June 2020

Simplified Electronic Notation for Sensor Reporting (SENSR) 1.0(beta)

Version 1.0(beta)

OMG Document Number:	dtc/2020-07-02 (this document)
Normative Machine Readable Files:	https://www.omg.org/spec/SENSR/20191204/SENSR.xmi

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification. Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means–graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems–without permission of the copyright owner.

Contents

Preface

About the Object Management Group

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia. OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML®(Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets. More information on the OMG is available at https://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at: https://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt	Standard body text
Helvetica/Arial - 10 pt Bold	OMG Interface Definition Language (OMG IDL) and syntax elements
`Courier – 10 pt Bold`	Programming language elements
Helvetica/Arial - 10 pt	Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to: https://www.omg.org/report_issue.htm.

1 Scope

Currently, manufacturers of hardware sensors producing data rely on arcane data sheets to describe the format of data supplied by their sensors. Each manufacturer has defined their own data sheet format, and each consumer of the data needs to interpret the data sheet for implementation in their own system. Errors and ambiguities can, and are likely to, occur in these situations.

In response to the experiences of the members of the Industrial Internet Consortium (IIC) [4], particularly those involved with the implementation of the IIC Testbeds initiative, this submission describes a metamodel for describing the form of serialized data streams emitted by sensors, and describing how that data should be interpreted by a client to derive the intended meaning of the data. The intent is that a manufacturer's sensor products can be characterized by a model expressed in the proposed metamodel. This submission does not concern itself with the transport, networking, or wire protocol for transmitting the data from the sensor to the client.

This submission specifies a metamodel appropriate for modeling the syntax of the data streamed, as well as a mechanism for authoring a guideline for interpretation of that data. The model of a sensor's data will then be provided by the manufacturer as a document called an Electronic Data Sheet (EDS). The EDS will be a machine consumable document such that it can be generated by the manufacturer and parsed by the consumer. The method for delivering an EDS document is outside the scope of this submission.

The EDS is analogous to the current hardware data sheets offered by manufacturers, but improves on the current state of the art by being highly structured, appropriate for machine consumption, and in a standardized form. An illustration of how the EDS may be used is as follows:

- Equipment is required to offer sensor data to one or more unknown consumers.
- A consumer wishing to receive the data will access the EDS for the sensor product. The delivery of the EDS may occur at any time prior to configuration of the consumer.
- Guided by the EDS, the consumer is adapted to parse the data provided by the sensor. This configuration may occur at any time prior to communication with the sensor.
- The consumers and the hardware will communicate over a pre-determined protocol, such as Bluetooth, mDNS + TCP/IP, or other to be determined channels.
- The consumer begins to receive data from the sending hardware, and can interpret the data in the way that is intended.

The intent is that manufacturers can use the metamodel to specify an EDS to provide a precise model of the data their equipment is producing, and consumers can faithfully refer to a common standard to understand the EDS, and information needed to interpret the data stream. This knowledge can be used to pre-configure a sensor client prior to deployment. There is also the possibility of creating a dynamic configuration engine that parses EDS documents and produces appropriate parsers and interpreters of data streams at run time. This allows manufacturers to provide a precise description of the output of their hardware, confident that future consumers of the data produced by that hardware can properly interpret the data stream.

An example of a bit stream that should be able to be defined is shown in Listing 1.1 at the end of this section. This listing was provided by Bosch as an example of an existing data stream descriptor for an existing sensor, and appears in Section 6.8.1 under Evaluation Criteria of the SENSR RFP[SENSR RFP].

The scope of the SENSR specification includes:

- Provide a metamodel that enables the unambiguous definitions of data types
- Describe an appropriate format to express EDS documents, where the EDS is a representation of a use of the metamodel

- Provide a mechanism to ensure unique identifiers for each encoding organization offering EDS documents.

The SENSR specification is not intended to:

- Define a new communication mechanism
- Restrict the kinds of hardware sensors that can provide data
- Restrict the kinds of consumers that can interpret data

```
[FileInfo]
FileName=Temperature_Sensor_EDS.eds
FileVersion=1
FileRevision=0
Description=Temperature Sensor
CreationTime=08:00AM
CreationDate=21-08-2018
CreatedBy=A. Cordes, BOSCH
ModificationTime=08:00AM
ModificationDate=21-08-2018
ModifiedBy=A. Cordes, BOSCH

[DeviceInfo]
ManufacturerID=0x02A6

[Payload_Value_1]
Length=16
Offset=0x00
Scale=1
Name=Sensor ID
Info=0x3815
Unit=n

[Payload_Value_2]
Length=8
Offset=0x00
Scale=1
Name=Counter
Unit=n

[Payload_Value_3]
Length=8
Offset=0x00
Scale=1
Name=Unused
Unit=n

[Payload_Value_4]
Length=2
Offset=0x00
Scale=1
Name=Telegram Version
Unit=n

[Payload_Value_5]
```

```
Length=1
Offset=0x00
Scale=1
Name=Unused
Info=Value is constant 0
Unit=n

[Payload_Value_6]
Length=1
Offset=0x00
Scale=1
Name=Payload Encryption
Unit=n

[Payload_Value_7]
Length=1
Offset=0x00
Scale=1
Name=Battery Status
Unit=n

[Payload_Value_8]
Length=16
Offset=0
Scale=1
Name=Temperature
format=signend8.8
Unit=C

[Payload_Value_9]
Length=8
Offset=0
Scale=1
Name=Temperature_max1h
format=signed8
Unit=C

[Payload_Value_10]
Length=8
Offset=0
Scale=1
Name=Temperature_min1h
format=signed8
Unit=C

[Payload_Value_11]
Length=8
Offset=0
Scale=1
Name=Temperature_max24h
format=signed8
Unit=C

[Payload_Value_12]
```

```
Length=8
Offset=0
Scale=1
Name=Temperature_min24h
format=signed8
Unit=C
```

Listing 1.1: Example Bitstream

2 Conformance

2.1 Basic Conformance

All conformant products must comply with these conformance points:

- *Implement the Base package*. The Base package defines each abstract base class for the Syntax, Semantics, and Unit class families, the DataElement class, and the DataSheet class. It provides the initial set of relationships.
- *Implement the Syntax package*. The Syntax package defines the bit layout in a data stream.
- *Implement the Semantics package*. The Semantics package defines the basic data kinds that consumers can expect to encounter.
- *Implement the Units package*. The Units package defines a comprehensive way to describe physical units in a measurement-system-independent manner, and then impose a measurement-system on the dimensioned descriptors.

2.2 Extended Conformance

Products compliant with Extended Conformance must comply with all points of Basic Conformance, and in addition provide a suitably useful data types library that offers either US Customary or Metric compliant units, a suite of syntax and semantic types to handle the most common types such as a Boolean (single bit) type, and both 8 and 16 bit integer types.

3 References

3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[MOF] *Meta Object Facility (MOF) Core*. Version 2.5.1. Object Management Group. URL: https://www.omg.org/spec/MOF/2.5.1.

[SENSR RFP] Object Management Group. *Simple Electronic Notation for Sensor Reporting (SENSR) RFP*. URL: https://www.omg.org/cgi-bin/doc?mantis/2018-09-10.

[UML] *Unified Modeling Language*. Version 2.5.1. Object Management Group. URL: https://www.omg.org/spec/UML/2.5.1.

[XMI] *XML Metadata Interchange (XMI)*. Version 2.5.1. Object Management Group. URL: https://www.omg.org/spec/XMI/2.5.1.

[XML Schema] *XML Schema Part 2: Datatypes*. World Wide Web Consortium. 2004. URL: https://www.w3.org/TR/xmlschema-2.

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Complete MOF (CMOF)
The CMOF, or Complete MOF, Model is the model used to specify other metamodels such as UML2.

Electronic Data Sheet (EDS)
An Electronic Data Sheet is an electronically transmitted document that describes a component, generally a hardware component in electrical engineering. That component can be as low level as a single analog device (resistor, diode, capacitor), or a complex device such as an environmental sensor. The data sheet describes how to integrate that device into a larger system.

Essential MOF (EMOF)
Essential MOF is the subset of MOF that most closely corresponds to the facilities found in object-oriented programming languages and in XML. It provides a straightforward framework for mapping MOF models to implementations such as JMI and XMI for simple metamodels. A primary goal of EMOF is to allow simple metamodels to be defined using simple concepts while supporting extensions (by the usual class extension mechanism in MOF) for more sophisticated metamodeling using CMOF.

Industrial Internet Consortium (IIC)
"The Industrial Internet Consortium was founded in March 2014 to bring together the organizations and technologies necessary to accelerate the growth of the industrial internet by identifying, assembling, testing and promoting best practices. Members work collaboratively to speed the commercial use of advanced technologies. Membership includes small and large technology innovators, vertical market leaders, researchers, universities and government organizations." https://www.iiconsortium.org/

Metamodel
A metamodel is a model that acts as the schema for a family of models.

Meta-Object Facility (MOF)
The Meta Object Facility (MOF), an OMG specification, provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata-driven systems. Examples of systems that use MOF include modeling and development tools, data warehouse systems, metadata repositories etc.

Model
A model is a formal specification of the function, structure and/or behavior of an application or system.

XML Metadata Interchange (XMI)
XMI is a widely used interchange format for sharing objects using XML. XMI is a comprehensive solution that build on sharing data with XML. XMI is applicable to a wide variety of objects: analysis (UML), software (Java, C++), components (EJB, IDL, CORBA Component Model), and databases (CWM).

5 Symbols

The following symbols and/or abbreviations are used throughout this specification.

None.

6 Additional Information

(informative)

6.1 How to read this Specification

This **specification** presents a metamodel for describing Electronic Data Sheets suitable for use by manufacturers of devices that sense and report on observed phenomena. Clauses 1 to 6 provide compliance rules, terms definitions and reference information. Clause 7 provides the description of the metamodel for SENSR. Clause 8 provides an informative example of how to use SENSR to build a library of Units. Clause 9 provides informative examples of how to define EDSs in SENSR, with equivalent XML representations. Clause 10 provides an informative example of a PSM for a Bluetooth LE implementation.

All clauses of this document are normative unless explicitly marked "(informative)". The marking "(informative)" of a particular clause applies also to all contained sub-clauses of that clause.

6.2 Acknowledgments

- Axel Cordes and team members of Bosch Software Innovations reviewed this submission
- Peter Denno, NIST, provided review and feedback
- Jason McC. Smith, Elemental Reasoning, prepared the final submission

7 Core Packages

7.1 Base

SENSR's Base package defines the fundamental concepts of SENSR. The DataSheet is composed of DataElements, and DataElements have three parts to their definition: Syntax, Semantics, and optionally Units.

