The Consultative Committee for Space Data Systems

Draft Recommendation for
Space Data System Standards

SPACECRAFT ONBOARD INTERFACE SERVICES—XML SPECIFICATION FOR ELECTRONIC DATA SHEETS

DRAFT RECOMMENDED STANDARD

CCSDS 876.0-R-3

RED BOOK
August 2018

The Consultative Committee for Space Data Systems

Draft Recommendation for Space Data System Standards

SPACECRAFT ONBOARD INTERFACE SERVICES— XML SPECIFICATION FOR ELECTRONIC DATA SHEETS

DRAFT RECOMMENDED STANDARD

CCSDS 876.0-R-3

RED BOOK
August 2018

AUTHORITY

Issue:	Red Book, Issue 3
Date:	August 2018
Location:	Not Applicable

(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF AUTHORITY:)

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the e-mail address below.

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
E-mail: secretariat@mailman.ccsds.org

STATEMENT OF INTENT

(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF INTENT:)

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommended Standards** and are not considered binding on any Agency.

This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

- o Whenever a member establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommended Standard**. Establishing such a **standard** does not preclude other provisions which a member may develop.

- o Whenever a member establishes a CCSDS-related **standard**, that member will provide other CCSDS members with the following information:

  - -- The **standard** itself.

  - -- The anticipated date of initial operational capability.

  - -- The anticipated duration of operational service.

- o Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommended Standard** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommended Standard** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or (3) be retired or canceled.

In those instances when a new version of a **Recommended Standard** is issued, existing CCSDS-related member standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such standards or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Standard.

FOREWORD

This document is a technical Recommended **Standard** for the XML Specification for Electronic Data Sheets for Onboard Devices and has been prepared by the Consultative Committee for Space Data Systems (CCSDS). The XML Specification for Electronic Data Sheets for Onboard Devices described herein is intended for missions that are cross-supported between Agencies of the CCSDS, in the framework of the Spacecraft Onboard Interface Services (SOIS) CCSDS area.

This Recommended Standard specifies the XML schema, and associated constraints, to be used by space missions to describe the data interface of an onboard device accessed over a spacecraft subnetwork. The XML Specification for Electronic Data Sheets for Onboard Devices may be used for an onboard device regardless of the particular type of data link or protocol being used for communication with that device.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in the *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the e-mail address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d’Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People’s Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

PREFACE

This document is a draft CCSDS Recommended Standard. Its 'Red Book' status indicates that the CCSDS believes the document to be technically mature and has released it for formal review by appropriate technical organizations. As such, its technical contents are not stable, and several iterations of it may occur in response to comments received during the review process.

Implementers are cautioned **not** to fabricate any final equipment in accordance with this document's technical content.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 876.0-R-1	Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets for Onboard Devices, Draft Recommended Standard, Issue 1	July 2015	Original draft issue, superseded
CCSDS 876.0-R-2	Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets, Issue 2	June 2016	Draft issue 2, superseded
CCSDS 876.0-R-3	Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets, Draft Recommended Standard, Issue 3	August 2018	Current draft (note)

NOTE – Changes from the previous issue are too numerous to permit markup.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

1 INTRODUCTION

1.1 PURPOSE AND SCOPE OF THIS DOCUMENT

This document is one of a family of documents specifying the Spacecraft Onboard Interface Services (SOIS)-compliant services to be provided in support of applications.

This document defines the XML Specification for SOIS Electronic Data Sheet (SEDS) for Onboard Devices. The SEDS is for use in electronically describing the data interfaces offered by flight hardware such as sensors and actuators.

Once the description is in machine-readable format, a toolchain can be used to facilitate the various phases in the life of a space vehicle.

The definition encompasses the XML representation of the functional interfaces offered by any protocols used to access the data interfaces.

1.2 APPLICABILITY

This document applies to any mission or equipment claiming to provide SEDS for Onboard Devices.

1.3 RATIONALE

SOIS provides an XML schema specification in order to enable toolchain compatibility amongst systems implementing interfaces defined by SOIS EDS.

1.4 DOCUMENT STRUCTURE

This document has four major sections:

– Section 1 contains administrative information, definitions and references.

– Section 2 provides a brief overview of Electronic Data Sheets for onboard devices.

– Section 3 provides a normative description of the structure of the SEDS XML schema and compliant SEDS XML instances.

– Section 4 provides normative instructions on how to construct valid instantiations of SEDS for onboard devices, describing requirements and constraints beyond those imposed by the schema.

In addition, the following annexes are provided:

– Annex A comprises a Protocol Implementation Conformance Statement (PICS) proforma.

– Annex B discusses security, Space Assigned Numbers Authority (SANA), and patent considerations relating to the specifications of this document.

– Annex C contains a list of acronyms.

– Annex D contains a list of informative references.

– Annex E provides for illustrative purposes an example instantiation of the SEDS XML schema.

1.5 CONVENTIONS AND DEFINITIONS

1.5.1 DEFINITIONS

1.5.1.1 Definitions from the Open Systems Interconnection Reference Model

The document is defined using the style established by the Open Systems Interconnection (OSI) Basic Reference Model (reference [D1]). This model provides a common framework for the development of standards in the field of systems interconnection.

The following terms used in this Recommended Standard are adapted from definitions given in (reference [D1]):

layer: A subdivision of the architecture, constituted by subsystems of the same rank.

service: A capability of a layer, and the layers beneath it (service providers), provided to the service users at the boundary between the service providers and the service users.

1.5.1.2 Terms Defined in this Recommended Standard

For the purposes of this Recommended Standard, the following definitions apply:

application: An algorithm that applies SOIS services to accomplish the goals of a mission.

component: A logical element of a system accessed through defined interfaces. A component may be purely conceptual or realized in software or hardware (e.g., as a field-programmable gate array).

device: A physical element of a system accessed through subnetwork-layer interfaces. (See reference [D2].)

dictionary of terms, DoT: Ontology of terms used to describe data in interfaces in Electronic Data Sheets. (See reference [D2].)

Electronic Data Sheet, EDS: Electronic description of some details of a device, software component or standard. Unless qualified with the acronym ‘SOIS’, this term is general, referring to any machine-readable data sheet. (See **SOIS Electronic Data Sheet, SEDS**.)

engineering profile: A collection of attributes used to define the meaning of an item of data used in operations and parameters. (See reference [D2].)

glossary: A collection of terms with brief informal explanations of their usage in a particular document.

interface: A facility provided or supplied by a component that allows exchange of data.

portability: The capability of a component to be integrated into an assembly without change either to the component or to the assembly interfaces. Portability requires that the definitions of interfaces be consistent across all systems to which they may be ported. Consistency requires that the terms used to define an interface are defined in the DoT. (See '**toolchain compatibility**.)

semantic attribute: A property of an engineering profile, such as reference frame or unit of measure.

SOIS Electronic Data Sheet, SEDS: Electronic description of a device's metadata, device-specific functional and access interfaces, device-specific access protocol, and, optionally, device abstraction control procedure (see reference [D2]), compliant with SOIS standards. (See **Electronic Data Sheet, EDS**.)

syntactic type: A type of data that is defined by attributes for encoding the data for storage (transmission through time) or communication (transmission through space). An example of an attribute for a syntactic type is the choice of interpretation of bits as an integer, as a floating-point number, or other choices. (See reference [D2].)

term: A word or phrase that has a formally defined interpretation in a particular context of usage. The terms in the SOIS dictionary of terms are defined in the context of describing spacecraft components in Electronic Data Sheets.

toolchain compatibility: Capability to function in a sequence of computer-assisted engineering steps, optionally with locally defined ontology extensions. Toolchain compatibility is a weaker form of interface consistency than portability. The locally defined ontology extensions make it possible for SOIS Electronic Data Sheets to function in a toolchain early in the life of a project without waiting for terms to be defined in the SANA DoT ontology. For complete portability, all terms in an Electronic Data Sheet must be defined in the SANA DoT ontology.

type: A conceptual class that is defined in an EDS as a class. The instances of a type share those properties that define the type. The properties are defined in the dictionary of terms.

value: A formatted instance of data that is acquired from or used as a command to a component.

1.6 NOMENCLATURE

1.6.1 NORMATIVE TEXT

The following conventions apply for the normative specifications in this Recommended Standard:

a) the words ‘shall’ and ‘must’ imply a binding and verifiable specification;

b) the word ‘should’ implies an optional, but desirable, specification;

c) the word ‘may’ implies an optional specification;

d) the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

NOTE – These conventions do not imply constraints on diction in text that is clearly informative in nature.

1.6.2 INFORMATIVE TEXT

In the normative sections of this document, informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

– Overview;

– Background;

– Rationale;

– Discussion.

1.7 REFERENCES

The following publications contain provisions which, through reference in this text, constitute provisions of this document. At the time of publication, the editions indicated were valid. All publications are subject to revision, and users of this document are encouraged to investigate the possibility of applying the most recent editions of the publications indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS publications.

[1] *Spacecraft Onboard Interface Services—Specification for Dictionary of Terms for Electronic Data Sheets for Onboard Components*. Issue 2. Draft Recommendation for Space Data System Practices (Red Book), CCSDS 876.1-R-2. Washington, D.C.: CCSDS, June 2016.

