionOutputPowerSetPoint: LevelOfPower [1..*] {ordered}
::UCSInterfaceSetType
+ msgTimeStamp: Time
+ sessionID: UniqueID [0..1]
«ref»
::UCSStatusType
+ statusSource: DomainResourceType
+airVehicleCourseSetPoint
+airVehicle.course.angleSetPoint
«Projection»
+airVehicleHeadingSetPoint
+airVehicle.heading.angleSetPoint
«Projection»
+airVehicleOrientationSetPoint
+airVehicle.attitude.orientationSetPoint
«Projection»
+propulsionFuelLoad
+propulsion.fuelSystem.fuelLoad.mass
«Projection»
+airVehicleAltitudeSetPoint
+airVehicle.altitude.distanceSetPoint
«Projection»
+airVehicleBingoFuel
+airVehicle.bingoFuel.mass
«Projection»
+airVehicleOrientationRateSetPoint
+airVehicle.attitudeVelocity.orientationRateSetPoint
«Projection»
+airVehicleSpeedSetPoint
+airVehicle.speed.speedSetPoint
«Projection»
+airVehicleTurnRateSetPoint
+airVehicle.turnRate.angleRateSetPoint
«Projection»
+propulsionOutputPowerSetPoint
+propulsion.unit.outputPower.powerSetPoint
«Projection»
+airVehicleAltitudeRateSetPoint
+airVehicle.altitudeRate.speedSetPoint
«Projection»
+airVehicleFlightMode
+airVehicle.flightMode.value
«Projection»
VehicleSystemType
«Resource»
AirSystem::AirVehicleSystemType
+ airVehicle: AirVehicleType
+ flightPhase: FlightPhaseType
+ flightTerminationMode: FlightTerminationModeType
+ fuelingSystem: AirFuelingSystemType [0..1]
+ IFFSystem: IFFSystemType [0..1]
+ maintenanceStatus: MaintenanceStatusType
::VehicleSystemType
+ electricalSystem: ElectricalSystemType
+ fuelSystem: FuelSystemType [0..1]
+ generatorUnit: GeneratorUnitType [0..*]
+ hydraulicSystem: HydraulicSystemType [0..1]
+ navigationSystem: NavigationSystemType [0..1]
+ propulsion: PropulsionSystemType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
SubsystemType
«Resource»
VehicleSubsystems::PropulsionSystemType
+ aggregateEnergyAvailable: EnergyCapabilityType
+ aggregateEnergyUsageRate: LevelOfPowerCapabilityType
+ outputPower: LevelOfPowerCapabilityType
+ unit: PropulsionUnitType [1..*]
::SubsystemType
+ alert: AlertType [0..*]
+ subsystemReport: SubsystemReportType [1..*]
+ subsystemState: SubsystemStateType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
VehicleType
«Resource»
AirSystem::AirVehicleType
+ altimeterPressure: Pressure
+ altitudeMode: AirVehicleModePreferenceType
+ ATCCallSign: Description
+ attackAngle: AngleCapabilityType
+ bingoFuel: MassCapabilityType
+ bingoFuelVolume: VolumeCapabilityType
+ centerOfGravity: PositionVector
+ courseHeadingMode: AirVehicleModePreferenceType
+ flapAngle: AngleCapabilityType [1..*] {ordered}
+ flightMode: FlightModeType
+ GLoadCapacity: ScalarAccelerationCapabilityType
+ launchOption: LaunchOptionType
+ maximumAirSpeed: Speed
+ optimumCruiseAirspeed: Speed
+ optimumEnduranceAirspeed: Speed
+ recoveryOption: RecoveryOptionType
+ sideslipAngle: AngleCapabilityType
+ speedBrakeAngle: AngleCapabilityType [0..*] {ordered}
+ speedMode: AirVehicleModePreferenceType
+ tailNumber: UniqueID
::VehicleType
+ acceleration: AccelerationCapabilityType
+ accelerationError: AccelerationErrorType
+ altitude: DistanceCapabilityType
+ altitudeAGL: DistanceCapabilityType
+ altitudeError: DistanceErrorType
+ altitudeRate: SpeedCapabilityType
+ attitude: EulerAngleCapabilityType
+ attitudeAcceleration: EulerAngleAccelerationCapabilityType
+ attitudeAccelerationError: EulerAngleAccelerationErrorType
+ attitudeError: EulerAngleErrorType
+ attitudeVelocity: EulerAngleVelocityCapabilityType
+ attitudeVelocityError: EulerAngleVelocityErrorType
+ capabilitySpecs: UVCapabilitiesType
+ course: AngleCapabilityType
+ courseHeadingMode: CourseHeadingModeType
+ heading: AngleCapabilityType
+ locationError: PositionErrorType
+ magneticVariation: AngleCapabilityType
+ mass: MassCapabilityType
+ navigationMode: Description
+ physicalSpecs: UVSpecificationsType
+ propulsiveLinearEffort: LinearEffortCapabilityType
+ propulsiveRotationalEffort: RotationalEffortCapabilityType
+ resistiveLinearEffort: LinearEffortCapabilityType
+ resistiveRotationalEffort: RotationalEffortCapabilityType
+ sideslipAngle: AngleCapabilityType
+ speed: SpeedCapabilityType
+ speedMode: VehicleSpeedModeType
+ speedReference: VehicleSpeedReferenceType
+ speedThroughAir: SpeedCapabilityType
+ speedThroughWater: SpeedCapabilityType
+ turnRate: AngularVelocityCapabilityType
+ vehicleInformation: VehicleInformationType
+ vehicleNumber: UniqueID
+ velocity: VelocityCapabilityType
+ velocityError: VelocityErrorType
::SubsystemType
+ alert: AlertType [0..*]
+ subsystemReport: SubsystemReportType [1..*]
+ subsystemState: SubsystemStateType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID

Figure 25 - VehicleOperatingStateType Projections

End Example

Begin Example

In Figure 26, the CDM resource **LandingGearType** has the Attribute **status: LandingGearStatusType**. The AttributeType is refined into **LandingGearStatusType_All.**