Users of this metamodel are expected to create their own types from one of the four provided subtypes of DataElement, depending on the representation in the sensor's data.

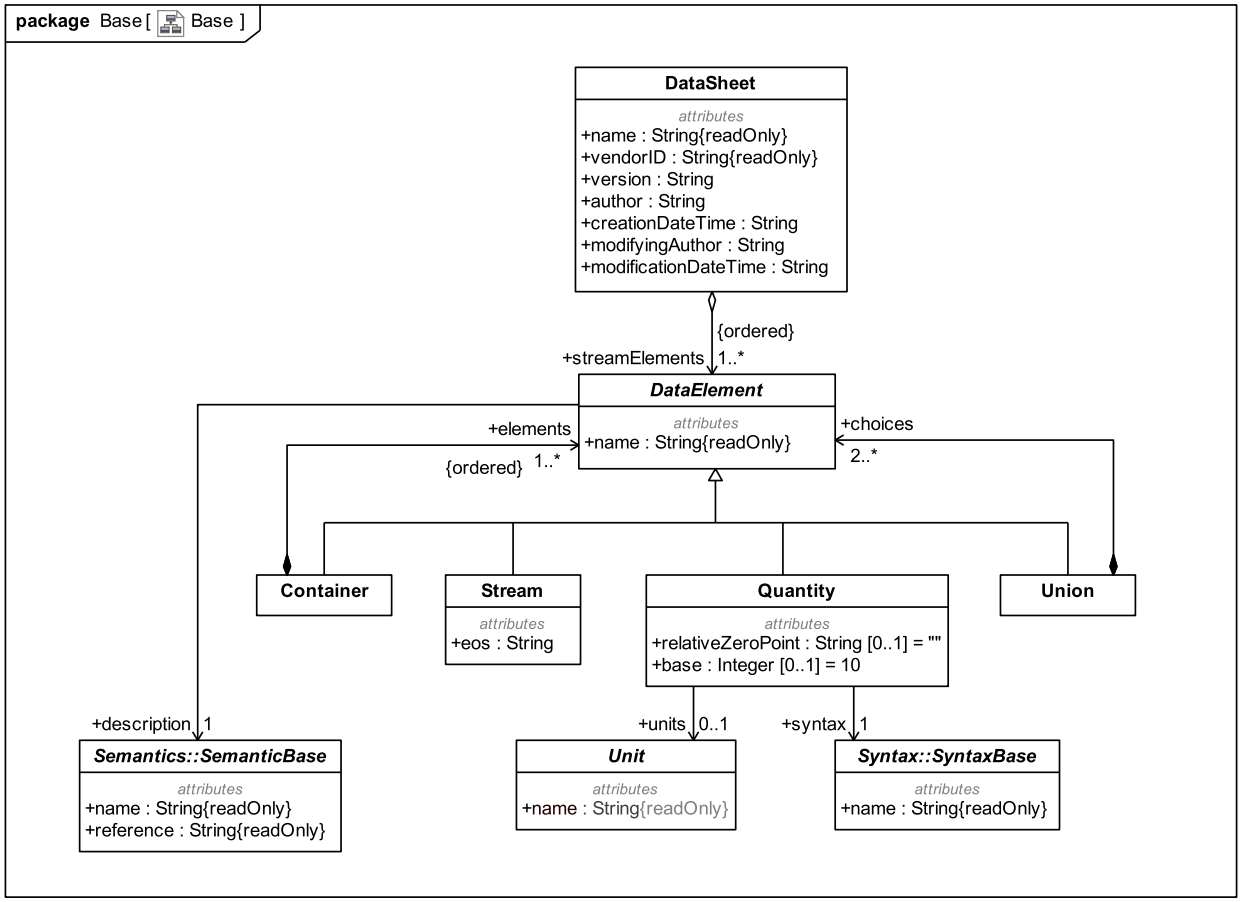

Figure 7.1: Base Package Class Diagram

7.1.1 Container

A Container wraps an ordered collection of DataElements into a single entity. This is useful in cases where, for example, a triplet of Distance measurements comprise a CartesianCoordinate, or it is necessary to break down a Numeric type into the individual components such as Sign, Exponent, and Mantissa.

Generalizations

DataElement

Associations (Required)

elements : DataElement [1..*] {ordered}
The elements attribute points to an ordered list of DataElement instances, such that the container acts as a wrapper providing a syntactic ordering of subelements. The subelements can be any DataElement subtype, another Container, a Quantity, a Union, or even a Stream. In the last case, the semantics are such that when the Stream's data is ended, however that is achieved, the next element in the Container's sequence is expected.

7.1.2 DataElement (Abstract)

A DataElement models a unit of data of the DataSheet's streamElements, such that a consumer of the DataSheet can understand that portion of the data stream. A DataElement contains physical layout (Syntax), a descriptor of the kind of expected data (Semantics), and an optional Units that explains how to interpret the data as an observation of a physical phenomenon.

Attributes (Required)

name : String
The name for the DataElement, this is vendor defined

Associations (Required)

description : SemanticBase [1]
The instance pointed to by the description association is of type SemanticBase. This class family provides a DataElement with a way of describing the kind of data it will define, thus guiding interpretation.

7.1.3 DataSheet

DataSheet is the top level modeling element in SENSR. It is the electronic equivalent of a traditional manufacturer's data sheet for electronic devices. It provides guidance for implementors on how to interpret a data stream, and is composed of DataElement instances.

Attributes (Required)

author : String
Initial author of the Data Sheet.

creationDateTime : String
Date and timestamp of the initial creation of this DataSheet. Format is ISO 8601 compliant, and assumes UTC timezone: YYYYMMDDThhmmZ.

modificationDateTime : String
Date and timestamp of the last modification of this DataSheet. Format is ISO 8601 compliant, and assumes UTC timezone: YYYYMMDDThhmmZ.

modifyingAuthor : String
Author who last modified this DataSheet.

name : String
The name of the DataSheet, provided by the vendor. Each vendor is likely to have their own naming convention, such as the make of the sensor whose output stream is being modeled.

vendorID : String
The identifier for a specific vendor. This vendorID is unique to a vendor, and requires registration with a central organization.

version : String
Version is defined in x.y.z format, where x is major version, y is revision, and z is patch, and each is a monotonically increasing integer starting at 0, and reset when the number to the left is incremented.

Associations (Required)

streamElements : DataElement [1..*] {ordered}
An ordered sequence of DataElements that make up the bit stream.

7.1.4 Quantity

A Quantity is a 'simple' value, such as a number, a string, a flag, or a bitmask. It has a fixed data size (in bits), a known semantics, and optionally a units signifier.

Generalizations

DataElement

Attributes (Optional)

base : Integer [0..1]
The numeric base of the number being represented. Defaults to 10 unless otherwise specified.

relativeZeroPoint : String [0..1]
If a value for relativeZeroPoint is provided, it is to be interpreted as establishing this Quantity as representing a relative measure. For instance, assume a home thermostat is reporting temperature in degrees Celsius as a fixed point number using a relatively small number of bits. Since it is unlikely the thermostat will need to report values below 15C, or above 35C, it sets a relative zero point of 25, and is able to use its bit space to more precisely measure the +/- 10C it is most concerned with. This value is provided as a String, to allow it to be independent of the syntax, description, and units associations.

Associations (Required)

syntax : SyntaxBase [1]
An association to an instance of SyntaxBase, this defines the layout of the bits for this Quantity.

Associations (Optional)

units : Unit [0..1]
An optional association, units provide dimensionality to a measure.

7.1.5 Stream

A Stream has no particular end to its data. Unlike a Quantity, which has a specific bit layout, or a Container, which has a specific ordered list of DataElements, or a Union, which has a fixed size with alternate semantics, a Stream has no predefined length.

Generalizations

DataElement

Attributes (Required)

eos : String
End of Stream signifier. For simple value streams, a String value should suffice. For more complex streams, such as video or audio, it may be better to use the reference attribute to point to an external specification, and leave this attribute blank.

7.1.6 Union

A Union, named after a similar data structure in the C programming language family, offers alternative semantics for the same syntax field. Two or more DataElements are pointed to with the understanding that only one DataElement can occupy the relevant bits in the data stream. As such, the DataElements in the collection of choices usually have the same syntactic length. However, if the Union points to a VariableSize instance for its Syntax, then at least one of the DataElements pointed to by the Union will have a unique Syntax element. Often, which semantics are used to interpret the bit field is based on a conditional expression or value of a flag in the stream.

Generalizations

DataElement

Associations (Required)

choices : DataElement [2..*]
Two or more choices to be selected from. Each choice must have the same Syntax length.

7.2 Syntax

The Syntax package lets authors of DataSheets define the 'physical' layout of bits at the most basic level. This version of the specification provides the FixedSize class, a provision that satisfies most needs. By providing an abstract base class, however, we leave open the opportunity for future possibilities.