[2] Tim Bray, et al., eds. “Extensible Markup Language (XML) 1.0.” W3C Recommendation. 5th ed., 26 November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[3] Shudi (Sandy) Gao, C. M. Sperberg-McQueen, and Henry S. Thompson, eds. “W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures.” W3C Recommendation. Version 1.1, 5 April 2012. http://www.w3.org/TR/xmlschema11-1/.

[4] David Peterson, et al., eds. “W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes.” W3C Recommendation. Version 1.1, 5 April 2012. http://www.w3.org/TR/xmlschema11-2/.

[5] Jonathan Marsh, David Orchard, and Daniel Veillard, eds. “XML Inclusions (XInclude) Version 1.0.” W3C Recommendation. 2nd ed., 15 November 2006. http://www.w3.org/TR/xinclude/.

[6] *Information Technology—Microprocessor Systems—Floating-Point Arithmetic*. International Standard, ISO/IEC/IEEE 60559:2011. Geneva: ISO, 2011.

[7] *Sixteen-Bit Computer Instruction Set Architecture*. Military Standard, MIL-STD-1750A. Washington, DC: USAF, 2 July 1980.

[8] Julie D. Allen, et al., eds. *The Unicode Standard*. Version 7.0. Mountain View, CA: The Unicode Consortium, October 2014.

[9] *Information Technology—8-Bit Single-Byte Coded Graphic Character Sets—Part 1: Latin Alphabet No. 1*. International Standard, ISO/IEC 8859-1:1998. Geneva: ISO, 1998.

[10] *TC Space Data Link Protocol*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 232.0-B-2. Washington, D.C.: CCSDS, September 2010.

[11] *Space Engineering—SpaceWire—Remote Memory Access Protocol*. ECSS-E-ST-50-52C. Noordwijk, The Netherlands: ECSS Secretariat, 5 February 2010.

[12] *CCSDS File Delivery Protocol (CFDP)*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 727.0-B-4. Washington, D.C.: CCSDS, January 2007.

[13] *Information Processing—Use of Longitudinal Parity to Detect Errors in Information Messages*. 2nd ed. International Standard, ISO 1155:1978. Geneva: ISO, 1978.

[14] “Spacecraft Onboard Interface Services Electronic Data Sheets and Dictionary of Terms.” Space Assigned Numbers Authority. http://sanaregistry.org/r/sois/.

NOTE – Informative references are contained in annex D.

2 OVERVIEW

2.1 GENERAL

The diagram below is a map of the major concepts involved in SEDS, and the relationships among them. The topics this document focuses on are highlighted in blue.

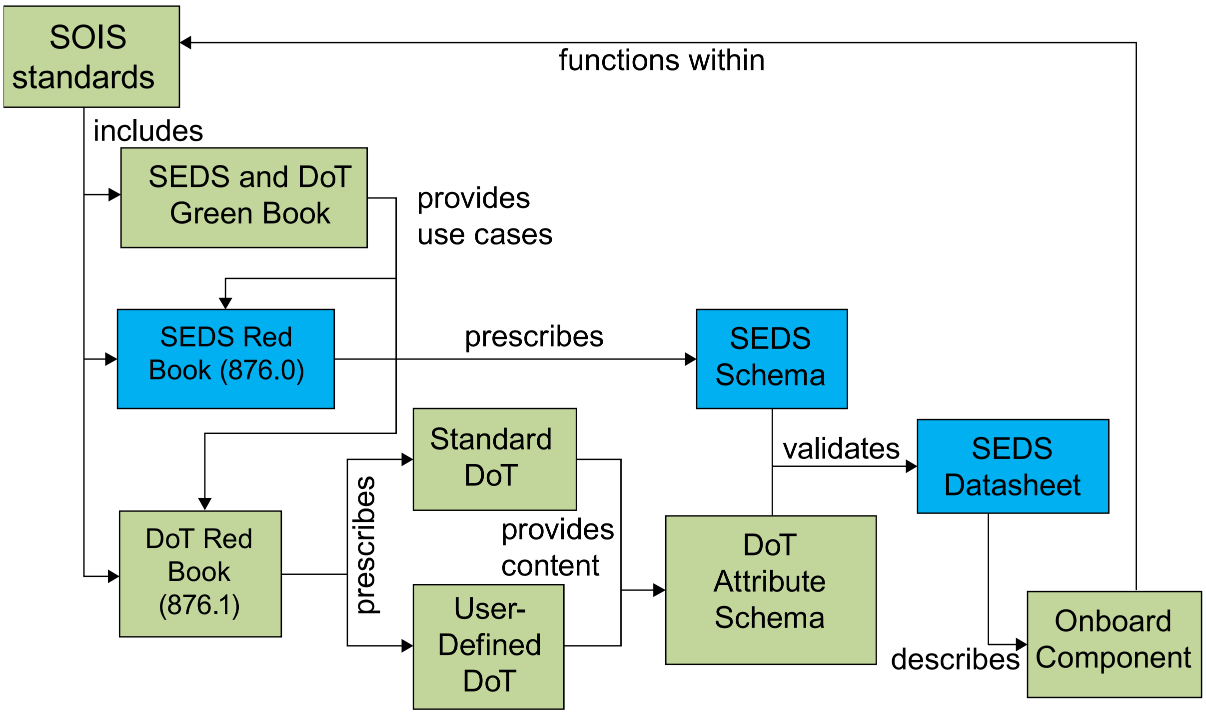

Figure 2-1: SEDS Concept Map

A SOIS Electronic Data Sheet (SEDS) specifies the usage of SOIS standards by an Onboard Component. The XML schema used to validate such a datasheet is defined in two parts:

– The SEDS Schema is defined and described in this document.

– The DoT Attribute schema is extensible, and comes from combining the standard, and optionally, a user-defined Dictionary of Terms (DoT), as described in reference [3].

2.2 THE SUBJECT MATTER OF ELECTRONIC DATA SHEETS

The SOIS Electronic Datasheets (SEDS) are defined within the context of the overall SOIS architecture (see reference [D2]). This section is a series of diagrams beginning at the scope of a spacecraft, and descending in scope to the software layers at the top of the SOIS protocol stack.

Figure 2-2 illustrates how SEDS can describe data interfaces at various points in a spacecraft data system. Devices appear on the left side, with increasing degrees of aggregation of data interfaces in moving to the right side of the diagram.

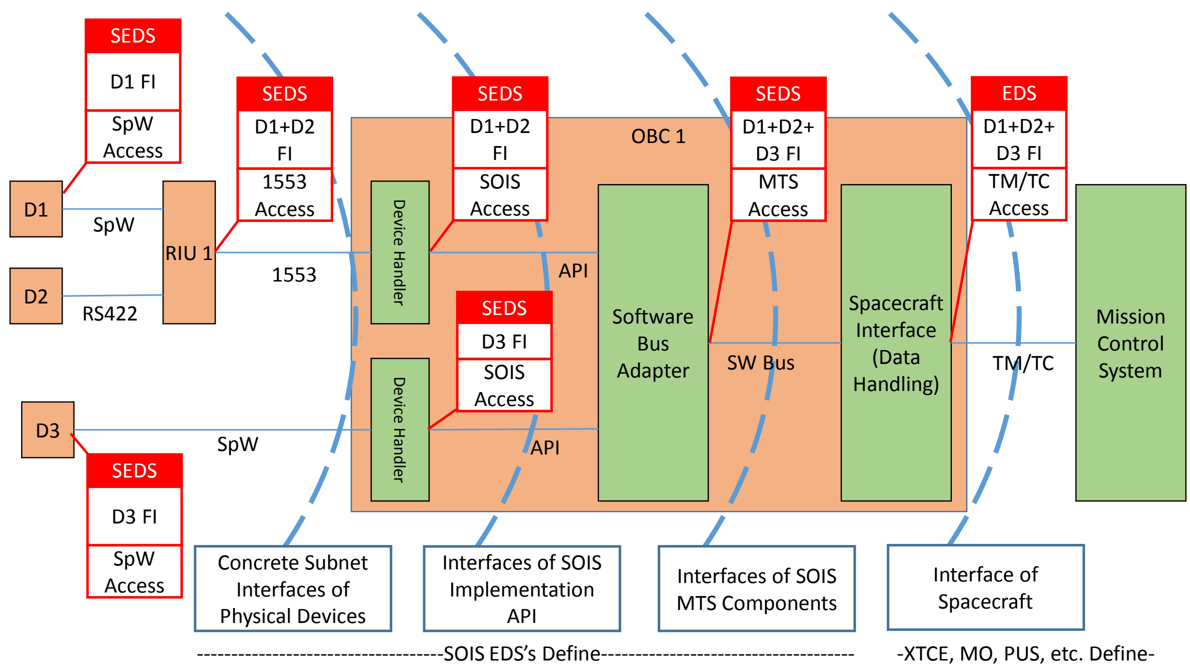

Figure 2-2: SEDS Describe Data Interfaces in a Spacecraft

A SEDS can be used to describe the format of information in any data interface for any onboard device. It can also define a simple structural or behavioural mapping between such interfaces, such as that required to map the binary message format used by a device to the set of commands and parameters it supports.