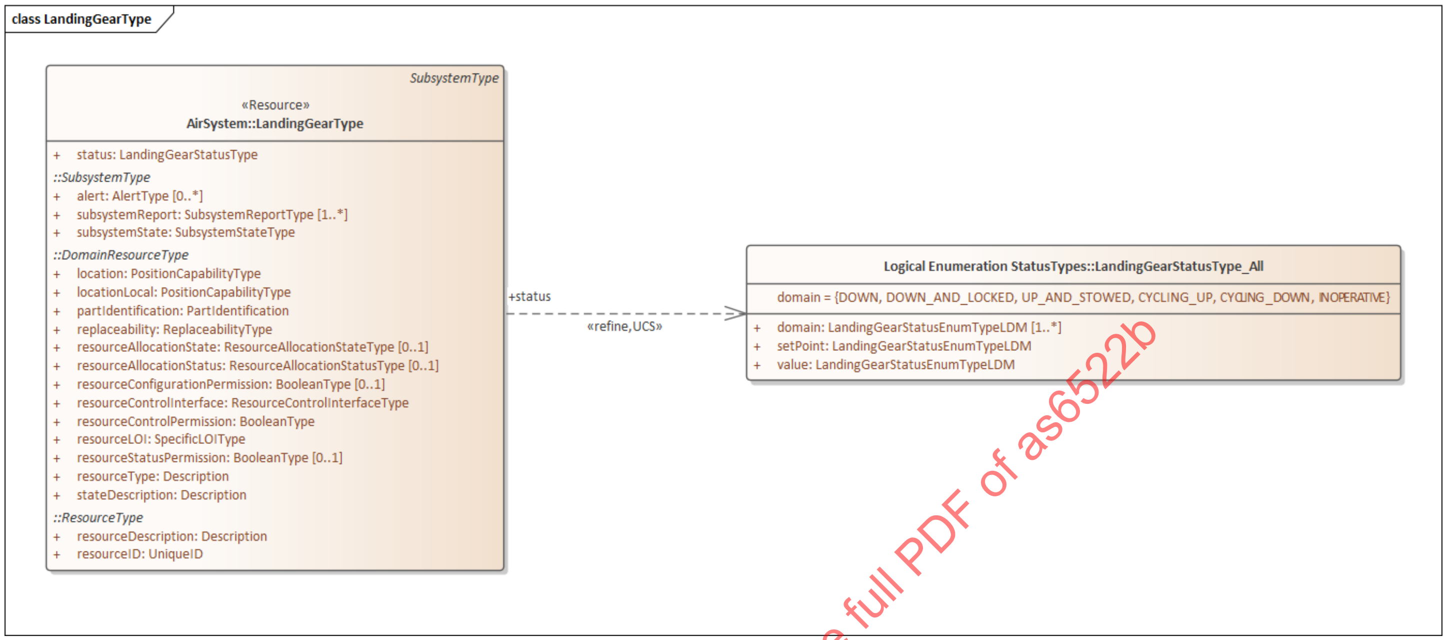

class LandingGearType
SubsystemType
«Resource»
AirSystem::LandingGearType
+ status: LandingGearStatusType
::SubsystemType
+ alert: AlertType [0..*]
+ subsystemReport: SubsystemReportType [1..*]
+ subsystemState: SubsystemStateType
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
+status
«refine,UCS»
Logical Enumeration StatusTypes::LandingGearStatusType_All
domain = {DOWN, DOWN_AND_LOCKED, UP_AND_STOWED, CYCLING_UP, CYCLING_DOWN, INOPERATIVE}
+ domain: LandingGearStatusEnumTypeLDM [1..*]
+ setPoint: LandingGearStatusEnumTypeLDM
+ value: LandingGearStatusEnumTypeLDM

Figure 26 - Refinement of LandingGearStatusType

End Example

5.5.6 Policy History

Original at Revision A.

5.6 Policy on Conceptual Service Model

5.6.1 Policy Usage and Conformance

This policy applies to the **Conceptual Service Model** Package of the **Conceptual Model** Package. This Package will conform to the policy.

5.6.2 Policy Description and Motivation

The **Conceptual Service Model** Package is an abstract model that identifies service Participants, where each Participant has one or more Ports that are Typed by ServiceInterfaces. These Ports are used to provide services or consume services. A service specification comprises a Class Diagram of the Participant and an Activity Diagram for each Service Port.

The Participants themselves are not normative and a conformant implementation of the UCS Architecture Model may have any Participant architecture. Only the ServiceInterface specifications are normative and the required Participant role in each ServiceInterface. The role is defined in the Notes Partition of the Participant Class. ServiceInterfaces have Signals as Features. Each Signal is parameterized by a message in the **Conceptual Message Model**.

5.6.3 Policy Constraints

None.

5.6.4 Policy

The Participant Class Diagrams Apply the OMG SoaML Profile as shown in Figure 27. The interpretation of each SoaML Stereotype is as follows.

- **Participant:** A role describing the provider or consumer of services.
- **Service Port:** A feature of a Participant that is offered by one Participant to others. A Participant designates a Port that defines the connection point through which a Participant offers its capabilities and provides a service to clients. The Service Port is Typed by a ServiceInterface.
- **Request Port:** A feature of a Participant that is used in the consumption of a service. A Request designates a port that defines the connection point through which a Participant meets its needs through the consumption of services provided by another Participant. The Request Port is Typed by a ServiceInterface.
- **ServiceInterface:** A ServiceInterface Types a Service Port or Request Port.
- **MessageType:** An Information Exchange Item that is conveyed across a ServiceInterface by a Signal.
- **ServicesArchitecture:** A description of how Participants work together using ServiceInterfaces. The UCS Services Architecture support the System Use Case Model but is not normative. Whereas simple services may not have any dependencies, UCS domain services are better understood in their context.
- **ServiceChannel:** A communication path between Participant Service Ports and Request Ports within an architecture. The UCS Architecture model uses the ServiceChannel to denote the explicit binding between Participants on ServicesArchitecture diagrams and to apply non-functional properties to these service channels.