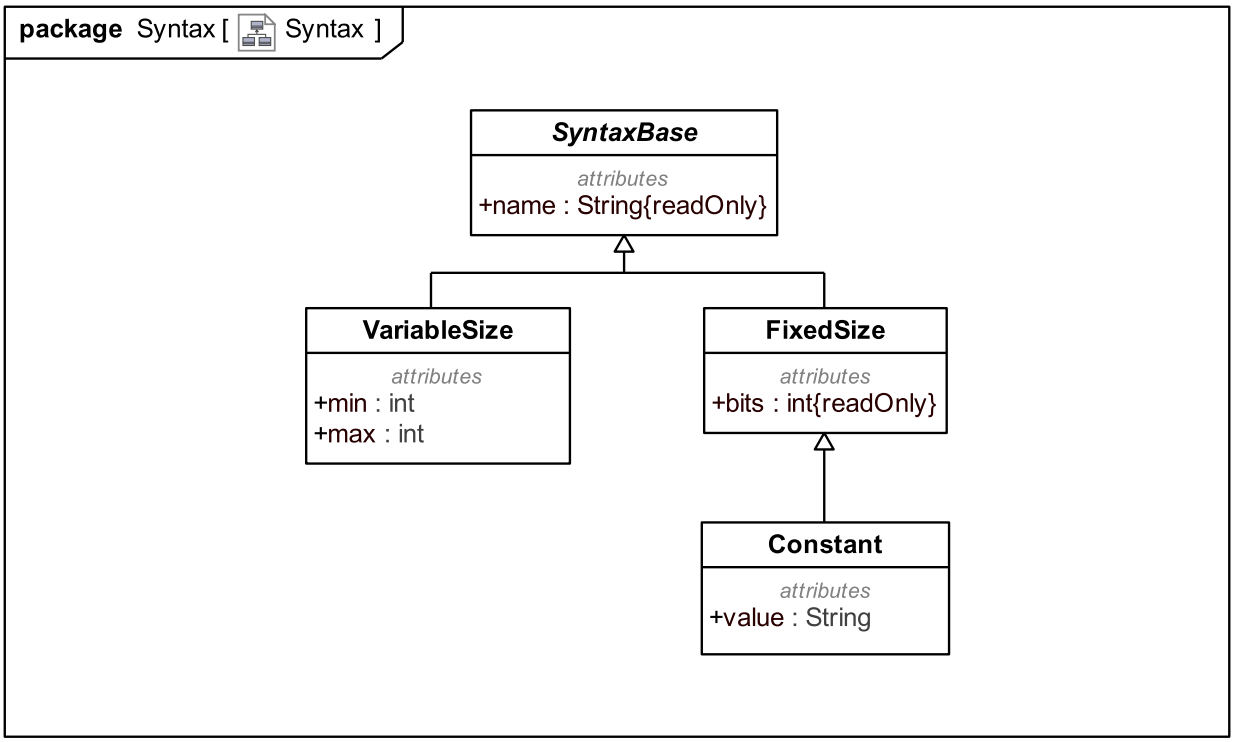

Figure 7.2: Syntax Package Class Diagram

7.2.1 Constant

A Constant data element has not only a fixed size, but a fixed value. The value String attribute gives the value in the expected format indicated by the Semantics that accompany the data element.

Generalizations

FixedSize

Attributes (Required)

value : String
The value of the Constant held by the data element.

7.2.2 FixedSize

The FixedSize class is expected to be the most commonly used Syntax class. It is used to define the number of bits that a Quantity DataElement will take up in the datastream.

Generalizations

SyntaxBase

Attributes (Required)

bits : int
A simple count of the bits used for a Quantity.

7.2.3 SyntaxBase (Abstract)

SyntaxBase provides a common interface for describing the physical layout of bits within a bitstream.

Attributes (Required)

name : String
The name of the Syntax being defined.

7.2.4 VariableSize

A VariableSize Syntax informs the DataElement that the number of bits used by it may change due to other considerations. This is useful with a Union DataElement subtype.

Generalizations

SyntaxBase

Attributes (Required)

max : int
Maximum number of bits that will be occupied by this Syntax element.
min : int
Smallest number of bits that will be occupied by this Syntax element.

7.3 Semantics

The Semantic package defines an extensible system for describing the kind of data being expressed by a DataElement. The included descriptors range from the usual text and numeric data, to more complicated audio-visual data kinds. The Semantic classes do not describe the physical layout of the data (which is handled by the Syntax package), or how to interpret the data with respect to the physical world (which is handled by the Units package). The reference attribute in the DataElement class is used to point to an authoritative external source for guidance, where applicable. It is not embedded here, because the external source will necessarily depend on the syntax layout and, potentially, the physical units.

7.3.1 Audio

Describes an audio stream.

Generalizations

SemanticBase

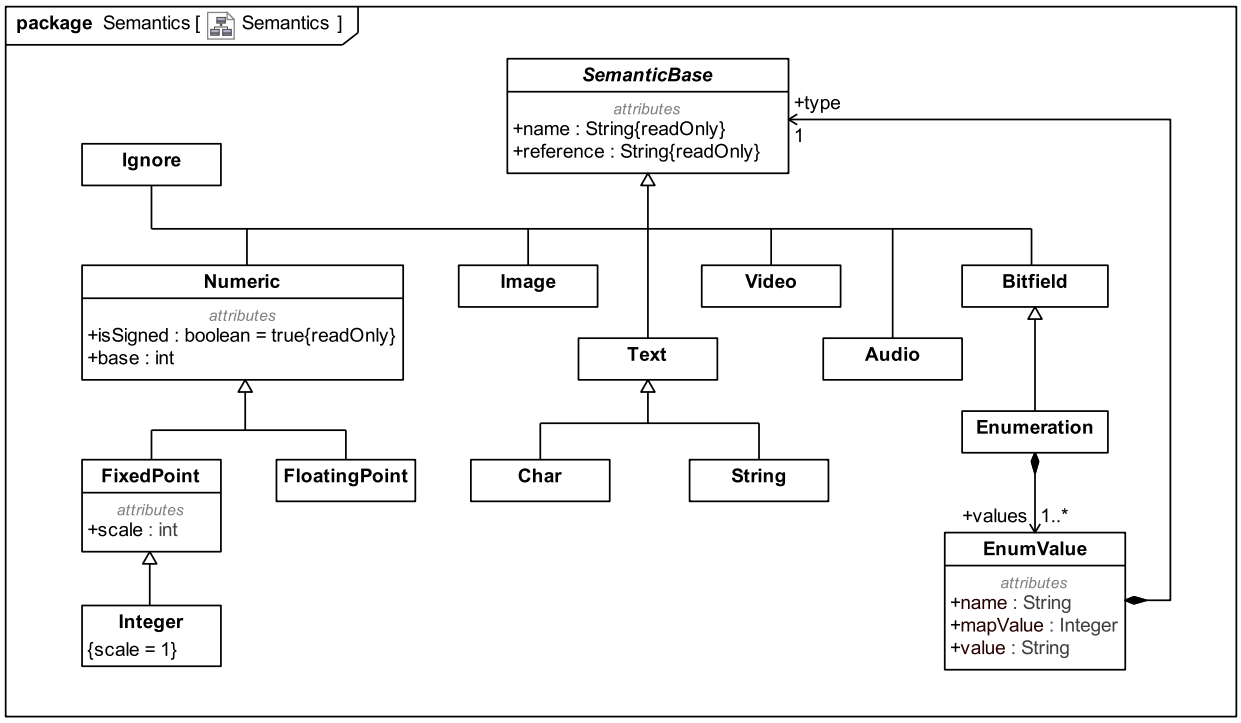

Figure 7.3: Semantics Package Class Diagram

7.3.2 Bitfield

A Bitfield describes a fixed length vector of single bits. This is commonly used to collect boolean flags into a common space. (Note that the field can be a single bit long, in which case it is just a flag.)

Generalizations

SemanticBase

7.3.3 Char

Describes a single character. Encoding is handled by an appropriate reference.

Generalizations

Text

7.3.4 Enumeration

Describes an enumeration, a fixed set of values that are the only valid values for the DataElement.

Generalizations

Bitfield

Associations (Required)

values : EnumValue [1..*]

A set of EnumValue objects that describe the allowed values for this Enumeration.

7.3.5 EnumValue

A single value for an Enumeration kind. It has a name to refer to it by, and a value. The value is an Integer, interpreted from the Bitfield that it specializes as big-endian base 2 values. Note that this is not to specify that the data must be big-endian or base 2, but that for the purposes of mapping the bits to the enumerated value, there will be a specific bit pattern that is to be interpreted in that matter.

Attributes (Required)

mapValue : Integer
: The Integer value that corresponds to the bit pattern of the Enumeration to map to this EnumValue. The bit pattern in the base Bitfield will be interpreted as big-endian base 2 to form this value.

name : String
: The name of this particular enumeration value.

value : String
: The value of the EnumValue. This is represented as a String, which is to be interpreted as the type specified by the type attribute.

Associations (Required)

type : SemanticBase [1]
: Guidance on how to interpret the value attribute. If, for example, this points to a String instance, then the String value of the value attribute is to be taken literally. If this attribute points to an Integer instance, then "1" would be the integer 1, "20" would be interpreted as the integer 20, and so on.

7.3.6 FixedPoint

A fixed point number, having a scale value that converts the integer to a fixed decimal position.

Generalizations

Numeric

Attributes (Required)

scale : int
: The scaling factor by which to divide the integer.

7.3.7 FloatingPoint

A floating point number. Floating point representation can be quite complicated. Typically, FloatingPoint objects will be used in a DataElement that points to an established reference such as IEEE 754.

Generalizations

Numeric

7.3.8 Ignore

Some sections of a bit stream are unused, and are to be ignored. Use this Semantic kind to indicate that a consumer should simply skip these bits.

Generalizations

SemanticBase

7.3.9 Image

Describes an image. Except for simple bitmaps, this is likely going to be pointing to an established specification using DataElement::reference.

Generalizations

SemanticBase

7.3.10 Integer

An integer. Equivalent to a fixed point number with a scaling factor of 1.

Generalizations

FixedPoint

Constraints

scale : [scale = 1]
An integer is a fixed point with a scaling factor of 1.

7.3.11 Numeric

Any single-dimensional numeric value.

Generalizations

SemanticBase

Attributes (Required)

base : int
Base of the numeric type.

isSigned : boolean
Flag to indicate if the numeric type is signed. Default is true.

7.3.12 SemanticBase (Abstract)

SemanticBase is the base abstract class for all semantic classes. It supplies only a name attribute for the kind to be described.

Attributes (Required)

name : String
The name of the semantic kind being described.

reference : String
URI to find reference material on the DataElement. Examples could be a link to the IEEE 745 floating point specification, UTF-8 for strings, etc.

7.3.13 String

A string, a collection of characters. Encoding is handled by an appropriate reference.

Generalizations

Text

7.3.14 Text

Any kind of textual data.

Generalizations

SemanticBase

7.3.15 Video

A video stream. Almost certainly this will use DataElement::reference to point to an established specification.

Generalizations

SemanticBase

7.4 Units

Units provide a dimensionality to a number. They give a rigorous interpretation guide for measurements of physical phenomena, as well as provide a concrete representation space such as SI, US Customary, or Imperial measurements.

Users of SENSR are given four options for defining a Unit:

- Simple, an abstract base class for units that have a single dimensionality such as SI units or Imperial units
- Scaled, which scales another Unit by a fixed amount, such as 1000 grams in a kilogram, or 12 inches in a foot
- Multiply which combines dissimilar Units into a complex dimensioned Unit
- Exponent, which provides a compact way of expressing higher dimensionality of a single Unit.

Complex units such as those for measuring velocity, density, and force, can be built using the above mechanisms.

Conversions from one unit system to another can be accomplished by judicious use of the Scaled class.

7.4.1 Exponent

The Exponent class raises a Unit to a given power. The power is floating point, to accommodate fractal cases, and is signed to provide a mechanism for division.

Generalizations

Unit

Attributes (Required)

power : float
The exponent to raise the Unit to. It is a signed number, so it can be used naturally for division via a negative exponent. It is a floating point to handle fractal dimensionalities.