Some aspects of a device data interface may correspond to standardized protocols directly usable at the Application Layer. The remainder is specified by the SOIS Electronic Datasheet for that device, which captures all device-specific aspects. This includes those specified at some other level of commonality (i.e., agency, company, product line, etc.).

The interfaces defined in a datasheet may then be used in the specification or implementation of applications and Application Layer services.

2.3 PURPOSE AND OPERATION OF SOIS ELECTRONIC DATA SHEETS

A SEDS is intended to be a machine-understandable mechanism for describing onboard components, as more fully described in the SOIS Green Book (reference [D2]).

The SEDS is intended to replace the traditional interface control documents and proprietary data sheets which accompany a device and are necessary to determine the operation of the device and how to communicate with it. The SEDS could then be used for a wide variety of purposes, whilst ensuring consistency and completeness of information:

a) generating human-readable documentation;

b) specifying interfaces to the device;

c) automatically generating software implementing the relevant parts of the onboard software for the device;

d) automatically generating device-interface-simulation software for use in test or device-simulation software;

e) transforming the device functional interface into telecommands and telemetry suitable for processing by a command and data handling system onboard and on the ground;

f) capturing interface information for the spacecraft database.

Further information on the potential uses of SEDS can be found in the SOIS Green Book (reference [D2]).

In order to be able to relate the elements of the data sheet to physical (and nonphysical) concepts, and to promote standardization and interoperability, a SANA DoT (reference [1]) provides a core ontology for data sheet authors and users. These core semantic terms effectively form part of the language that is used to write SEDS. Where the semantics provided by the SANA DoT are insufficient, a data sheet author may utilize an additional user-defined DoT, which must then be supplied with the data sheet itself. This provides a standard, flexible, and extensible mechanism for capturing the semantics of device operation in a machine-understandable form.

The SEDS schema enumerates some external standards and conventions so that these standards and conventions may be associated with data in a SEDS instance. Examples are error control check words, math operations, and encoding schemes. These enumerations will never cover the variety of such standards and conventions, so the SEDS schema allows for extension of these enumerations. The extensions needed within a project to support a local toolchain can be written into an auxiliary schema, which the SEDS schema includes. The extensions that have been identified by manufacturers for use in products that are interoperable across agencies, but have not been incorporated into the SEDS schema, are written into another auxiliary schema, which is generated from the Dictionary of Terms, and which the SEDS schema includes. (See 4.8.3 for details.)

Finally, a SEDS may contain device metadata that allows recording arbitrary values such as physical device characteristics, author, version and status, etc. No predefined meaning is attached to any of the information in this area, but it may have semantic terms attached as above.

The combination of these mechanisms allows datasheets to be both:

- defined in a fixed and published standard, allowing interoperability between tools that support that standard;
- customizable to support additional features or requirements of any particular tool or system, without compromising the above.

2.4 USE OF W3C RECOMMENDATIONS

The specification and use of SEDS makes use of a number of World-Wide Web Consortium (W3C) standards:

a) XML—The Extensible Markup Language (reference [2]) is used to mark up data sheet documents in a machine-readable manner.

b) XSD—The XML Schema Definition language (references [3] and [4]) is used to specify valid construction rules for data sheet documents. It should be noted that version 1.1 of the XSD recommendation is used.

c) XInclude—To permit the construction of data sheet documents from multiple files, some of which may represent standardized data sheet elements, support for the XML Inclusions recommendation (reference [5]) is expected of applications processing SEDS documents.

3 BASIC STRUCTURE OF THE SEDS/XML SCHEMA

3.1 OVERVIEW

This section describes the structure of an electronic data sheet.

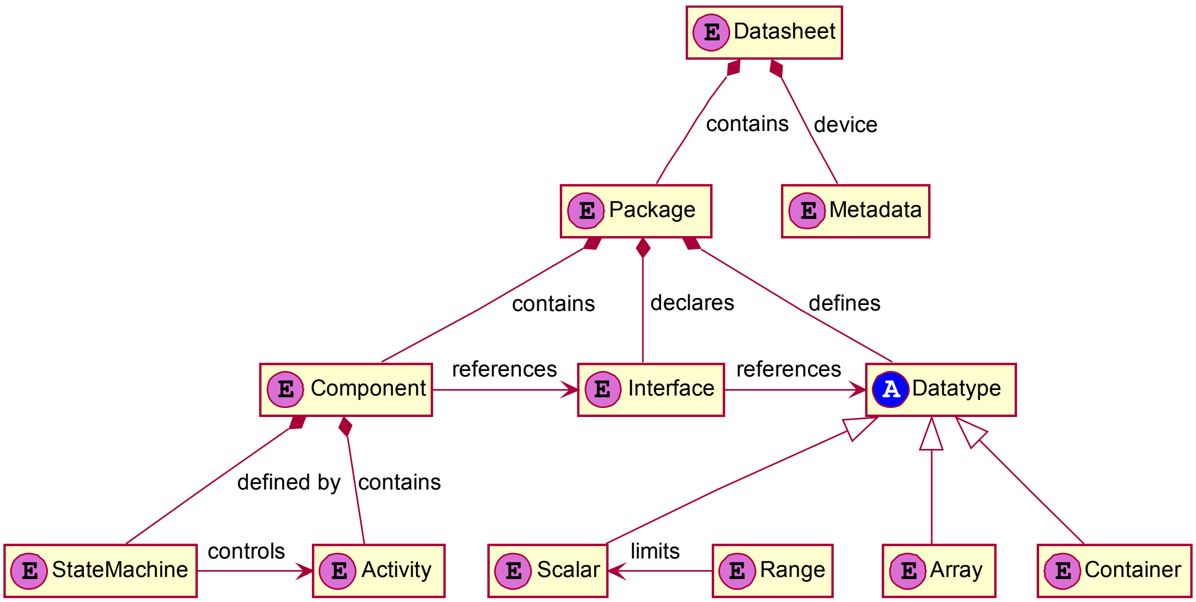

Figure 3-1: Overview of Selected Key Elements and Abstract Types of a Datasheet

Following this overview, subsection 3.2 covers the nature and relation of the various XML and XSD files that make up a datasheet. Within those files, a `Datasheet` (3.3) contains device `Metadata` (3.4) plus multiple `Packages` (3.5), which

- define a variety of `Data Types` (3.6);
- declare `Interfaces` (3.12) referencing those types;
- contain `Components` (3.13) that specify a behavioural mapping (3.14) between those interfaces.

In turn, components

- are defined by a set of `State Machines` (3.16);
- that control the execution of a set of `Activities` (3.15).

The data types supported are

- single-valued `Scalar Data Types` (3.7), which can be limited by `Ranges` (3.8);
- `Arrays` (3.9), which contain repeated similar elements;
- `Containers` (3.10) which contain a sequence of named `Fields` (3.11);

In addition to the rules laid out by the schema, further patterns must be followed in constructing a data sheet document (a schema instance) to ensure that the data sheet is logically and functionally consistent. These additional rules are described in section 4.

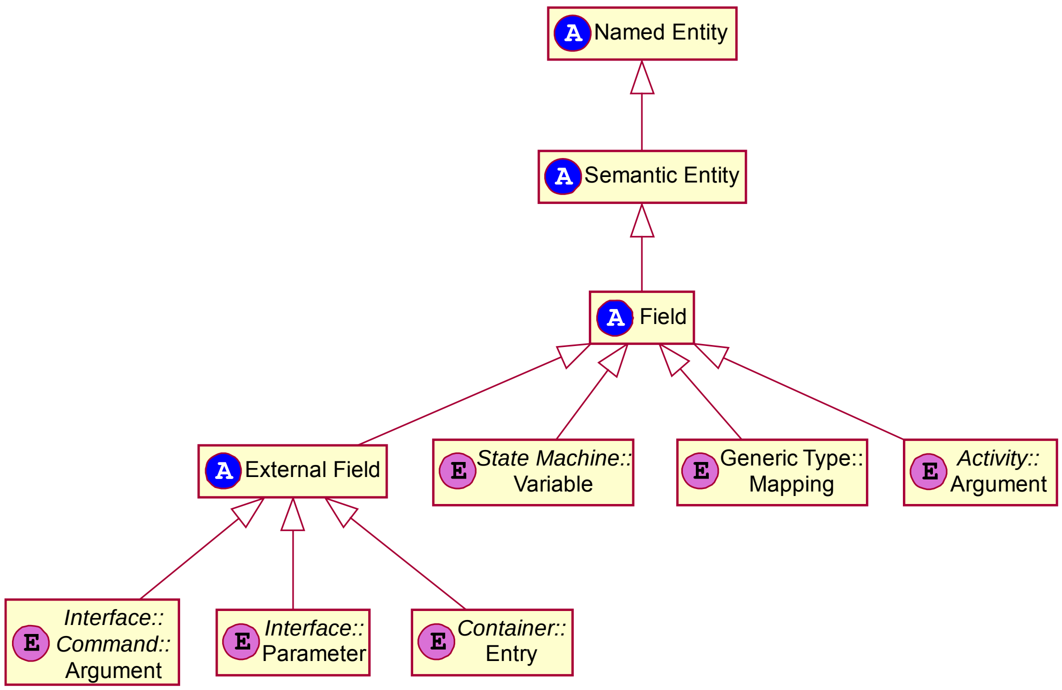

Figure 3-2: Key Inheritance Relationships between Elements and Abstract Schema Types

The above diagram shows that:

- A `Semantic Entity` (3.3) is a `Named Entity` (3.3).
- A `Field` (3.11) is a `Semantic Entity` (3.3).
- The following are all `Fields`:
  - `Variables` of a `State Machine` (3.16);
  - `Mappings` of a `Generic Type` (3.13);
  - `Arguments` of an `Activity` (3.15).
- An `External Field` (3.11) is also a `Field`.
- The following are all `External Fields`:
  - `Parameters` on an `Interface` (3.12);
  - `Arguments` to a `Command` on an `Interface` (3.12);
  - `Entries` within a `Container` (3.10).