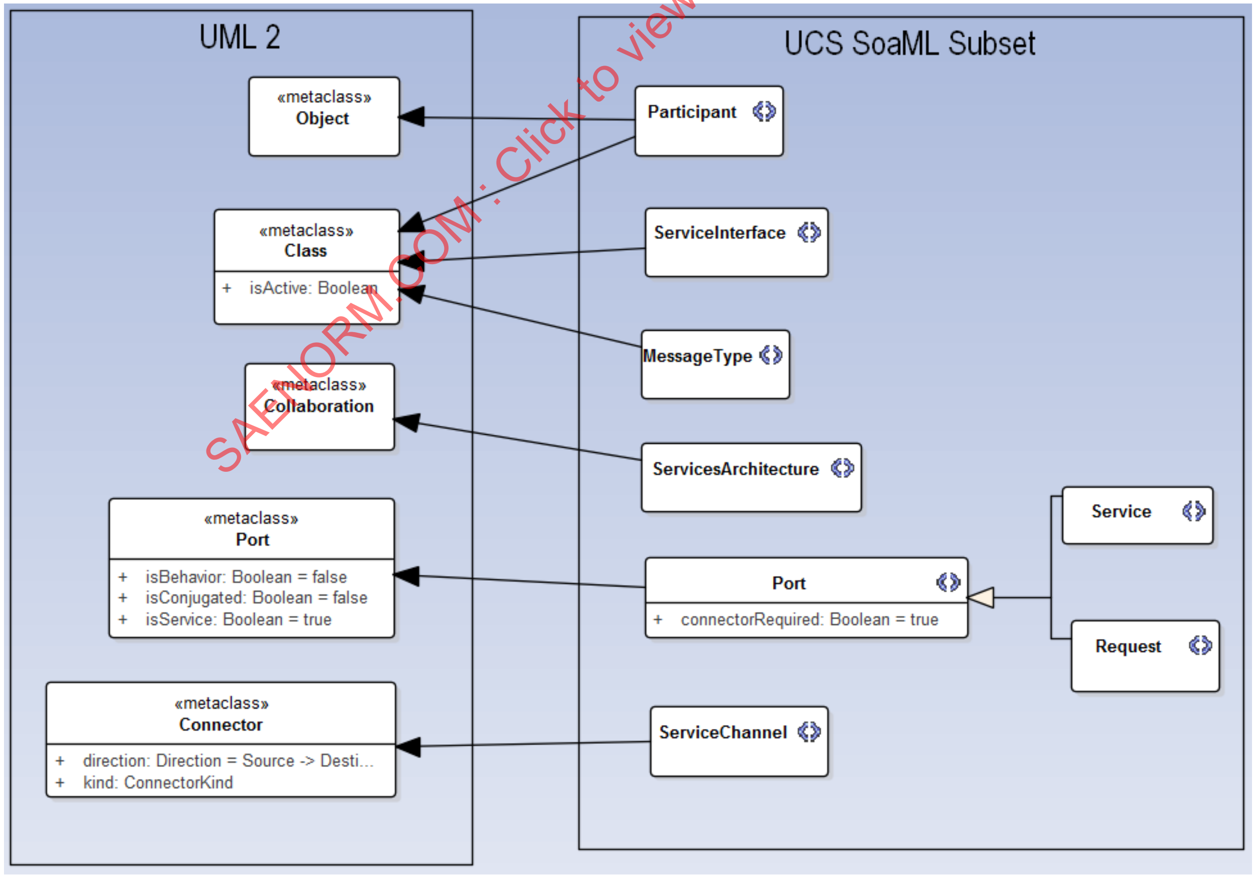


Figure 27 - UCS SoaML UML profile

A Protocol Activity Diagram is provided for each ServiceInterface. Protocol Activity Diagrams use only the following elements from UML Activity Diagrams.

- ActivityPartition (from IntermediateActivities), also known as “swimlanes”
- InitialNode (from BasicActivities)
- ActivityFinalNode (from BasicActivities)
- ControlFlow (from BasicActivities)
- DecisionNode (from IntermediateActivities)
- MergeNode (from IntermediateActivities)
- CallOperationAction (from BasicActions)

Since Protocol Activity Diagrams serve only to show the sequencing of messages/operations, Input and Output data pins are not used on the CallOperation Actions. Only the Target pin is required by UML, to identify the context of (the classifier for) the CallOperationAction. In the **UCS Architectural Model**, Target pins are redundant with the Classifier already assigned to the CallOperation, so Target pins are removed.

5.6.5 Policy Example(s)

Begin Example

In Figure 28, the Participant **AtomosphericSensor** provides the Service Port **AtmosphericSensor** whose Type is the ServiceInterface **AtmosphericSensor**. This ServiceInterface Implements the Interface **AtmosphericSensorRequests** and Uses the Interface **AtmosphericSensorResponses**. Each Interface has Signals that are Parameterized by Messages. For example, the Signal **CmdAtmosphericSensor** is Parameterized by **AtmosphericSensorCmdType**.

The Request Port **VehicleInterface** is a notional implementation of the Participant role and is not normative.