Associations (Required)

base : Unit [1]
The base attribute points to a defined Unit which is to be raised to a given exponential power. Any base Unit can be used, but the most common will be subtypes of Simple.

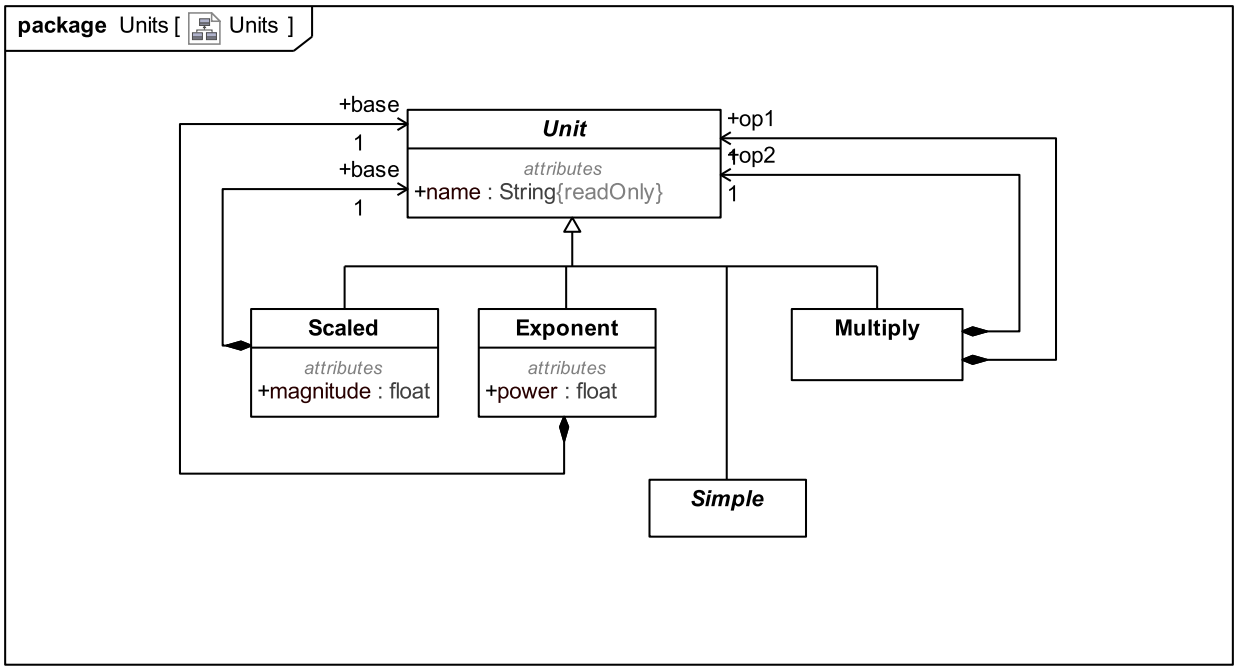

Figure 7.4: Units Package Class Diagram

7.4.2 Multiply

The Multiply class uses op1 as the first multiplicand, and op2 as the second.

Generalizations

Unit

Associations (Required)

op1 : Unit [1]
The first Unit to be referenced.

op2 : Unit [1]
The second Unit to be referenced.

7.4.3 Scaled

The Scaled class takes a base Unit and provides a scaling factor, the magnitude, to create a new unit that is a scale conversion of the same dimensionality. An example would be a gram as 0.001 of a kilogram.

Generalizations

Unit

Attributes (Required)

magnitude : float
Value to scale Base unit by to create this Scaled unit.

Associations (Required)

base : Unit [1]
The Base unit kind that this Scaled unit is built off of.

7.4.4 Simple (Abstract)

Simple is the core class for most Units that will ultimately be created using SENSR. It is abstract, and has seven normative subclasses, defined in Clause 7.5.

Generalizations

Unit

7.4.5 Unit (Abstract)

Unit is the base class for an entire family of descriptors for the units that physical observations are to be measured in.

Attributes (Required)

name : String
Name of the Unit

7.5 FoundationalUnits

The FoundationalUnits package provides a core set of Units that can be composed into highly complex multi-dimensioned Units. The seven foundational units were chosen to correspond to the seven basic units of the SI system. From these, any physical phenomena should be describable. These Units are 'pure' in the sense that they do not have a specific measure, and do not map to a specific measuring system. They are therefore suitable to be mapped to any system, such as SI (or metric), Imperial, or US Customary.

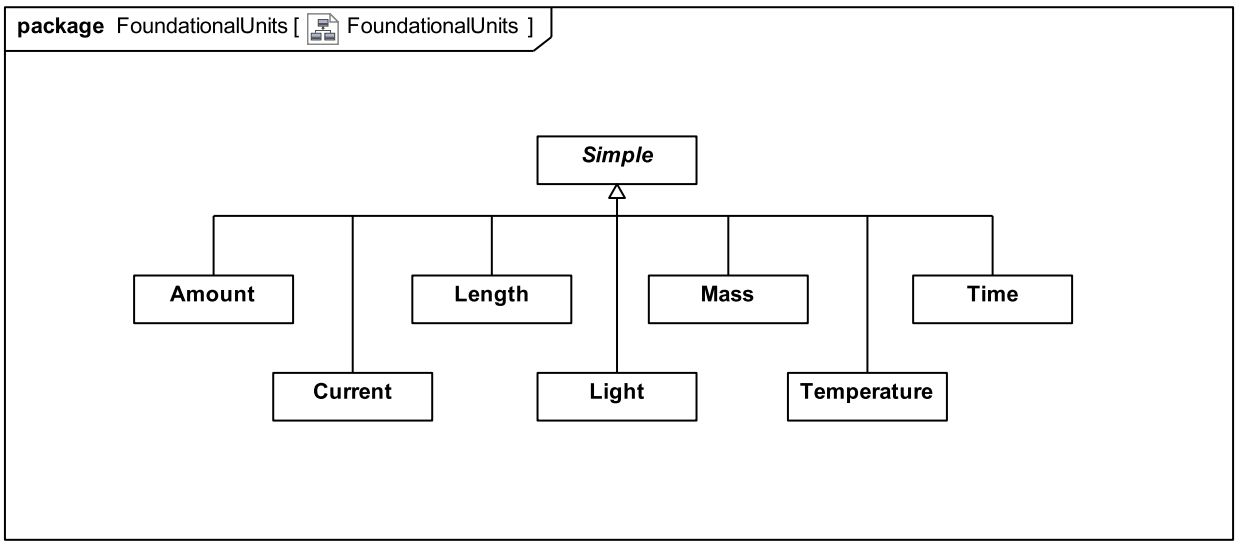

Figure 7.5: Simple Units Class Diagram

7.5.1 Amount

Amount describes a quantity. This can be the quantity of a substance, such as a chemical, or the number of birds in a flock, or how many dozen eggs are in stock. If none of the other six FoundationalUnits correspond to what you are measuring, use Amount as your starting point.

Generalizations

Simple

7.5.2 Current

Current describes the rate of net flow of electric charge past a specific spatial point.

Generalizations

Simple

7.5.3 Length

Length describes a unit of physical linear measure.

Generalizations

Simple

7.5.4 Light

Light defines a unit for luminous intensity.

Generalizations

Simple

7.5.5 Mass

Mass is a descriptor for a unit of the physical property which is a measure of its resistance to acceleration.

Generalizations

Simple

7.5.6 Temperature

Temperature is a descriptor of the thermodynamic temperature of a system, a measure of the mean kinetic energy of the components of the system.

Generalizations

Simple

7.5.7 Time

Time describes a unit of, well, time.

Generalizations

Simple

8 UnitsLibrary (informative)

8.1 CompoundUnits

SENSR does not offer normative units, nor does it offer normative compound dimensional units. At the most abstract, SENSR provides 'pure' units in the FoundationalUnits package which are then filled in to form a basic unit set.

Compound Units combine these Foundational Units into more complex forms, such as velocity (Length / Time), acceleration (Length / Time Squared), area (Length Squared), and volume (Length Cubed).

This is accomplished through the Unit::Multiply and Unit::Exponent classes. In general, use the Exponent class to raise a Foundational Unit to the proper power, including negative powers, and then Multiply to combine like terms into a single Compound Unit. Common examples are included in Figure 8.1.