3.2 ELECTRONIC DATA SHEETS AND THE ASSOCIATED SCHEMA

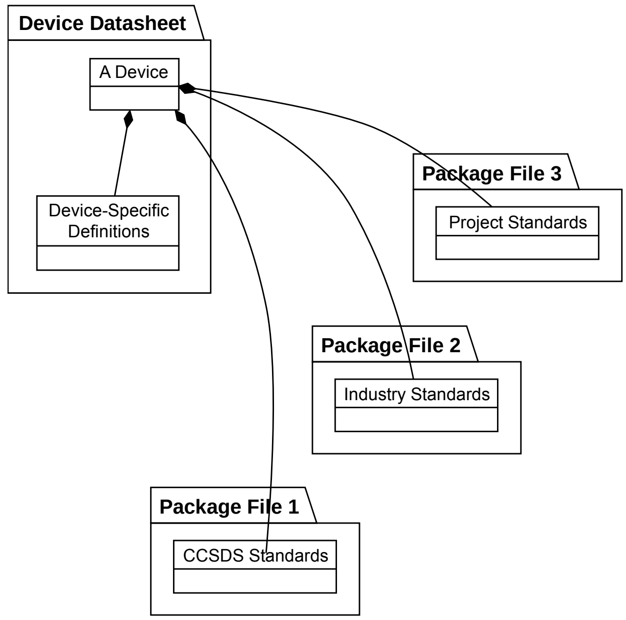

Figure 3-3: Device Datasheets and Package Files

Figure 3-3 shows the relationship among SEDS files. (It is not a syntax diagram.) A `Device Datasheet` contains all known information about a particular device or class of devices. It may reference one or more `Package Files` that capture an independently managed subset of that data, such as:

– a standard or specification it supports;

– a product line it belongs to;

– a compatibility mode it is capable of;

– a replaceable hardware or software part it contains or manages.

A `Package File` may describe a composable unit of software or hardware, in the manner of a `Device Datasheet`, but without the `Device` element, and without XInclude (reference [5]). A `Package File` may contain only metadata or data types for a platform, for a project, or for universal reference.

3.2.1 The basic unit of data exchange of SOIS device information is an XML document known as a Device Datasheet or Package File.

3.2.2 A device datasheet or package file shall be de defined by a single top-level XML file.

3.2.3 Any files referenced by a device datasheet shall be XML Package Files compliant to the `PackageFile` element of the SEDS schema.

3.2.4 When a Package File is used by a datasheet, XInclude (reference [5]) may be used to incorporate the `Package` element of that file into a single logical document compliant to the `Datasheet` element of the SEDS schema.

3.2.5 A Package File shall be a single standalone XML file without any use of XInclude.

NOTES

1 The additional constraints listed in section 4 apply only to complete documents.

2 The SEDS schema is available on the Internet-accessible CCSDS SANA registry (reference [14]).

3 Schemas that are referenced by the SEDS schema, and thus form part of the SEDS schema, will also be publicly available on the same CCSDS resource such that they may be located using the reference information in the SEDS schema. This includes the schema which corresponds to the standard Dictionary of Terms (reference [1]).

3.2.6 A SEDS document can make reference to one or more user-defined dictionaries of terms. In this case, the actual schema reference from the datasheet will be to a schema which is an extension of the SEDS schema.

NOTE – The process of generating such an extended schema is defined in reference [1].

3.3 SEDS/XML BASIC STRUCTURE

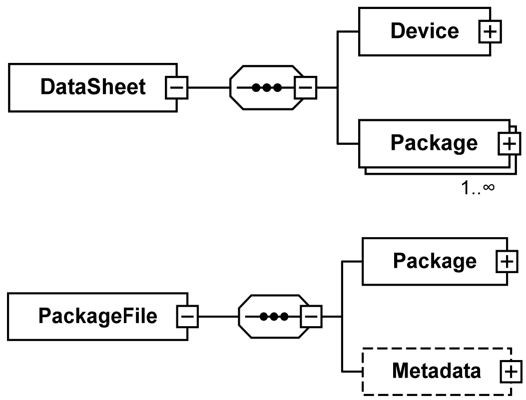

Figure 3-4: Datasheets and Package Files

NOTE – The purpose of the basic structure of a SEDS instance is to indicate whether a device is being described with XInclude capability, or a software package is being described.

3.3.1 The root element of a SEDS document shall be one of the `DataSheet` and `PackageFile` elements.

3.3.2 The `DataSheet` element shall contain exactly one `Device` element.

NOTE – This element represents the device which is being described by the SEDS document. In the diagram notation above, this is shown by the `DataSheet` element having a child element `Device`.

3.3.3 The `DataSheet` element shall contain one or more `Package` elements.

3.3.4 The `PackageFile` element shall contain exactly one `Package` element.

3.3.5 The `Device` and `PackageFile` elements shall contain one or more `MetaData` elements (see 3.4).

3.3.6 Where any SEDS element is based on the `NamedEntityType`, the element shall have a `name` attribute and may have the optional `shortDescription` attribute and `LongDescription` child element.

NOTES

1 The attribute `name` is restricted by this regular expression [a-zA-Z][a-zA-Z0-9_]* in `NameType`.

2 All named elements of a data sheet use this mechanism to allow summaries and full descriptions to be provided at any level.

3 To keep the diagrams simple, attributes are not shown, only elements.

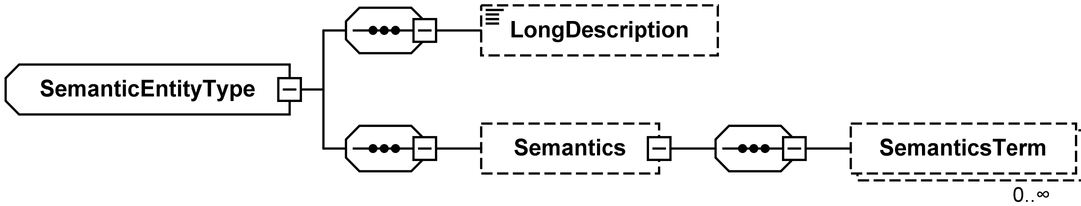

Figure 3-5: Semantics and Terms

3.3.7 Where any SEDS element is based on the `SemanticEntityType`, that element is required to be based on the `NamedEntityType` (see 3.3.3), and may have an optional `Semantics` child element.

NOTES

1 In the above diagram, the box style of the `SemanticEntityType` indicates that this is not an XML element in itself, but an XML schema-type definition that can apply to one or more elements.

2 These XML schema types use inheritance to share attribute and element definitions. So 'an element based on the `SemanticEntityType`' means 'any element with an XML schema type derived from `SemanticEntityType`'.

3.3.8 A `Semantics` element may carry a number of attributes specified by the standard dictionary of terms (reference [1]).

NOTE – This includes attributes for units, coordinate reference frames, etc.

3.3.9 A `Semantics` element shall contain zero or more `SemanticTerm` elements which associate this data type with a term in an accompanying OWL/RDF user-defined dictionary of terms using a URI.

3.3.10 The `Semantics` element shall carry zero or one `prefix` attribute which specify a URI to be used as a prefix on the URIs of all enclosed `Term` elements.