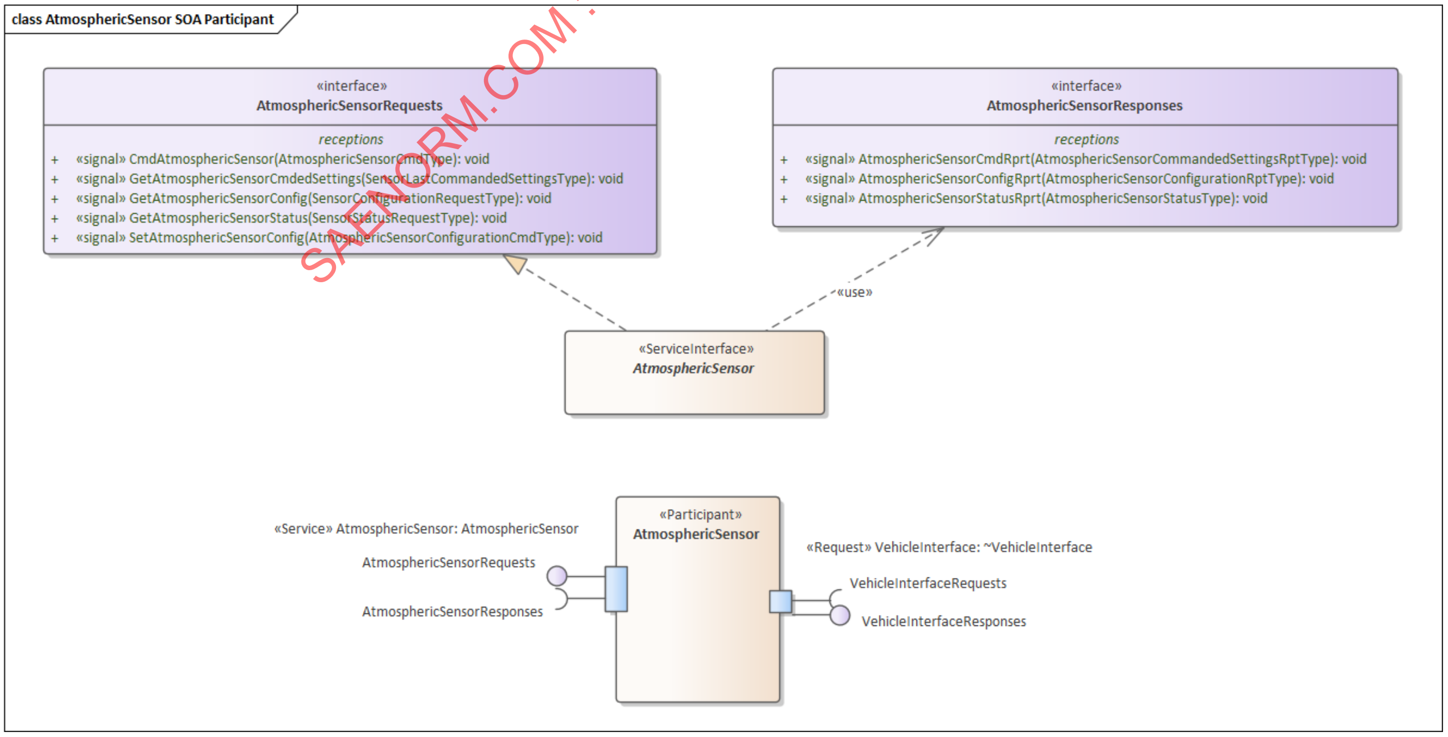

class AtmosphericSensor SOA Participant
«interface»
AtmosphericSensorRequests
receptions
+ «signal» CmdAtmosphericSensor(AtmosphericSensorCmdType): void
+ «signal» GetAtmosphericSensorCmdedSettings(SensorLastCommandedSettingsType): void
+ «signal» GetAtmosphericSensorConfig(SensorConfigurationRequestType): void
+ «signal» GetAtmosphericSensorStatus(SensorStatusRequestType): void
+ «signal» SetAtmosphericSensorConfig(AtmosphericSensorConfigurationCmdType): void
«interface»
AtmosphericSensorResponses
receptions
+ «signal» AtmosphericSensorCmdRprt(AtmosphericSensorCommandedSettingsRptType): void
+ «signal» AtmosphericSensorConfigRprt(AtmosphericSensorConfigurationRptType): void
+ «signal» AtmosphericSensorStatusRprt(AtmosphericSensorStatusType): void
«use»
«ServiceInterface»
AtmosphericSensor
«Service» AtmosphericSensor: AtmosphericSensor
AtmosphericSensorRequests
AtmosphericSensorResponses
«Participant»
AtmosphericSensor
«Request» VehicleInterface: ~VehicleInterface
VehicleInterfaceRequests
VehicleInterfaceResponses

Figure 28 - SOA participant diagram example

End Example

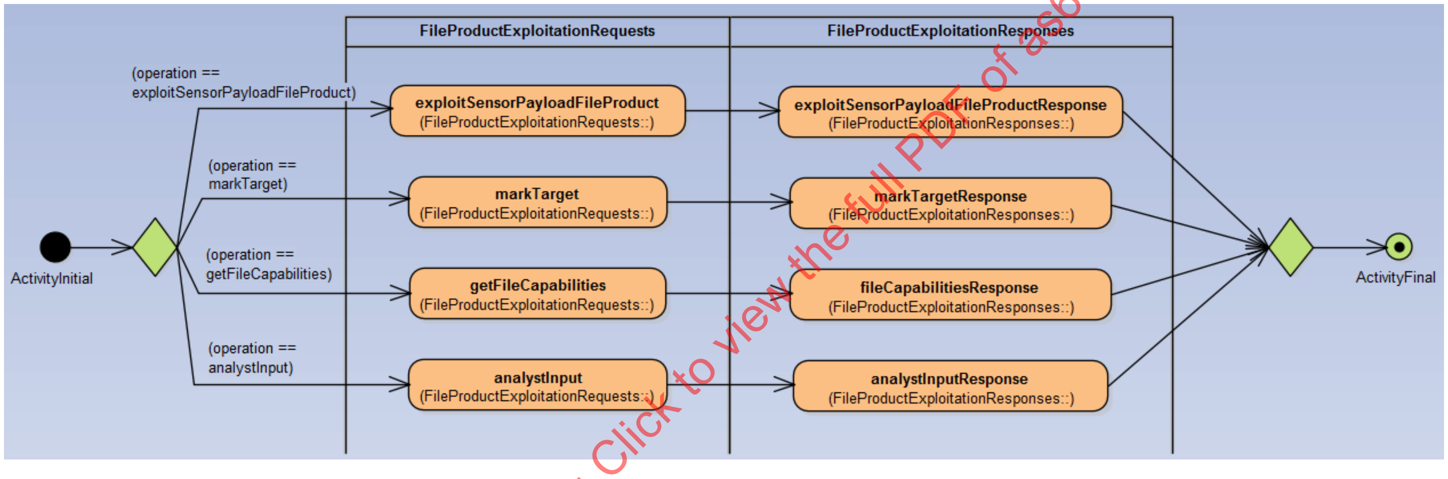
Begin Example

In Figure 29, The Protocol Activity starts with any one of the **FileProductExploitationRequest** Signals on the Service Port, followed by the appropriate response Signal.

Figure 30 demonstrates how ServiceInterface Signals can show a message exchange pattern providing for typical "request/response" flight status and Warning-Caution-Advisory (WCA), augmented by a "subscription" like message pattern using **startVehicleFlightStatus** and **stopVehicleFlightStatus** Signals to start and stop a sequence of flight status reports.