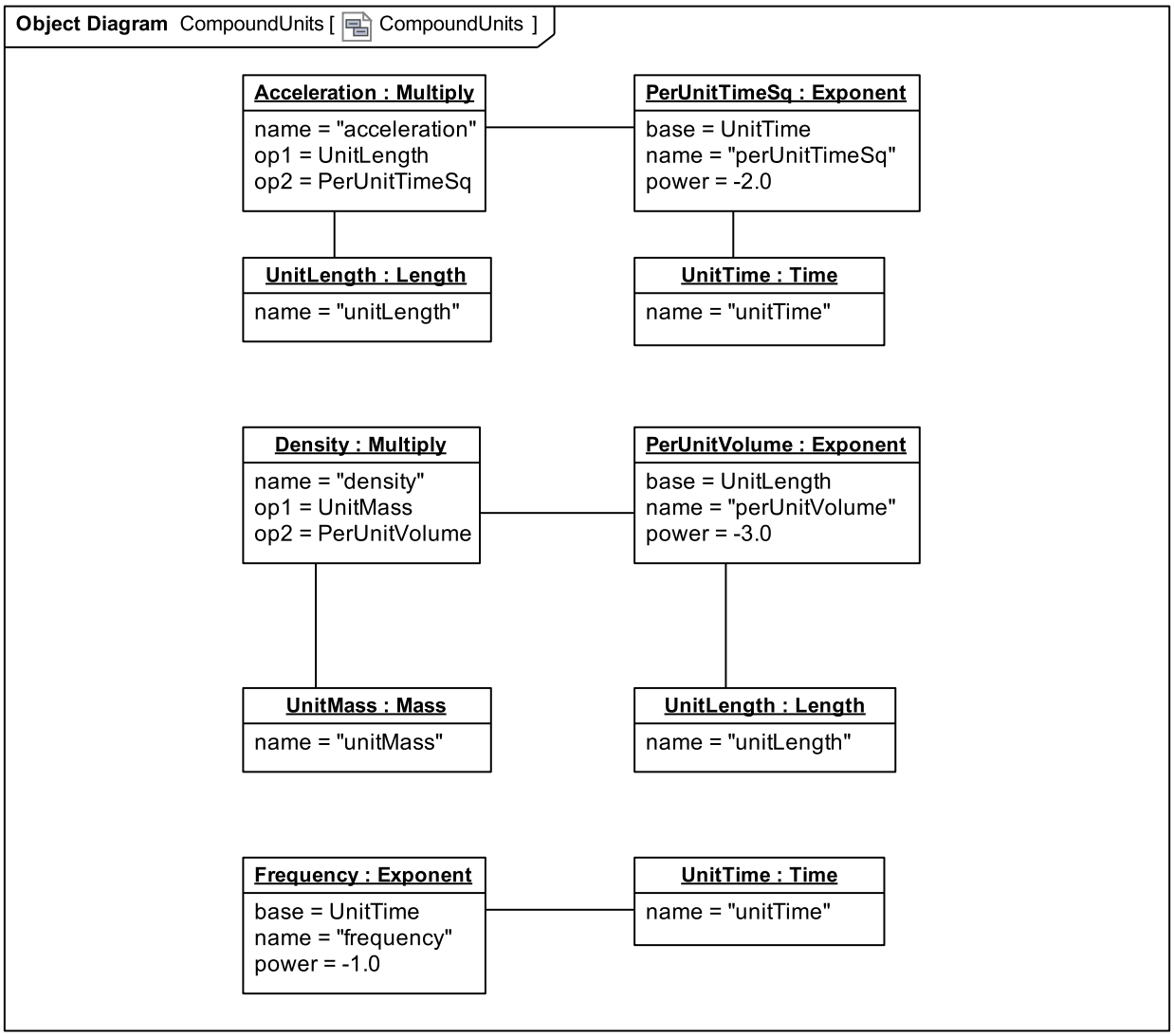

Figure 8.1: Compound Units Object Diagram

8.2 SIUnits

The International System of units (SI), commonly known as the metric system, is arguably the most common and important unit system in technical environments. It is not, however, the only unit system. Instead of creating a normative SI Unit library, SENSR offers vendors and consumers a way to jointly describe SI Units in a natural way.

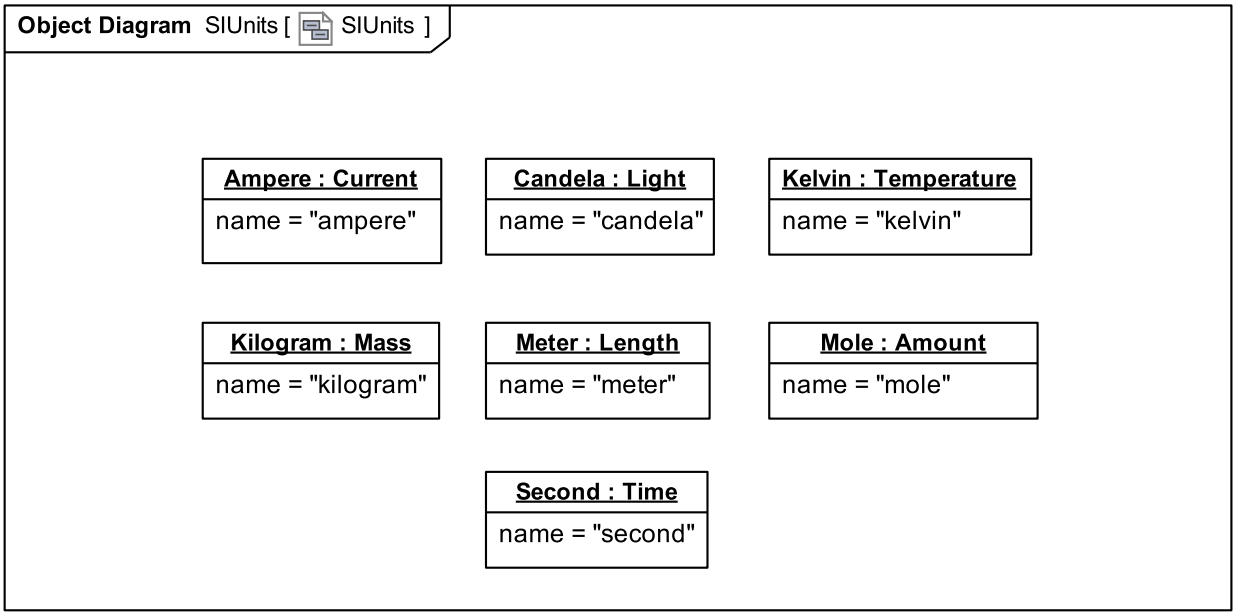

Figure 8.2: SI Units Object Diagram

8.3 CompoundSIUnits

This library provides examples of how to use the SI Units and the techniques from the Compound Units to create the common and useful forms seen in industry. Any physical phenomenon that can be measured using SI Units can be modeled in this manner. Common examples are shown in Figure 8.3.

8.4 USCustomary

An example of how the seven Simple Units can be used to create instances corresponding to the US Customary unit system. Examples of Scaled units and compound units are provided.