3.3.11 The `Device` element shall be based on the `SemanticEntityType`.

3.4 METADATA

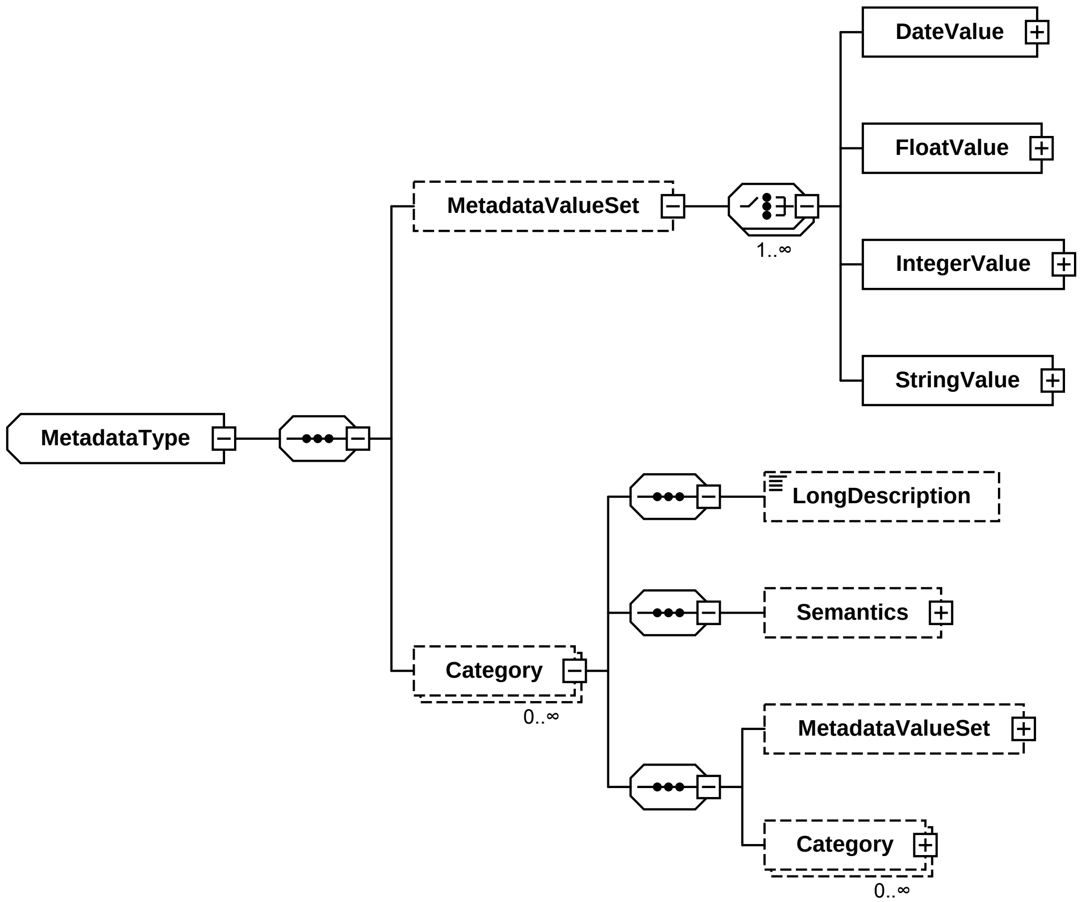

Figure 3-6: Metadata Element

NOTE – Metadata is used for purposes that include the following:

- configuration management information for the SEDS instance in which it appears;
- manufacturers information, including manufacturer name, device model, and serial number;
- model of operation of device to enable semantic reference to subjects of data;
- parameters of mission and platform that are configurable at design time.

3.4.1 A `Metadata` element shall specify a hierarchical set of categories of constant data values, each of which can be associated with machine-understandable semantics.

3.4.2 A `Category` element shall specify a categorization or grouping of metadata.

3.4.3 The `Category` element is based on `NamedEntityType` (see 3.3.3).

3.4.4 The `Category` element shall contain zero or one `Semantics` element and one or more further child elements, each of which is either a `Category` element or `MetadataValueSet` element.

3.4.5 A `MetadataValueSet` element shall contain one or more child elements, each of which is either a `DateValue` element, a `FloatValue` element, an `IntegerValue` element, or a `StringValue` element.

3.4.6 The `DateValue`, `FloatValue`, `IntegerValue`, and `StringValue` elements are all based on `FieldType`.

3.4.7 `DateValue` and `StringValue` elements shall contain a `value` attribute specifying the value of the metadata as a literal, per table 3-1.

3.4.8 `FloatValue` and `IntegerValue` elements may contain a `value` attribute specifying the value of the metadata as a literal, per table 3-1.

3.4.9 If a `FloatValue` or `IntegerValue` element does not contain a value attribute, the body of the element shall specify a `MathOperation` element, as described in 3.15.32 below, or a `Conditional` element, as described in 3.15.37 below, to describe how the value should be calculated.

3.5 PACKAGES

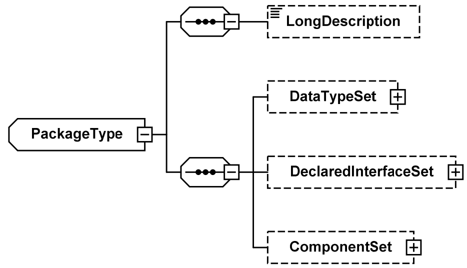

Figure 3-7: Package Element

NOTE – Packages describe software components, types of data, and interfaces.

3.5.1 The name of each `Package` element declared shall be unique within the datasheet.

3.5.2 A `Package` name may be hierarchical, in which case it shall consist of multiple name segments separated by the slash character ('/').

NOTES

1 This permits the declaration of hierarchical packages.

2 A hierarchical name is separated by the slash character which is enforced by a pattern. Hierarchical names are used to avoid accidental name conflicts between the names of packages; there is no special relationship between packages implied by their position in the hierarchy, and no special syntax for accessing the elements defined with package A from a package A/B.

3.5.3 A package may have a `shortDescription` attribute and a `LongDescription` child element.

NOTES

1 This is not done using `NameDescriptionType`, as the name attribute needs to be of type `QualifiedNameType`, not `NameType`.

2 Subsection 4.3 provides details of referencing types or interfaces within a package from other elements.

3.5.4 A `Package` element may contain the following optional elements, in the following order:

a) `DataTypeSet`;

b) `DeclaredInterfaceSet`;

c) `ComponentSet`.

3.6 DATA TYPES

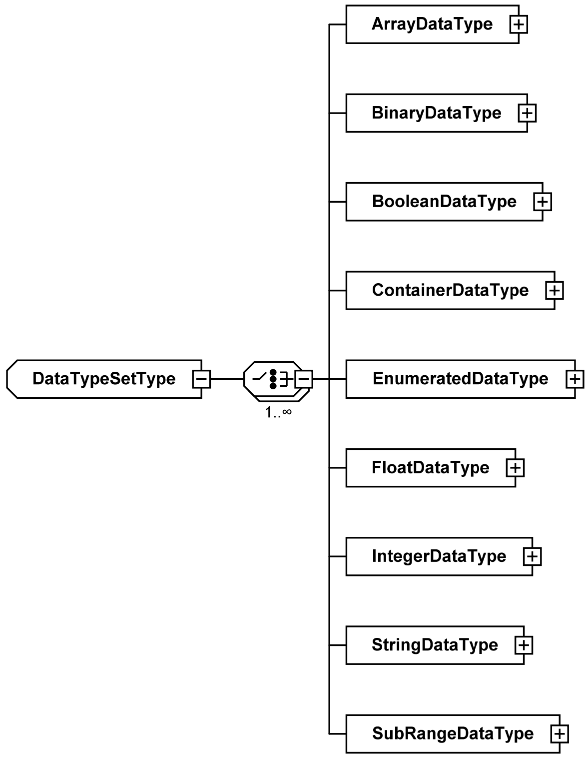

Figure 3-8: Data Types within a DataTypeSet Element

NOTE – Data types describe the syntax and semantics of units of data that flow through interfaces.

3.6.1 The `DataTypeSet` element contained in a package or component shall contain one or more of the following elements: `ArrayDataType`, `BinaryDataType`, `BooleanDataType`, `ContainerDataType`, `EnumeratedDataType`, `FloatDataType`, `IntegerDataType`, `StringDataType`, and `SubRangeDataType`.

3.6.2 Each child element of a `DataTypeSet` element is based on the `FieldType` (see 3.3.7).

3.6.3 The name of each child element of a `DataTypeSet` element shall be unique within the containing package.

NOTE – This forbids the definition of types with duplicate names inside components within the same package.

3.7 SCALAR DATA TYPES

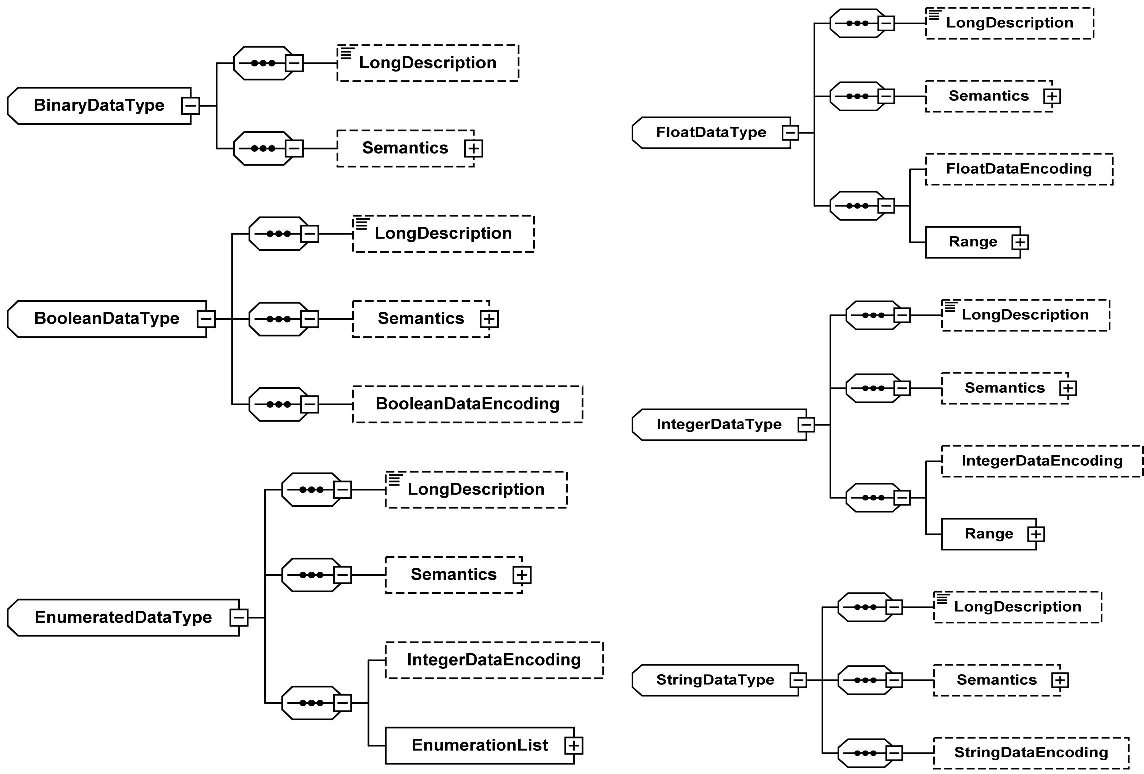

Figure 3-9: Scalar Data Types

NOTES

1 A 'scalar' data type is a single-valued data type, as opposed to structured types like arrays or containers.

2 A 'numeric' data type is an `IntegerDataType` or a `FloatDataType`.

3 Scalar types can specify how they are to be encoded. This information is used when they are transmitted over a subnetwork.