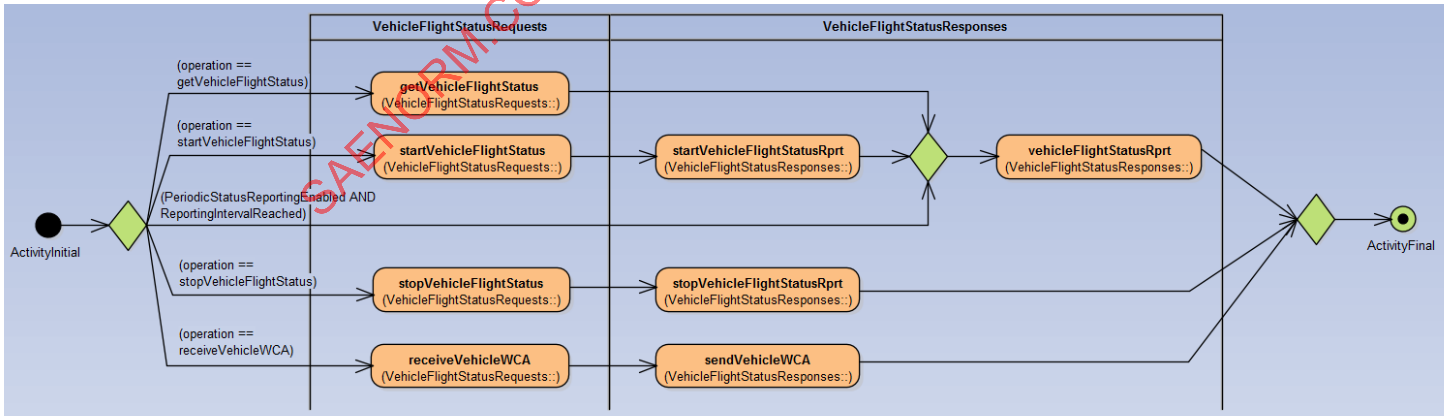
In all cases there is a single flow of control from the ActivityInitial nodes to the ActivityFinal nodes, with mutually exclusive guards on the initial DecisionNode. The second example shows that a Protocol Activity can fire either with the invocation of an operation on the Service Port, or with some other condition becoming true, as in **ReportingIntervalReached**.

Any named conditions or events such as **PeriodicStatusReportingEnabled** or **ReportingIntervalReached in** Figure 29, must be clearly specified in the Activity description.



FileProductExploitationRequests
FileProductExploitationResponses
(operation == exploitSensorPayloadFileProduct)
exploitSensorPayloadFileProduct (FileProductExploitationRequests::)
exploitSensorPayloadFileProductResponse (FileProductExploitationResponses::)
(operation == markTarget)
markTarget (FileProductExploitationRequests::)
markTargetResponse (FileProductExploitationResponses::)
ActivityInitial
(operation == getFileCapabilities)
getFileCapabilities (FileProductExploitationRequests::)
fileCapabilitiesResponse (FileProductExploitationResponses::)
ActivityFinal
(operation == analystInput)
analystInput (FileProductExploitationRequests::)
analystInputResponse (FileProductExploitationResponses::)

Figure 29 - Protocol Activity diagram example 1



VehicleFlightStatusRequests
VehicleFlightStatusResponses
(operation == getVehicleFlightStatus)
getVehicleFlightStatus (VehicleFlightStatusRequests::)
(operation == startVehicleFlightStatus)
startVehicleFlightStatus (VehicleFlightStatusRequests::)
startVehicleFlightStatusRprt (VehicleFlightStatusResponses::)
vehicleFlightStatusRprt (VehicleFlightStatusResponses::)
(PeriodicStatusReportingEnabled AND ReportingIntervalReached)
ActivityInitial
ActivityFinal
(operation == stopVehicleFlightStatus)
stopVehicleFlightStatus (VehicleFlightStatusRequests::)
stopVehicleFlightStatusRprt (VehicleFlightStatusResponses::)
(operation == receiveVehicleWCA)
receiveVehicleWCA (VehicleFlightStatusRequests::)
sendVehicleWCA (VehicleFlightStatusResponses::)

Figure 30 - Protocol Activity diagram example 2

End Example

5.7 Policy on Logical Model Package Structure

5.7.1 Policy Usage and Conformance

This policy applies to the **Logical Model** Package in the **UCS Architectural Model** starting at Revision A. The Package shall comply with the policy.

5.7.2 Policy Description and Motivation

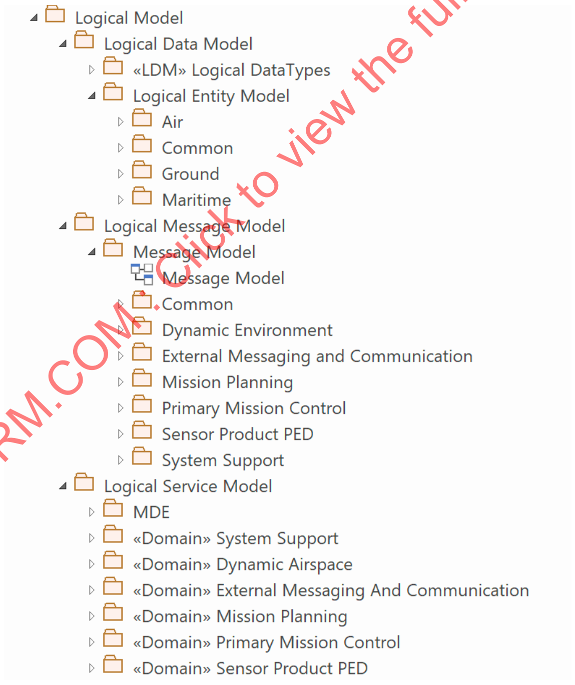
The **Logical Model** is the result of the **Refined Element Views** Package in the **Conceptual Model**. In that Package, the conceptual DataTypes in the CDM are refined into logical DataTypes. The essential difference between the **Logical Model** and **Conceptual Model** is the level of abstraction of the DataTypes. The **Logical Model** Package contains three Packages that are counterparts to the three Packages in the **Conceptual Model**. These are the **Logical Data Model**, the **Logical Message Model**, and the **Logical Service Model**. This policy describes each Package and the Dependencies between Packages.

5.7.3 Policy Constraints

None.

5.7.4 Policy

The Package structure of the **Logical Model** is shown in Figure 31.