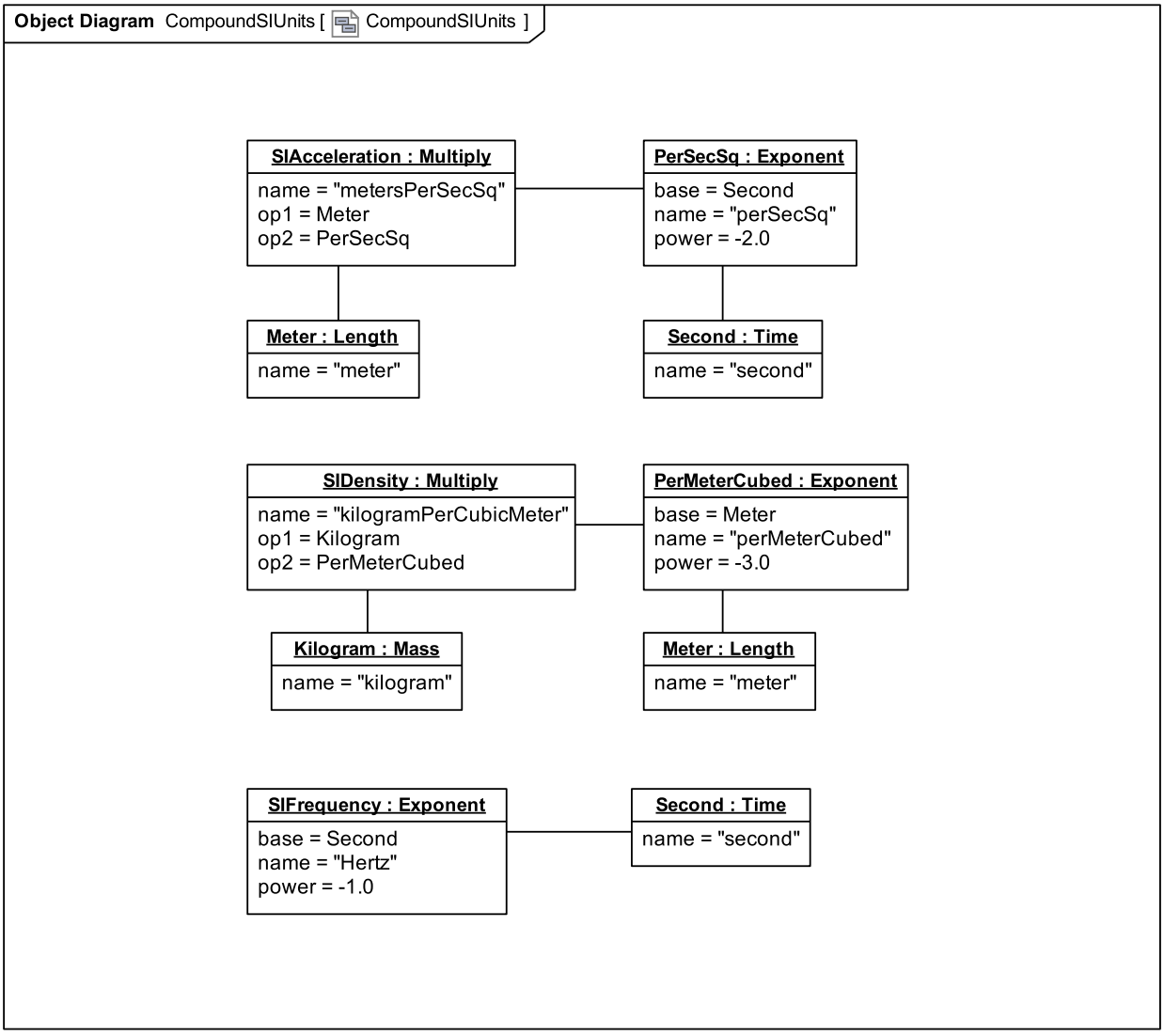

Figure 8.3: Compound SI Units Object Diagram

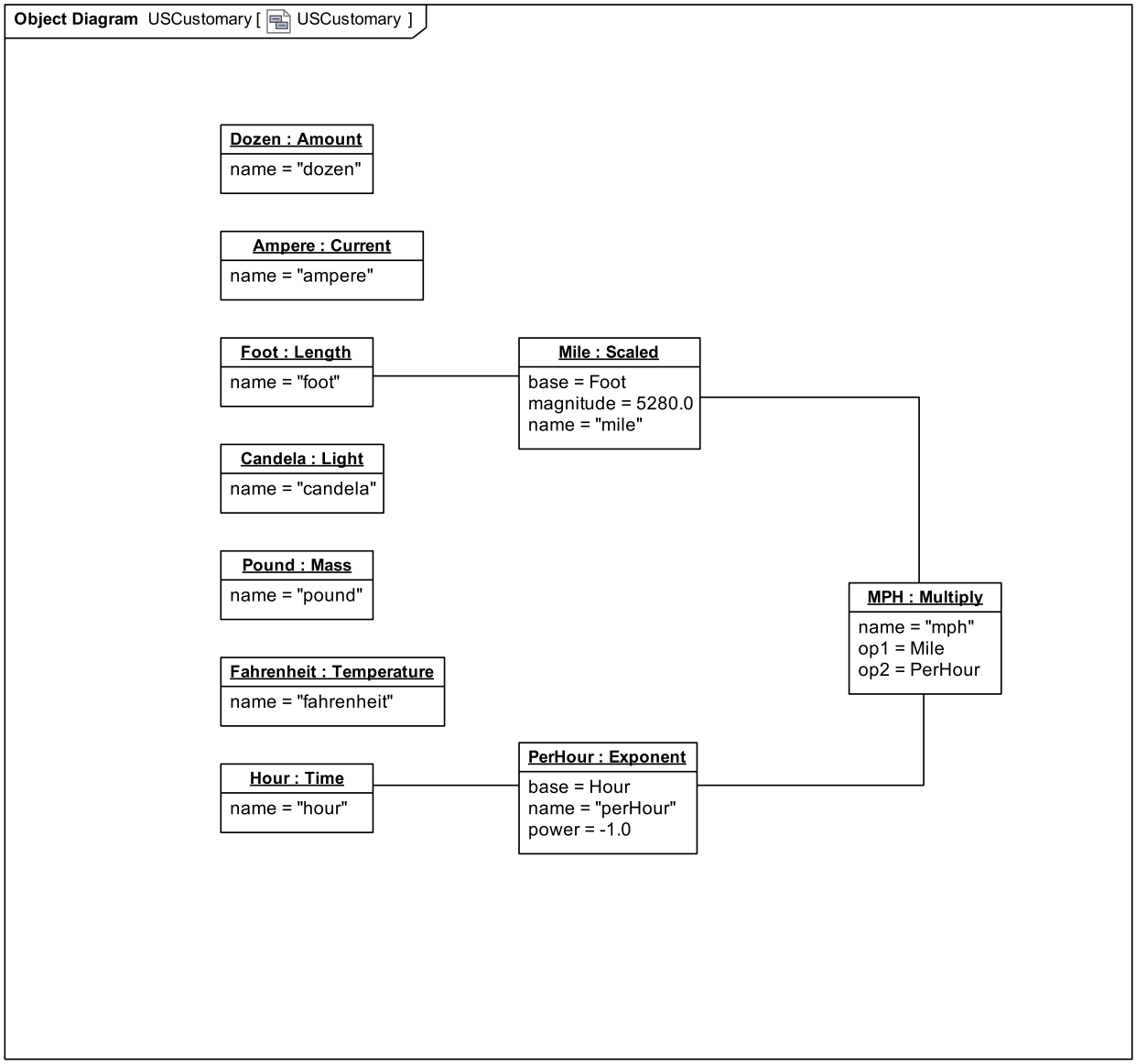

Figure 8.4: US Customary Units Object Diagram

9 Examples (informative)

9.1 Example 1

A simple example showing a DataSheet describing two DataElements. The first is a 32-bit IEEE 754 unsigned floating point number that represents frequency (Hz) in the SI unit system. The second is similarly a 32-bit IEEE 754 unsigned floating point number, but it represents acceleration in the SI unit system.

Listing 9.1 shows a fully expanded form of this data, with redundancy but clarity.

```
<?xml version='1.0' encoding='UTF-8'?>
<xmi:XMI xmlns:xmi='http://www.omg.org/spec/XMI/20131001'
         xmlns:sensr='http://www.omg.org/spec/SENSR/20191001'>

<!--
    frequency: 32-bit IEEE 754 Hz (1/sec)
    accleration: 32-bit IEEE 754 (m/sec^2)
-->

<sensr:DataSheet>
    <name xmi:type="String">Example1</name>
    <vendorID xmi:type="String">0</vendorID>
    <sensr:Quantity>
        <name xmi:type="String">frequency</name>
        <syntax xmi:type="sensr:FixedSize">
            <name xmi:type="String">32bit</name>
            <bits xmi:type="int">32</bits>
        </syntax>
        <unit xmi:type="sensr:Exponent">
            <name xmi:type="String">perSec</name>
            <power xmi:type="float">-1.0</power>
            <base xmi:type="sensr:Time">
                <name xmi:type="String">second</name>
            </base>
        </unit>
        <description xmi:type="FloatingPoint">
            <name xmi:type="String">IEEE754-1</name>
            <isSigned xmi:type="boolean">false</isSigned>
            <base xmi:type="int">10</base>
            <reference xmi:type="String">
                "https://ieeexplore.ieee.org/document/4610935"
            </reference>
        </description>
    </sensr:Quantity>
    <sensr:Quantity>
        <name xmi:type="String">acceleration</name>
        <syntax xmi:type="sensr:FixedSize">
            <name xmi:type="String">32bit</name>
            <bits xmi:type="int">32</bits>
        </syntax>
        <unit xmi:type="sensr:Multiply">
            <name xmi:type="String">acceleration</name>
```

```
            <op1 xmi:type="sensr:Length">
                <name>meter</name>
            </op1>
            <op2 xmi:type="sensr:Exponent">
                <power xmi:type="float">-2.0</power>
                <base xmi:type="sensr:Time">
                    <name xmi:type="String">second</name>
                </base>
            </op2>
        </unit>
        <description xmi:type="FloatingPoint">
            <name xmi:type="String">IEEE754</name>
            <isSigned xmi:type="boolean">false</isSigned>
            <base xmi:type="int">10</base>
            <reference xmi:type="String">
                "https://ieeexplore.ieee.org/document/4610935"
            </reference>
        </description>
    </sensr:Quantity>
</sensr:DataSheet>
```

Listing 9.1: ..//Example 1

9.2 Example 2

This is an equivalent example to Example 1, but instead of explicit instances, it shares instances where possible. You can see the reduction in size in Listing 9.2. The redundancy is eliminated, with a slight reduction in clarity.

```
<?xml version='1.0' encoding='UTF-8'?>
<xmi:XMI xmlns:xmi='http://www.omg.org/spec/XMI/20131001'
         xmlns:sensr='http://www.omg.org/spec/SENSR/20191001'>

<!--
    frequency: 32-bit IEEE 754 Hz (1/sec)
    accleration: 32-bit IEEE 754 (m/sec^2)
-->

<sensr:DataSheet>
    <name xmi:type="String">Example2</name>
    <vendorID xmi:type="String">0</vendorID>
    <sensr:Quantity>
        <name xmi:type="String">frequency</name>
        <syntax xmi:type="sensr:FixedSize" xmi:id="32bit">
            <name xmi:type="String">32-bit</name>
            <bits xmi:type="int">32</bits>
        </syntax>
        <unit xmi:type="sensr:Exponent">
            <power xmi:type="float">-1.0</power>
            <base xmi:type="sensr:Time" xmi:id="Second">
                <name xmi:type="String">Second</name>
            </base>
        </unit>
        <description xmi:type="FloatingPoint" xmi:id="IEEE754-FP">
            <name xmi:type="String">IEEE754</name>
```

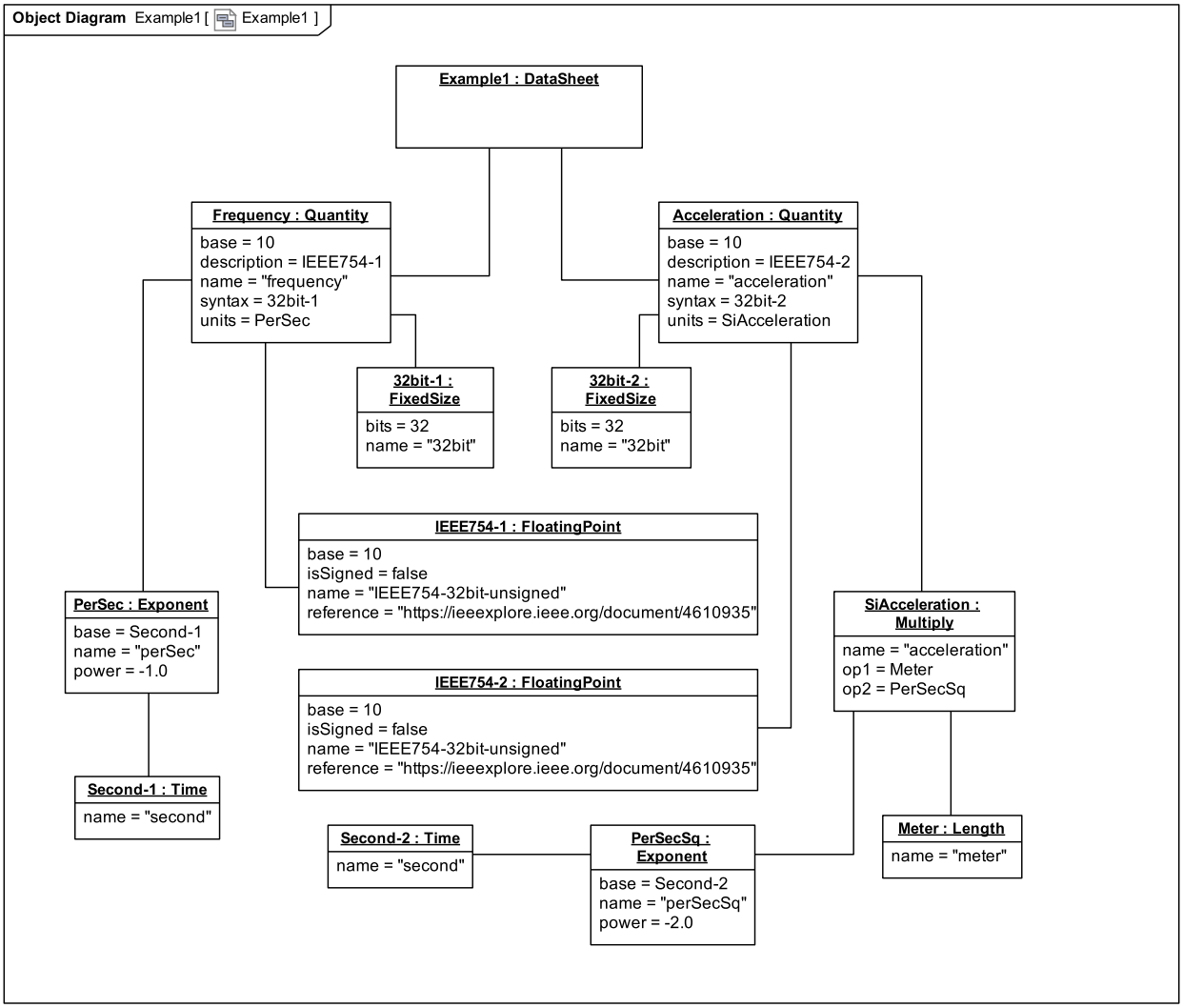

Figure 9.1: Example 1 Object Diagram

```
            <isSigned xmi:type="boolean">false</isSigned>
            <base xmi:type="int">10</base>
            <reference xmi:type="String">
                "https://ieeexplore.ieee.org/document/4610935"
            </reference>
        </description>
    </sensr:Quantity>
    <sensr:Quantity>
        <name xmi:type="String">acceleration</name>
        <syntax xmi:idref="32bit"/>
        <unit xmi:type="sensr:Multiply">
            <op1 xmi:type="sensr:Length" xmi:id="Meter">
                <name>Meter</name>
            </op1>
            <op2 xmi:type="sensr:Exponent">
                <power xmi:type="float">-2.0</power>
                <base xmi:idref="Second"/>
            </op2>
        </unit>
        <description xmi:idref="IEEE754-FP"/>
    </sensr:Quantity>
</sensr:DataSheet>
```

Listing 9.2: ..//Example 2

9.3 Example 3

This is an example of three identical data types being reported in succession. Each is a signed 8-bit integer, reporting acceleration in SI units across the x, y, and z axes.