4 All encoding specifications should be considered as complete and standalone, with no inheritance mechanism.

5 Numeric scalar data types can specify a range of representable values, which form a constraint on the possible encodings. (See 3.8 for details.)

6 Scalar types can have values specified in a datasheet as literals (see table 3-1).

3.7.1 Each `BooleanDataType`, `EnumeratedDataType`, `FloatDataType`, `IntegerDataType`, `StringDataType`, or `SubRangeDataType` element may contain an optional encoding element of a type corresponding to table 3-1.

Table 3-1: Data Types, Encodings, Ranges and Literals

Data Type	Encoding Type	Range Types	Literal Syntax
`BinaryDataType`			`xs:hexBinary`
`BooleanDataType`	`BooleanDataEncoding`		`xs:boolean`
`EnumeratedDataType`	`IntegerDataEncoding`	`EnumeratedRange`	`xs:string`, matching enumeration label.
`FloatDataType`	`FloatDataEncoding`	`PrecisionRange` `MinMaxRange`	`xs:float`
`IntegerDataType`	`IntegerDataEncoding`	`MinMaxRange`	`xs:integer`
`StringDataType`	`StringDataEncoding`		`xs:string`
`SubRangeDataType`		as base type	as base type, within range

3.7.2 A `FloatDataEncoding` or `IntegerDataEncoding` element may carry a `byteOrder` attribute specifying a value of:

a) `bigEndian`, the default, for values which are to be encoded most significant byte first;

b) `littleEndian` for values which are to be encoded least significant byte first.

NOTE – The `littleEndian` specification applies only to data types whose size is a multiple of 8 bits.

3.7.3 A `BooleanDataEncoding` element must carry a `sizeInBits` attribute which specifies the size, in bits, of the encoded data as a positive integer.

3.7.4 A `BooleanDataEncoding` element may carry a `falseValue` attribute which specifies the value that corresponds to logical falsehood, with options:

3.7.5 `zeroIsFalse` (the default);

3.7.6 `nonZeroIsFalse`.

3.7.7 An `IntegerDataEncoding` element must carry an `encoding` attribute which has a value of:

a) `unsigned`, for an unsigned value;

b) `signMagnitude`, for an encoding with a separate sign bit (the most significant bit is the sign bit, with 1 indicating negative);

c) `twosComplement`, for twos complement;

d) `onesComplement`, for ones complement;

e) `BCD`, for a natural unsigned binary coded decimal, where each byte is a decimal digit encoded as binary;

f) `packedBCD`, where each byte contains two decimal digits encoded as binary, followed by an optional sign nibble. A negative sign is 1011 or 1101; a positive sign is 1010, 1100, 1110, 1111, or omitted.

3.7.8 An `IntegerDataEncoding` element must carry a `sizeInBits` attribute which specifies the size, in bits, of the encoded data as a positive integer.

NOTE – The size in bits of a BCD encoding must be a multiple of 8. The size in bits of a packedBCD must be a multiple of 4. The size in bits of both forms of binary coded decimals is a fixed value, so all high-order digits that are zero must be present to fill the fixed size in bits.

3.7.9 A `FloatDataEncoding` element must carry an `encodingAndPrecision` attribute which has a value of either:

a) `IEEE754_2008_single`;

b) `IEEE754_2008_double`;

c) `IEEE754_2008_quad`;

d) `MILSTD_1750A_simple`;

e) `MILSTD_1750A_extended`.

NOTE – These represent the supported sizes of IEEE (reference [6]) and MIL-STD-1750A (reference [7]).

3.7.10 A `FloatDataEncoding` element must carry a `sizeInBits` attribute which specifies the size, in bits, of the encoded data as a positive integer.

3.7.11 A `StringDataType` must carry a `length` attribute which defines the maximum possible length of the string, in bytes.

3.7.12 A `StringDataType` may carry a `fixedLength` attribute which, if ‘false’, indicates that the string can be shorter than the value specified by the length attribute.

NOTE – Specification of `fixedLength="false"` indicates a data type that occupies a variable amount of memory. When such a data type is an entry in a container, then the container is of variable length. (See the note after 4.8.2 for details about string lengths.)

3.7.13 A `StringDataEncoding` element may carry an `encoding` attribute which has a value of either:

a) `UTF-8` specifying Unicode UTF-8 encoding (reference [8]); or

b) `ASCII`, the default, specifying US ASCII encoding (reference [9]).

3.7.14 The optional `terminationCharacter` attribute of a `StringDataEncoding` element shall specify the termination character for the string.

NOTES

1 For example, a termination character of zero (null) is used by C-language strings.

2 Bytes used by a termination character are not included in the count of bytes that constitute the length of the string.

3 UTF-8 characters use variable-length encoding that can contain as much as 4 bytes per character. Consequently, not all UTF-8 strings of a given character length are representable by a data type with that byte length.

3.7.15 An EnumeratedDataType shall contain an EnumerationList element, consisting of a list of one or more Enumeration elements.

3.7.16 Each `Enumeration` element has required `label` and `value` attributes, indicating the integer value corresponding to a given label.

3.7.17 An `Enumeration` element may have an optional child `Semantics` element.

3.8 RANGES

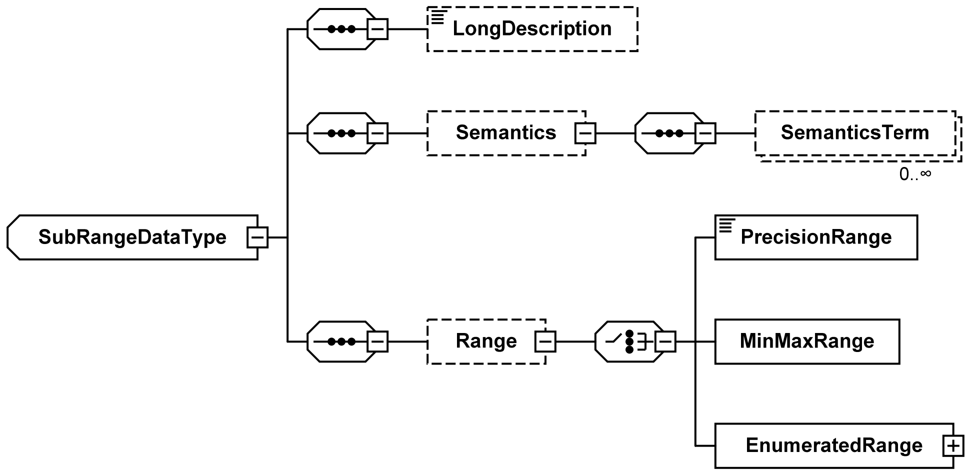

Figure 3-10: Ranges within a SubRangeDataType Element

NOTE – A range constrains the values of a data type for the purposes of validation and recognition.

3.8.1 Each `EnumeratedDataType`, `FloatDataType`, `IntegerDataType`, `EnumeratedDataType`, or `SubRangeDataType` element shall contain a single `Range` element of a type corresponding to table 3-1.

3.8.2 A `SubRangeDataType` element shall contain a `baseType` attribute, referring to the numeric or enumerated scalar type which defines all properties other than range.

3.8.3 A `PrecisionRange` element shall be either `SINGLE`, `DOUBLE`, or `QUAD`, representing the full supported representation range of the corresponding IEEE754 floating point data encodings.

3.8.4 A `MinMaxRange` element shall have an attribute `rangeType`, one of the options listed in table 3-2.

Table 3-2: MinMaxRange Options

Interval Notation	Relational Notation	XML Notation `rangeType`	`min`	`max`
`(a..b)`	{x | a < x < b}	exclusiveMinExclusiveMax	yes	yes
`[a..b]`	{x | a <= x <= b}	inclusiveMinInclusiveMax	yes	yes
`[a..b)`	{x | a <= x < b}	inclusiveMinExclusiveMax	yes	yes
`(a..b]`	{x | a < x <= b}	exclusiveMinInclusiveMax	yes	yes
`(a..+∞)`	{x | a < x}	greaterThan	yes	
`[a..+∞)`	{x | a <= x}	atLeast	yes	
`(-∞..b)`	{x | x < b}	lessThan		yes
`(-∞..b]`	{x | x <= b}	atMost		yes

3.8.5 A `MinMaxRange` element may have attributes `min` and `max`, whose presence and values shall be consistent with table 3-2.