- Logical Model
  - Logical Data Model
    - «LDM» Logical DataTypes
    - Logical Entity Model
      - Air
      - Common
      - Ground
      - Maritime
  - Logical Message Model
    - Message Model
      - Message Model
      - Common
      - Dynamic Environment
      - External Messaging and Communication
      - Mission Planning
      - Primary Mission Control
      - Sensor Product PED
      - System Support
  - Logical Service Model
    - MDE
    - «Domain» System Support
    - «Domain» Dynamic Airspace
    - «Domain» External Messaging And Communication
    - «Domain» Mission Planning
    - «Domain» Primary Mission Control
    - «Domain» Sensor Product PED

Figure 31 - Logical Model Packages

The **Logical Data Model** Package (LDM) is a model of the entities in the CDM but including their state Values at the logical level. It is the result of the **Refined Element Views** Package in the **Conceptual Message Model**. The library of LDM DataTypes is provided in the **Logical Data Types** Package, which conforms to the DMF. The **Logical Entity Types** Package follows its conceptual counterpart and contains a **Common** Package plus **Air**, **Ground**, and **Maritime** Packages. Refer to Policy 5.8

The **Logical Message Model** identifies logical messages where each logical message is a refinement of a conceptual message in the **Conceptual Message Model**. Via the **Projections** Package and **Refined Element Views** Package in the **Conceptual Message Model**, each Attribute in the **Logical Message Model** now correspondes to an Attribute in the LDM. Refer to Policy 5.9

The **Logical Service Model** Package identifies the same service ServiceInterfaces as the **Conceptual Message Model**. However, each Signal is parameterized by a message in the **Logical Message Model**. Refer to Policy 5.10.

5.7.5 Policy Example(s)

None.

5.7.6 Policy History

5.8 Policy on Logical Data Model (LDM)

5.8.1 Policy Usage and Conformance

This policy applies to the **Logical Data Model** Package in the **Logical Model** Package in the **UCS Architectural Model** starting at Revision A. The Package shall conform to the policy.

5.8.2 Policy Description and Motivation

The **Logical Data Model** Package (LDM) is a model of UCS entities and their state Values. The Package contains a **Logical Entity Model** Package whose **Datatypes** are defined in the **Logical Data Types** Package. The **Logical Entity Model** is derived from the CDM and **Refined Element Views** Package in the **Conceptual Message Model**. Whereas in the CDM, a conceptual DataType scopes the semantics of an Attribute, in the LDM, a logical DataType is able to express an Attribute's Value.

5.8.3 Policy Constraints

None.

5.8.4 Policy

As with its corresponding conceptual Package, the **Logical Entity Types** Package contains a **Common** Package plus domain-specific **Air**, **Ground**, and **Maritime** Packages. As with the CDM, these Packages are supported by the Class Stereotypes Objective, Mission, and Resource. See Policy 5.4.

The **Logical Data Types** Package applies the **DataType Profile** of the Data Model Framework. Refer to the DMF Technical Governance, Section 8 (Value Properties).

Naming Conventions in LDM

For status and selector Types, the names will be compound names, consisting of two parts, constructed as follows: **RootName_SubsetName. RootName** shall match the name of the DataType that the selector or status Type specializes. **SubsetName** shall reflect the subset of Enumerations included in the selector or status Type. When all Enumerations are included, the **SubsetName** shall be **All**. Part names shall be separated by an underscore. Part names shall be placed in the order specified. Part names shall conform to existing naming guidelines and should not themselves contain underscores.

Some example names, from the model are:

- **AlertStatusType_All**
- **MissionStatusType_All**
- **CountryCodingMethodType_ObjectCountry.**

If it is necessary to encode element properties into element names, the names will be compound names, consisting of at most five parts, constructed as follows:

RootName_RefFrameName_UnitsName_PrecisionName_RefinementType.

Properties for Refinements in the UCS message set SHOULD NOT be encoded in an element name unless two or more elements have the same RootName. Only those properties that are unique to a common RootName SHOULD be incorporated into the element name. Part names shall be separated by an underscore. Part names shall be placed in the order specified. Part names shall conform to existing naming guidelines and should not themselves contain underscores.

RefinementType is one of Capability, Constraint, Measurement, Requirement, Specification, Tolerance, or Tuple. No modifier shall precede the RefinementType part; Course_TrueNorth_Requirement is valid, but Course_TrueNorth_AngleRequirement would not be.

Some example names, from the model:

- **AirTemperature_ Capability**
- **Altitude_Barometric_Capability**
- **ElevationSensor_LineOfSightVerticalTarget_HalfAngle_Constraint**
- **GroundVelocity2D_Tuple**
- **Frequency_KiloHertz_Tolerance**
- **Position3D_PlatformXYZ_Tuple**

5.8.5 Policy Example(s)

Begin Example

Figure 32 shows the conceptual resource **UCSSystemType** on the left hand side and its logical representation, **UCSSystemType_**, on the right hand side. The logical Type is indicated by the undescore. The Features are identical except that they are Typed by conceptual and logical Types, respectively.