The corresponding XML file can be found in Listing 9.3. You can see how the xmi:id and xmi:idref mechanism can be used to reduce the size of the XML file by referring to pre-existing elements in the file. This allows a user of SENSR to achieve both a self-contained file, and compactness.

```
<?xml version='1.0' encoding='UTF-8'?>
<xmi:XMI xmlns:xmi='http://www.omg.org/spec/XMI/20131001'
         xmlns:sensr='http://www.omg.org/spec/SENSR/20191001'>

<!--
    vectored acceleration:
        accleration: 8-bit signed int (m/sec^2)
        accleration: 8-bit signed int (m/sec^2)
        accleration: 8-bit signed int (m/sec^2)
-->

<sensr:DataSheet>
    <name xmi:type="String">VectoredAcceleration</name>
    <vendorID xmi:type="String">0</vendorID>
    <sensr:Composite>
        <name xmi:type="String">vectored_acceleration</name>
        <sensr:Quantity>
            <name>x</name>
            <reference xmi:type="String"/>
            <syntax xmi:type="sensr:FixedSize" xmi:id="8bit">
                <name xmi:type="String">8-bit</name>
```

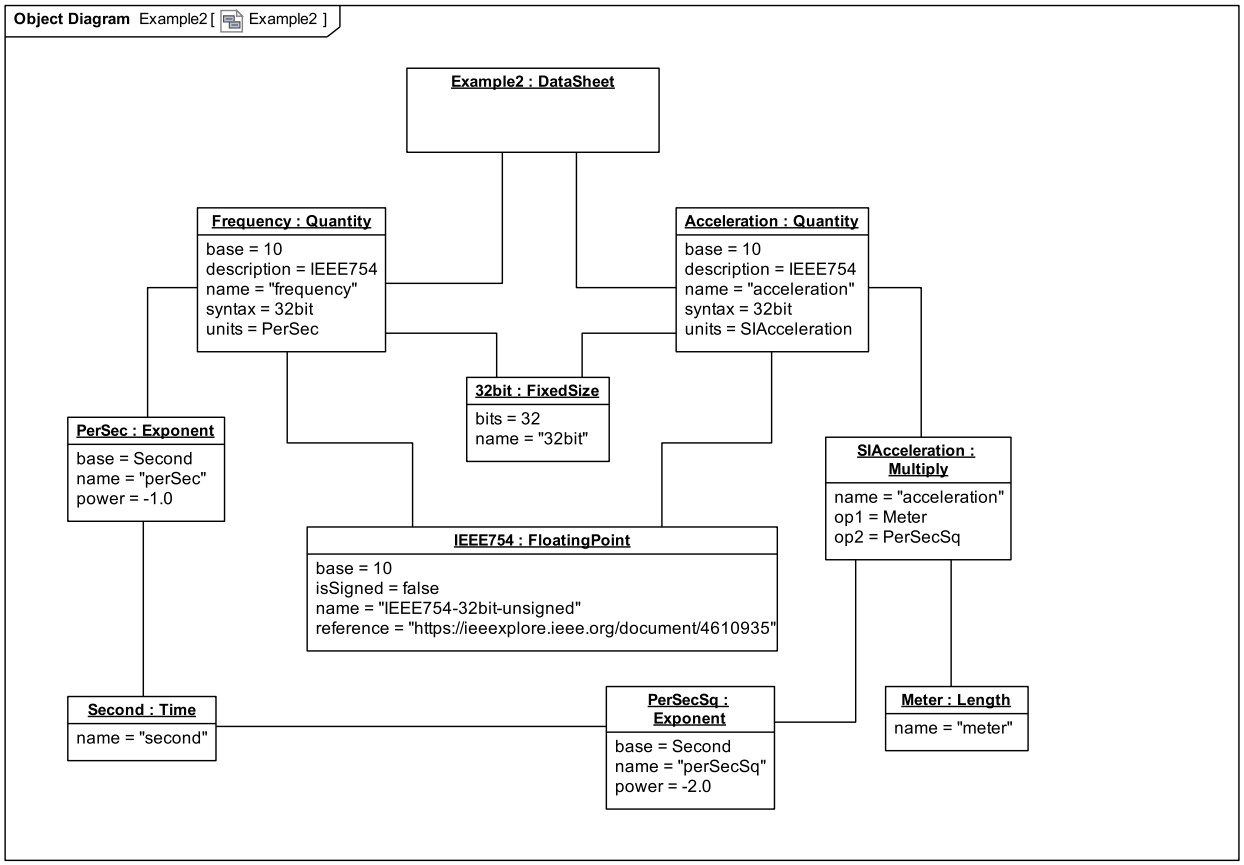

Figure 9.2: Example 2 Object Diagram

```
            <bits xmi:type="int">8</bits>
        </syntax>
        <unit xmi:type="sensr:Multiply" xmi:id="Acceleration">
            <op1 xmi:type="sensr:Length">
                <name>Meter</name>
            </op1>
            <op2 xmi:type="sensr:Exponent">
                <power xmi:type="float">-2.0</power>
                <base xmi:type="sensr:Time">
                    <name xmi:type="String">Second</name>
                </base>
            </op2>
        </unit>
        <description xmi:type="Integer" xmi:id="integer">
            <name xmi:type="String">short</name>
            <isSigned xmi:type="boolean">true</isSigned>
            <base xmi:type="int">10</base>
        </description>
    </sensr:Quantity>
    <sensr:Quantity>
        <name>y</name>
        <reference xmi:type="String"/>
        <syntax xmi:idref="8bit"/>
        <unit xmi:idref="Acceleration"/>
        <description xmi:idref="integer"/>
    </sensr:Quantity>
    <sensr:Quantity>
        <name>z</name>
        <reference xmi:type="String"/>
        <syntax xmi:idref="8bit"/>
        <unit xmi:idref="Acceleration"/>
        <description xmi:idref="integer"/>
    </sensr:Quantity>
</sensr:Composite>
</sensr:DataSheet>
```

Listing 9.3: ..//Example 3

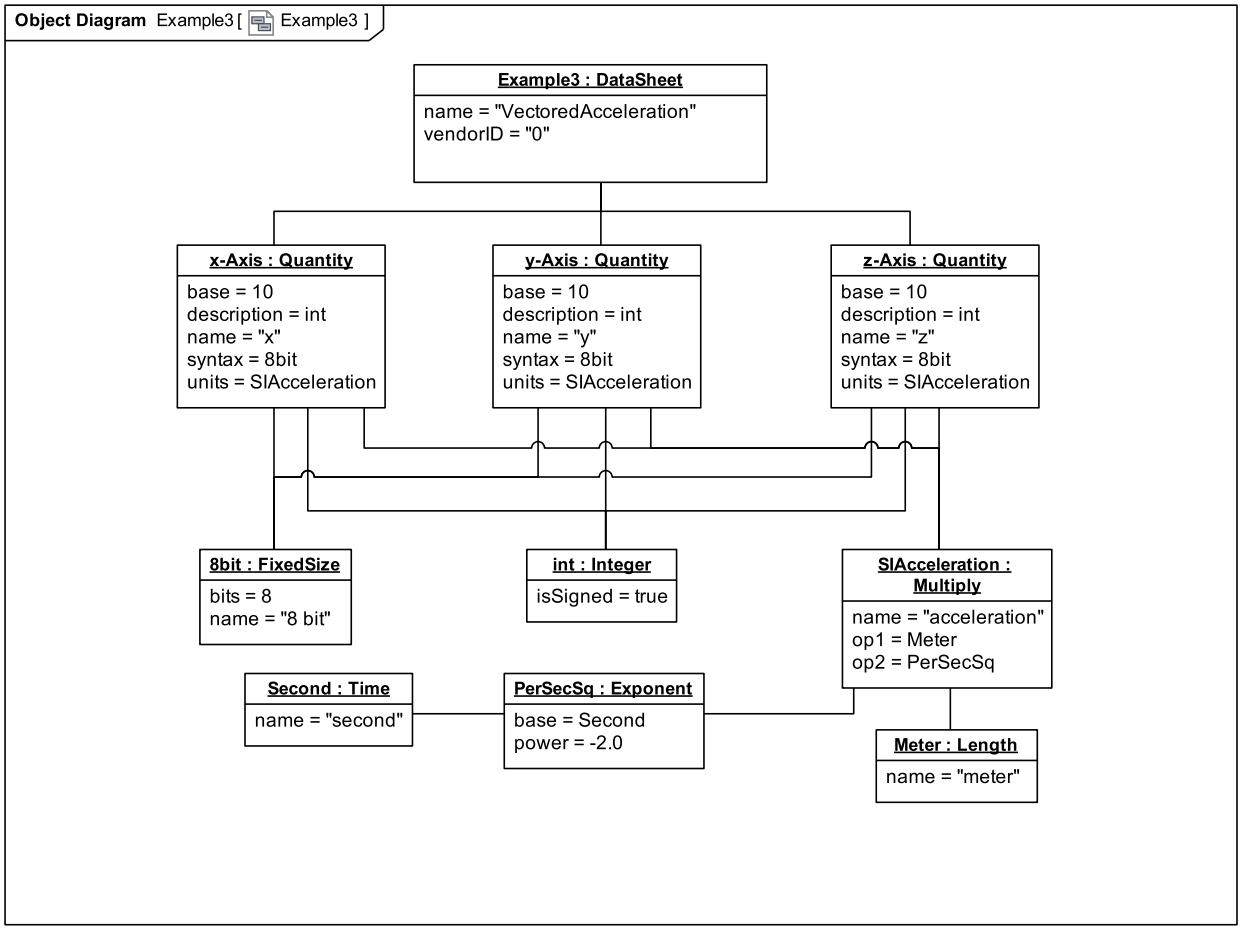

Figure 9.3: Example 3 Object Diagram

10 PSM (Bluetooth LE) (informative)

Bosch has created a Bluetooth LE (BLE) Advertising implementation, and provided example data being transmitted. The following PSM is a description of the model of the example data, and is presented in both XML and JSON formats suitable for machine reading and configuration. The UML object diagram in Figure 10 is a match for the example Listing 1.1 from Section 1. Listing 10.1 defines this model in XML. Note that the `xmlns` directive is pointing to a speculative URI. The correct URI will be generated upon adoption of this specification. The JSON representation requires a slight change, in that to reference JSON elements, one must use the name of the element in the JSON tree. To do this requires adding a 'type' field to each node to indicate which SENSR class it is an instance of. This is shown in Listing 10.2.