3.8.6 An `EnumeratedRange` element must have a list of `Label` child elements, with values that must be enumeration labels of the corresponding `EnumeratedDataType`.

3.9 ARRAYS

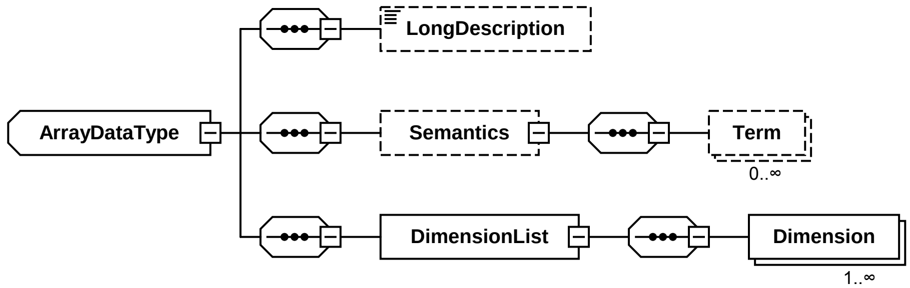

Figure 3-11: ArrayDataType and Dimension Elements

NOTE – Arrays provide the means to specify adjacent repetitions of the same type of data, the elements of which can be accessed by index.

3.9.1 An `ArrayDataType` element shall contain a `dataTypeRef` attribute, referring to the type of the elements within the array.

NOTE – Subsection 4.3 provides further details.

3.9.2 An `ArrayDataType` element shall contain a `DimensionList` element with one or more `Dimension` child elements.

3.9.3 A `Dimension` element determines the length of the array dimension, in elements, and shall either have attribute `size`, directly indicating the maximum length, or attribute `indexTypeRef`, indicating the integer data type to be used to index the array. The type referenced by an `indexTypeRef` attribute has maximum and minimum legal values, so the size of the array is 1 plus the maximum legal value minus the minimum legal value. When the `size` attribute is used, the index is zero-based; when the `indexTypeRef` is used, the index of the first element of the array is indexed by the minimum legal value of the index type.

3.9.4 An array having multiple `Dimension` elements shall use only the first `Dimension` element in its definition, with the remaining dimensions specified in the element type.

NOTES

1 The maximum length of an array with a specified index type can be found by looking at the maximum value of the index range (e.g., an array with index type ‘8 bit unsigned’ has a maximum size of 255).

2 There is currently no foreseen need to specify any array encoding other than the default behaviour of encoding array elements contiguously. Any required padding (e.g., for alignment purposes) can be added to the referenced array element type.

3 Variable-length arrays, i.e., lists, are only supported within the context of a container. (For details, see 3.10.)

3.10 CONTAINERS

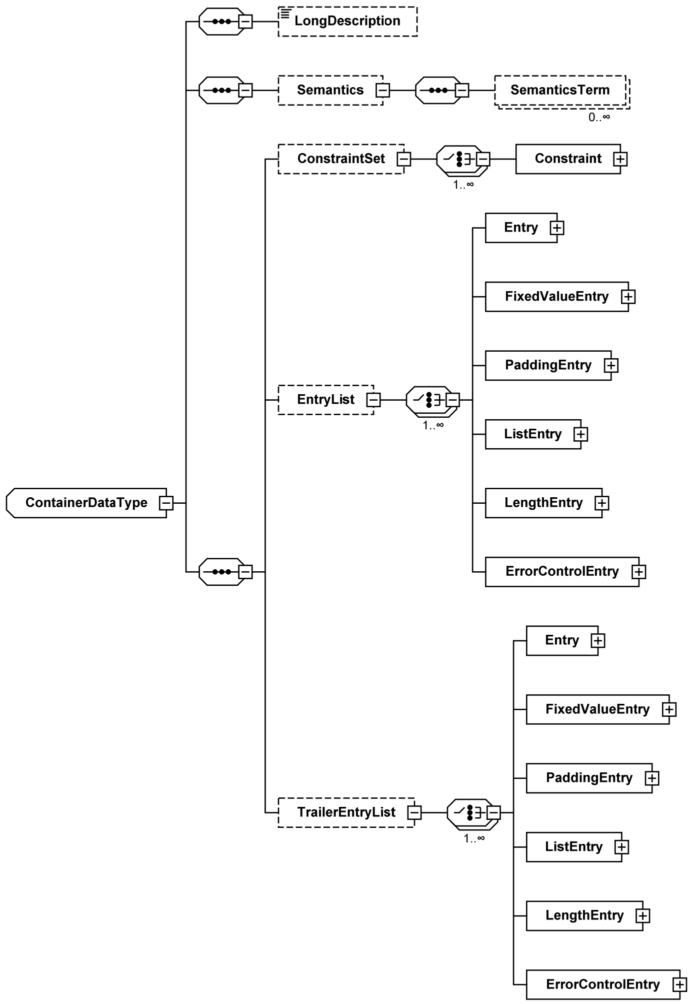

Figure 3-12: Constraints and Entries of a ContainerDataType Element

NOTES

1 Containers are aggregate data types with named entries;each can be of any type.

2 There is a concept of a 'container encoding' as explained in 4.7. For a container that describes a protocol data unit, the encoding determines the arrangement and content of the entries of the container in the protocol data unit. Any container can specify encoding information; a toolchain should ignore this where not applicable. The following information specifies the encoding of a container:

– padding entries;

– bit-width of entries;

– encoding of entries.

3 When used for multi-level self-identifying structured data (e.g., CCSDS space packets) containers are organized in a hierarchy where the concrete container representing a specific packet has a base type of the container representing the packet header and trailer.

4 Constraints on header fields represent an implicit algorithm for classifying raw packet data step by step until the final concrete container is selected. (See 4.7 for details.)

5 The constraints in a constraint set are combined by conjunction (AND).

3.10.1 A `ContainerDataType` element may carry an optional `abstract` attribute which, if set to ‘true’, indicates that the container is not to be used directly, only referenced as the base type of other containers.

3.10.2 A `ContainerDataType` element may carry an optional `baseType` attribute which indicates that the container is a constrained extension of another.

3.10.3 A `ContainerDataType` element shall include zero or one `ConstraintSet` element and zero or one `EntryList` element.

3.10.4 An abstract `ContainerDataType` element may include zero or one `TrailerEntryList` element.

3.10.5 The `ConstraintSet` element of a `ContainerDataType` element specifies the criteria which apply to the entries of the container type which is the base type of this container in order for the type to be valid.

3.10.6 The `ConstraintSet` element of a `ContainerDataType` element shall contain one or more child elements, which can be one of a `RangeConstraint`, a `TypeConstraint`, or a `ValueConstraint`.

3.10.7 Each child entry of a `ConstraintSet` shall have an attribute `entry`, which names the entry that the constraint applies to. This entry must exist within a base container reachable by a recursive chain of base container references from the current container.

3.10.8 A `RangeConstraint` element shall carry a child element of any type of range legal for the type of the constrained entry (see table 3-1).

3.10.9 A `TypeConstraint` element shall have an attribute `type`, which shall reference a numeric type which has a range included in the type of the constrained entry.

3.10.10 A `ValueConstraint` element shall have an attribute `value`, which shall contain a literal value of a type corresponding to the type of the constrained entry.

3.10.11 The `EntryList` and `TrailerEntryList` elements of a `ContainerDataType` element shall contain one or more `Entry`, `FixedValueEntry`, `PaddingEntry`, `ListEntry`, `LengthEntry`, and `ErrorControlEntry` child elements.

3.10.12 The first entry in an `EntryList` is located at a bit offset immediately following the last entry of the `EntryList` of any base container, or offset 0 if no such container exists.

3.10.13 For an abstract packet, the first entry in a `TrailerEntryList` is located at a bit offset immediately following all entries of the derived container.

3.10.14 Each other entry in an `EntryList` or `TrailerEntryList` is located at a bit offset immediately following the previous entry.

3.10.15 Each `Entry`, `FixedValueEntry`, `ListEntry`, `LengthEntry`, and `ErrorControlEntry` element shall have the attributes and child elements associated with an external field (see 3.11).

3.10.16 Each `Entry`, `FixedValueEntry`, `ListEntry`, `LengthEntry`, and `ErrorControlEntry` element within a container shall have a name that is unique within that container, plus all containers recursively referenced by its `baseType` attribute.

3.10.17 A `FixedValueEntry` entry shall have a `fixedValue` attribute which specifies the value to which the container entry should be fixed.

NOTE – The container entry therefore has a constant value and is effectively read-only. This is used for report IDs, command codes, etc.

3.10.18 If the `fixedValue` attribute is used to specify the value for an entry; the value shall be a literal whose type matches the type of the entry according to table 3-1.

3.10.19 A `PaddingEntry` element within a container shall have an attribute `sizeInBits`, which is used to specify the position of successive fields.

3.10.20 A `ListEntry` element within a container shall specify an attribute `listLengthField` which contains the name of another element of the same container whose value will be used to determine the number of times this entry should be repeated.

3.10.21 A `LengthEntry` element within a container shall specify an entry whose value is constrained, or derived, based on the length of the container in which it is present.