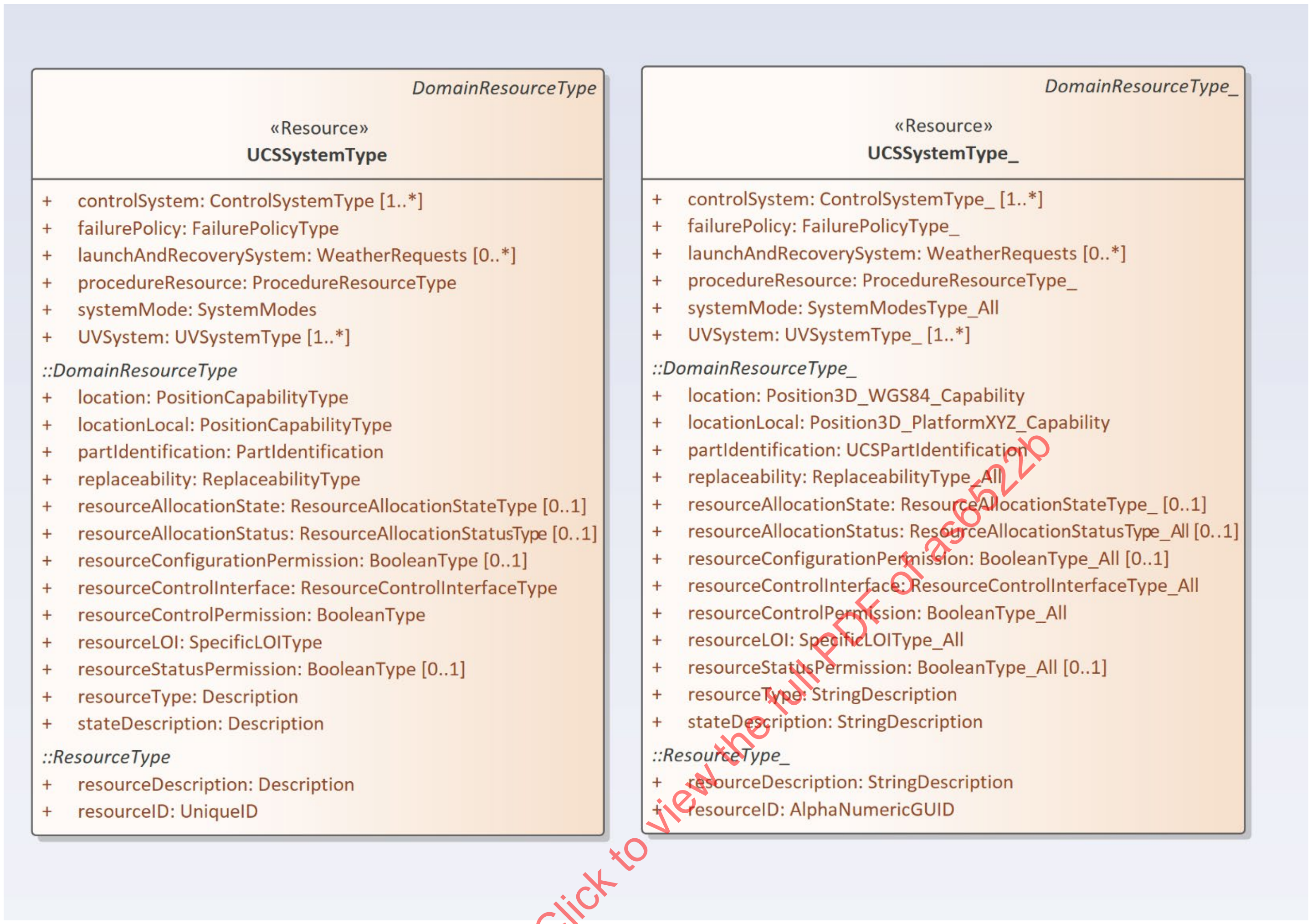

DomainResourceType
«Resource»
UCSSystemType
+ controlSystem: ControlSystemType [1..*]
+ failurePolicy: FailurePolicyType
+ launchAndRecoverySystem: WeatherRequests [0..*]
+ procedureResource: ProcedureResourceType
+ systemMode: SystemModes
+ UVSystem: UVSystemType [1..*]
::DomainResourceType
+ location: PositionCapabilityType
+ locationLocal: PositionCapabilityType
+ partIdentification: PartIdentification
+ replaceability: ReplaceabilityType
+ resourceAllocationState: ResourceAllocationStateType [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType [0..1]
+ resourceConfigurationPermission: BooleanType [0..1]
+ resourceControlInterface: ResourceControlInterfaceType
+ resourceControlPermission: BooleanType
+ resourceLOI: SpecificLOIType
+ resourceStatusPermission: BooleanType [0..1]
+ resourceType: Description
+ stateDescription: Description
::ResourceType
+ resourceDescription: Description
+ resourceID: UniqueID
DomainResourceType_
«Resource»
UCSSystemType_
+ controlSystem: ControlSystemType_ [1..*]
+ failurePolicy: FailurePolicyType_
+ launchAndRecoverySystem: WeatherRequests [0..*]
+ procedureResource: ProcedureResourceType_
+ systemMode: SystemModesType_All
+ UVSystem: UVSystemType_ [1..*]
::DomainResourceType_
+ location: Position3D_WGS84_Capability
+ locationLocal: Position3D_PlatformXYZ_Capability
+ partIdentification: UCSPartIdentification
+ replaceability: ReplaceabilityType_All
+ resourceAllocationState: ResourceAllocationStateType_ [0..1]
+ resourceAllocationStatus: ResourceAllocationStatusType_All [0..1]
+ resourceConfigurationPermission: BooleanType_All [0..1]
+ resourceControlInterface: ResourceControlInterfaceType_All
+ resourceControlPermission: BooleanType_All
+ resourceLOI: SpecificLOIType_All
+ resourceStatusPermission: BooleanType_All [0..1]
+ resourceType: StringDescription
+ stateDescription: StringDescription
::ResourceType_
+ resourceDescription: StringDescription
+ resourceID: AlphaNumericGUID

Figure 32 - Conceptual and Logical UCSSystemType

End Example

5.8.6 Policy History

New at Revision A.

5.9 Policy on Logical Message Model

This policy applies to the **Logical Message Model** Package in the **Logical Model** Package in the **UCS Architectural Model**. The Package shall conform to the policy.

5.9.1 Policy Description and Motivation

The **Logical Message Model** Package is a library of logical messages where each message is a logical view of the data model that exposes content of interest to one or more services in the **Logical Service Model**. Each Attribute in the **Logical Message Model** is a Projection of an Attribute in the LDM via the **Projections** Package in the **Conceptual Message Model**.

5.9.2 Policy Constraints

None.

5.9.3 Policy

The **Logical Message Model** contains one Package, the **Message Model**, whose Package structure follows that of the **Message Model** in the **Conceptual Data Model**. In the Logical Model, the DataTypes are at the logical level of abstraction.

5.9.4 Policy Example(s)

None.

5.9.5 Policy History

5.10 Policy of Logical Service Model

5.10.1 Policy Usage and Conformance

This policy applies to the **Logical Service Model** Package of the **Logical Model** Package. This Package will conform to the policy.

5.10.2 Policy Description and Motivation

The **Logical Service Model** Package is a logical refinement of the **Conceptual Service Model** Package. It uses logical messages as Signal Parameters.

5.10.3 Policy Constraints

None.

5.10.4 Policy

In the Logical Service Model, the structure and Applied Profiles are identical to Service Model. See Policy 5.6. The only difference is the level of abstration of the DataTypes.

5.10.5 Policy Example(s)

5.10.6 Policy History

6. NON-FUNCTIONAL PROPERTY POLICIES

The current **Non-functional Property Model** is a legacy work product developed for the OSD UCS Architecture, the precursor to the SAE UCS Architecture Model, AS6518. This was developed for the Air Domain content only, and has not been maintained in AS6518 as new content has been added in Revision A.

This Package is therefore provided as a reference only, and caution should be exercised in its usage.

6.1 Policy on Non-Functional Property Library

6.1.1 Policy Usage and Conformance

This policy applies to the **Non-Functional Properties** Package of the **UCS Architectural Model** starting at Revision A.

6.1.2 Policy Description and Motivation

The **Non-Functional Properties Properties** Package provides libraries of non-functional properties based on the Class Stereotype **UCSNfpProperty**. This library is a resource available to users to bind the UCS Architecture Model to a system model by assigning non-functional properties to model Features.

6.1.3 Policy Constraints

None.