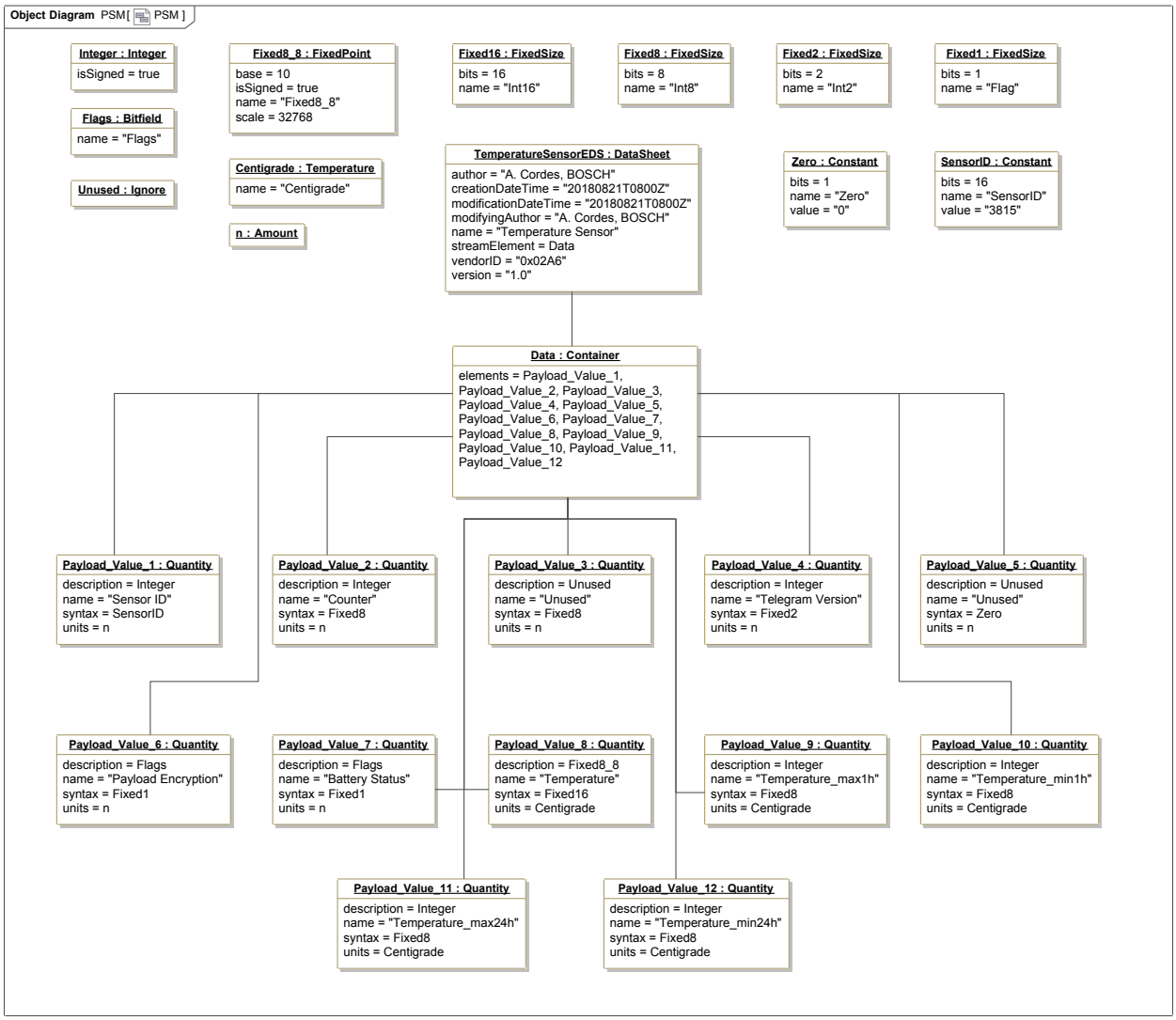

Figure 10.1: PSM Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:sensr="https://www.omg.org/spec/SENSR/20191000">
  <DataSheet id="TemperatureSensorEDS" name="Temperature␣Sensor" version="1.0"
```

```
    author="A. Cordes, BOSCH" creationDateTime="20180821T0800Z"
    modifyingAuthor="A. Cordes, BOSCH" modificationDateTime="20180821T0800Z">
    <Integer isSigned='true' id='Integer'/>
    <Bitfield name='Flags' id='Flags'/>
    <Ignore id='Unused'/>
    <FixedPoint id='Fixed8_8' name='Fixed8_8' scale='32768'/>
    <FixedSize id='Fixed16' name='Int16' bits='16'/>
    <FixedSize id='Fixed8' name='Int8' bits='8'/>
    <FixedSize id='Fixed2' name='Int2' bits='2'/>
    <FixedSize id='Fixed1' name='Flag' bits='1'/>
    <Temperature id='Centigrade' name='Centigrade'/>
    <Amount id='n'/>
    <Constant id='Zero' name='Zero' bits='1' value='0'/>
    <Constant id='SensorID' name='SensorID' bits='16' value='3815'/>
    <streamElement>
      <Container id='Data'>
        <elements>
          <Quantity id='Payload_Value_1' name='Sensor ID'
            description='Integer' syntax='SensorID' units='n'/>
          <Quantity id='Payload_Value_2' name='Counter'
            description='Integer' syntax='Fixed8' units='n'/>
          <Quantity id='Payload_Value_3' name='Unused'
            description='Unused' syntax='Fixed8' units='n'/>
          <Quantity id='Payload_Value_4' name='Telegram Version'
            description='Integer' syntax='Fixed2'  units='n'/>
          <Quantity id='Payload_Value_5' name='Unused'
            description='Unused' syntax='Zero' units='n'/>
          <Quantity id='Payload_Value_6' name='Payload Encryption'
            description='Flags' syntax='Fixed1' units='n'/>
          <Quantity id='Payload_Value_7' name='Battery Status'
            description='Flags' syntax='Fixed1' units='n'/>
          <Quantity id='Payload_Value_8' name='Temperature'
            description='Fixed8_8' syntax='Fixed16' units='Centigrade'/>
          <Quantity id='Payload_Value_9' name='Temperature_max1h'
            description='Integer' syntax='Fixed8' units='Centigrade'/>
          <Quantity id='Payload_Value_10' name='Temperature_min1h'
            description='Integer' syntax='Fixed8' units='Centigrade'/>
          <Quantity id='Payload_Value_11' name='Temperature_max24h'
            description='Integer' syntax='Fixed8' units='Centigrade'/>
          <Quantity id='Payload_Value_12' name='Temperature_min24h'
            description='Integer' syntax='Fixed8' units='Centigrade'/>
        </elements>
      </Container>
    </streamElement>
  </DataSheet>
</xmi:XMI>
```

Listing 10.1: PSM XML

```
{"TemperatureSensorEDS": {
  "type": "DataSheet",
  "name": "Temperature Sensor",
  "version": "1.0",
  "author": "A. Cordes, BOSCH",
  "creationDateTime": "20180821T0800Z",
```

```
"modifyingAuthor": "A.␣Cordes,␣BOSCH",
"modificationDateTime": "20180821T0800Z",
"Integer": {
  "type": "Integer",
  "isSigned": "true"
},
"Flags":{
  "type": "Bitfield",
  "name": "Flags"
},
"Unused": {
  "type": "Ignore"
},
"Fixed8_8": {
  "type": "FixedPoint",
  "name": "Fixed8_8",
  "base": "10",
  "scale": "32768"
},
"Fixed16": {
  "type": "FixedSize",
  "name": "Int16",
  "bits": "16"
},
"Fixed8": {
  "type": "FixedSize",
  "name": "Int8",
  "bits": "8"
},
"Fixed2": {
  "type": "FixedSize",
  "name": "Int2",
  "bits": "2"
},
"Fixed1": {
  "type": "FixedSize",
  "name": "Flag",
  "bits": "1"
},
"Centigrade": {
  "type": "Temperature",
  "name": "Centigrade"
},
"n": {
  "type": "Amount",
},
"Zero": {
  "type": "Constant",
  "name": "Zero",
  "bits": "1",
  "value": "0"
},
"SensorID": {
  "type": "Constant",
```

```
    "name": "SensorID",
    "bits": "16",
    "value": "3815"
  },
  "Data": {
    "type": "Container",
    "elements": {
      "Payload_Value_1": {
        "type": "Quantity",
        "name": "Sensor␣ID",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#SensorID"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_2": {
        "type": "Quantity",
        "name": "Counter",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_3": {
        "type": "Quantity",
        "name": "Unused",
        "description": {"$ref": "/#Unused"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_4": {
        "type": "Quantity",
        "name": "Telegram␣Version",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed2"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_5": {
        "type": "Quantity",
        "name": "Unused",
        "description": {"$ref": "/#Unused"},
        "syntax": {"$ref": "/#Zero"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_6": {
        "type": "Quantity",
        "name": "Payload␣Encryption",
        "description": {"$ref": "/#Flags"},
        "syntax": {"$ref": "/#Fixed1"},
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_7": {
        "type": "Quantity",
        "name": "Battery␣Status",
        "description": {"$ref": "/#Flags"},
        "syntax": {"$ref": "/#Fixed1"},
```

```
        "units": {"$ref": "/#n"}
      },
      "Payload_Value_8": {
        "type": "Quantity",
        "name": "Temperature",
        "description": {"$ref": "/#Fixed8_8"},
        "syntax": {"$ref": "/#Fixed16"},
        "units": {"$ref": "/#Centigrade"}
      },
      "Payload_Value_9": {
        "type": "Quantity",
        "name": "Temperature_max1h",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#Centigrade"}
      },
      "Payload_Value_10": {
        "type": "Quantity",
        "name": "Temperature_min1h",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#Centigrade"}
      },
      "Payload_Value_11": {
        "type": "Quantity",
        "name": "Temperature_max24h",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#Centigrade"}
      },
      "Payload_Value_12": {
        "type": "Quantity",
        "name": "Temperature_min24h",
        "description": {"$ref": "/#Integer"},
        "syntax": {"$ref": "/#Fixed8"},
        "units": {"$ref": "/#Centigrade"}
      }
    }
  }
}
```

Listing 10.2: PSM JSON