6.1.4 Policy

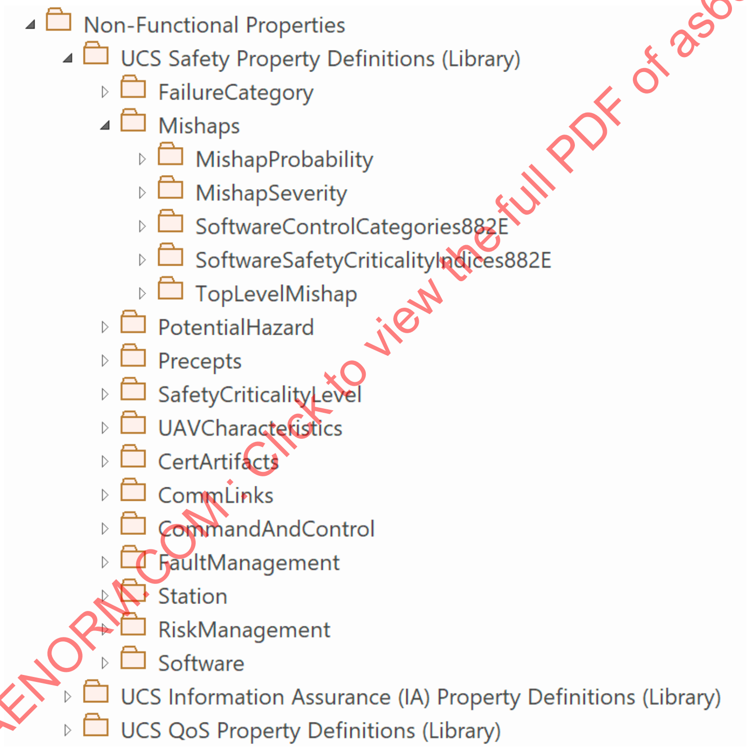
The **UCS Architectural Model** contains property definition libraries for quality of service, safety, and information assurance. These libraries consist of enumerations and **UCSNfpProperty** Elements. These may be aggregated into property groups, service level agreements, and service contracts.

The libraries are organized by Namespaces such that Namespaces shall only contain packages, and at the lowest level a UCS non-functional property. Groups of NFPs, SLAs, and service contracts may be constructed for specific implementations using UCSNfpProperty elements provided in the **UCS Architectural Model** library.

6.1.5 Policy Example(s)

Begin Example

Figure 33 shows the context for the **TopLevelMishap** Package in the **Non-Functional Properties** Package. The UCSNfpProperty **TopLevelMishapType** is shown in Figure 34. The linked documents provide or reference the definitions of concepts and literals.



```
◢ Non-Functional Properties
    ◢ UCS Safety Property Definitions (Library)
        ▷ FailureCategory
        ◢ Mishaps
            ▷ MishapProbability
            ▷ MishapSeverity
            ▷ SoftwareControlCategories882E
            ▷ SoftwareSafetyCriticalityIndices882E
            ▷ TopLevelMishap
        ▷ PotentialHazard
        ▷ Precepts
        ▷ SafetyCriticalityLevel
        ▷ UAVCharacteristics
        ▷ CertArtifacts
        ▷ CommLinks
        ▷ CommandAndControl
        ▷ FaultManagement
        ▷ Station
        ▷ RiskManagement
        ▷ Software
    ▷ UCS Information Assurance (IA) Property Definitions (Library)
    ▷ UCS QoS Property Definitions (Library)
```

Figure 33 - Context of TopLevelMishap Package

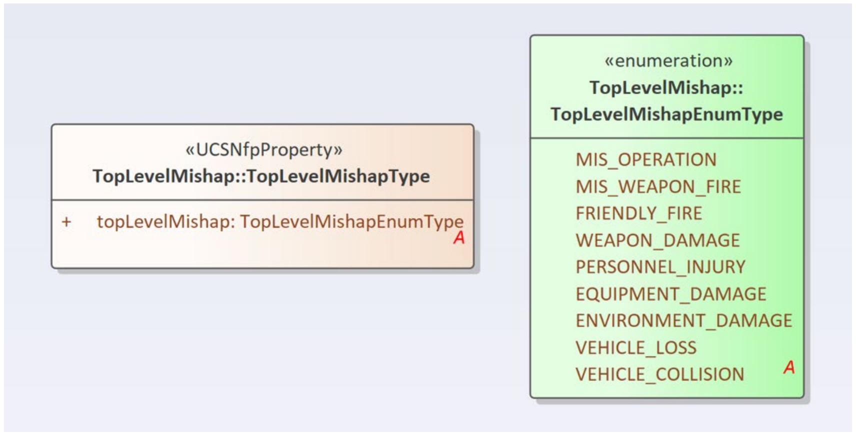


Figure 34 - TopLevelMishapType

End Example

6.1.6 Policy History

New at Revision A.

7. SYSTEM USE CASE MODEL POLICIES

The current **System Use Case Model** is a legacy work product based on the original Use Cases developed for the OSD UCS Architecture, the precursor to the SAE UCS Architecture Model, AS6518. These Use Cases apply to the Air Domain content only, and have not been maintained in AS6518 as new content has been added in Revision A.

This Package is therefore provided as a reference only, and caution should be exercised in its usage.

7.1 Policy on Use Case Diagrams

7.1.1 Policy Usage and Conformance

This policy applies to all Use Case Diagrams in the **System Use Case Model** Package of the UCS Architectural Model Package. All such Diagrams shall conform to this policy.

7.1.2 Policy Description and Motivation

Standard UML is used in the development of Use Case Diagrams. The Use Cases are either Level 0 (L0), Level 1 (L1), Level (L2) or Level 3 (L3). L0 applies to the entire UCS system. L1 applies to the service domains in the service model. L2 applies to the service subdomains in the service model. L3 applies to individual services if required.

Use Cases are a means for describing the reference service choreography of the UCS. Typically, they are used to identify the required service Signals and Messages. The required behavior of the reference system is specified by one or more Use Cases, which are defined according to the needs of Actors. A use case is a kind of UCS behavioral Classifier that represents a declaration of an offered behavior. Each Use Case specifies some behavior, possibly including variants, which the subject can perform in Collaboration with one or more Actors. Use Cases define the offered behavior of the subject without reference to its internal structure.

Use case diagrams are in the **System Use Case Model** at Package levels L0 through L3.

7.1.3 Policy Constraints

None.