3.10.22 If a `LengthEntry` element has a calibration (see 3.11.7), that calibration shall be used to map between the length in bytes of the container and the value of the entry, according to the formula:

`container length in bytes = calibration(entry raw value).`

NOTE – A constraint that refers to a `LengthEntry` compares to the entry raw value.

3.10.23 Any calibration specified for a `LengthEntry` shall be *reversible*, i.e., a linear polynomial, or spline with all points of degree 1.

3.10.24 An `ErrorControlEntry` element within a container shall specify an entry whose value is constrained, or derived, based on the contents of the container in which it is present. As well as a subset of the attributes and elements supported for a regular container entry, it has the mandatory attribute type, which is one of the values specified in the Dictionary of Terms for `errorControlType` as illustrated in table 3-3.

Table 3-3: Error Control Types

Value	Description	Reference
CRC16_CCITT	G(X) = X^16 + X^12 + X^5 + 1	[10], subsection 4.1.4.2
CRC8	G(x) = x^8 + x^2 + x^1 + x^0	[11], clause 5.2
CHECKSUM	modulo 2^32 addition of all 4-byte	[12], subsection 4.1.2
CHECKSUM_LONGITUDINAL	Longitudinal redundancy check, bitwise XOR of all bytes	[13]

3.11 FIELDS

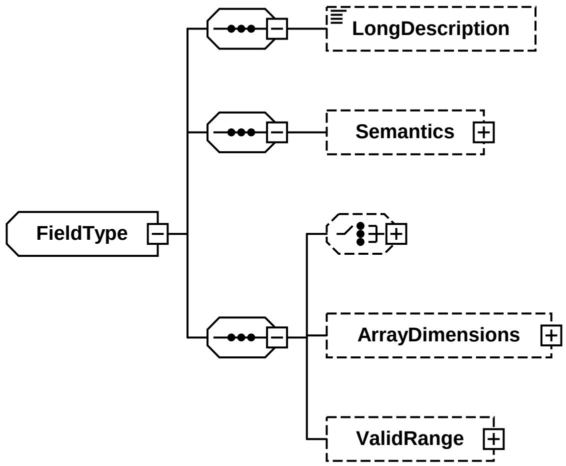

Figure 3-13: Field Schema Type

NOTES

1 Data types are instantiated in many different circumstances; however, whenever a data type is instantiated there is a set of common valid attributes and elements. This is referred to as a 'field'. This subsection describes these attributes and elements such that they may be referenced whenever a data type instantiation is described elsewhere in this document.

2 The concept of 'field' permits the definition of parameter, arguments, or container entries that are a subrange, encoding or array of the referenced type; this definition of an anonymous type avoids the need to define artificial explicit named types.

3 An extension of the field concept is the External Field schema type, which is used in cases where the value of the field may appear on an interface (i.e., parameters, command arguments, and container entries).

3.11.1 A field is based on the `SemanticEntityType` (see 3.3.7).

3.11.2 A field shall carry a `type` attribute identifying the name of the data type it is based on.

NOTE – Subsection 4.3 provides further details.

3.11.3 A field shall have zero or one `ValidRange` element, zero or one `ArrayDimensions` element, and zero or one encoding element.

3.11.4 Any `ValidRange` and encoding child elements of a field shall match the `type` attribute according to table 3-1.

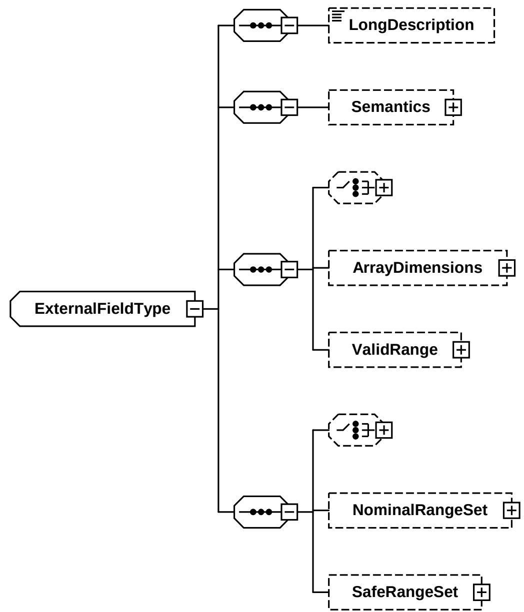

Figure 3-14: External Field Schema Type

3.11.5 A `NominalRangeSet` child element of an external field specifies the nominal operating limits of the field. It has the same child elements and attributes as other `Range` elements (see 3.8).

3.11.6 A `SafeRangeSet` child element of an external field specifies the safe operating limits of the field. It has the same child elements and attributes as other `Range` elements (see 3.8).

3.11.7 A `SplineCalibrator` or `PolynomialCalibrator` child element of an external field specifies any calibration that would be required to take the raw value represented by the data type and convert it into the units and other semantic terms associated with the field (see 3.15).

3.12 INTERFACES

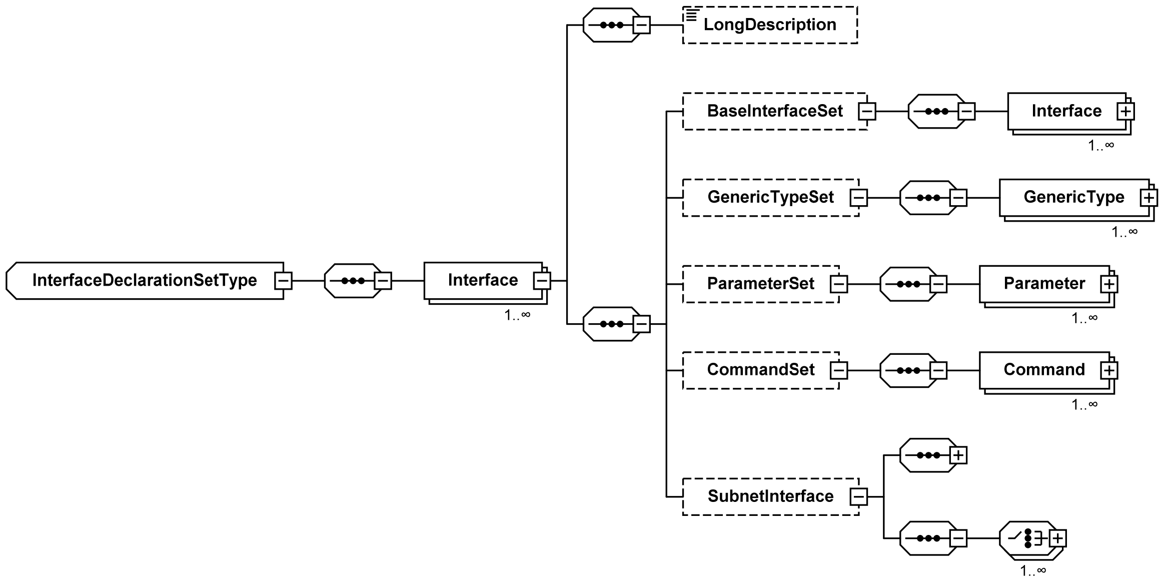

Figure 3-15: Interfaces within an InterfaceDeclarationSet Element

NOTES

1 Standardized interfaces, including those to the subnetwork, are defined with this interface construct to allow them to be treated symmetrically with user-defined interfaces.

2 Any interface declared within a data sheet is implicitly scoped to the data sheet, bypassing any complexities associated with having multiple devices.

3 An interface can be defined in terms of generic types, to avoid placing undue restrictions on its implementation or use. Such interfaces need to have those generic types mapped to fully specified types before use (see 3.12.7).

3.12.1 An `InterfaceDeclarationSet` element shall contain one or more `Interface` elements.

3.12.2 Each `Interface` child element of an `InterfaceDeclarationSet` is based on the `NameDescriptionType` (see 3.3.3).

3.12.3 The name of each `Interface` child element of an `InterfaceDeclarationSet` element shall be unique.

3.12.4 An `Interface` element shall contain zero or one `BaseInterfaceSet` element referencing one or more `Interface` elements, zero or one `GenericTypeSet` element containing one or more `GenericType` elements, zero or one `ParameterSet` element containing one or more `Parameter` elements, and zero or one `CommandSet` element containing one or more `Command` elements.

3.12.5 An `Interface` element may have an optional attribute `abstract`, which, if true, indicates the interface may not be used directly by a component.

3.12.6 An `Interface` element must have an attribute `level`, with value taken from table 3-4, which indicates the system level at which it operates.

Table 3-4: Interface Levels

Name	Description
Application	Not directly related to device data.
Functional	Higher-level virtual abstraction of device data.
Access	Lower-level specification of device data.
Subnetwork	Raw uninterpreted communication channel to a device.

3.12.7 Each `Interface` child element of a `BaseInterfaceSet` element shall identify one existing interface type which shall be used as a parent type for this interface type.

NOTE – This interface will therefore inherit all of the parameters and commands of each identified parent interface type (including any parameters and commands inherited from their parents, and so on). Circular inheritance (‘A has base B which has base A’) is not permitted. (See notes attached to 3.12.12 and 3.12.16.)

3.12.8 Each `GenericType` child element of a `GenericTypeSet` element specifies a generic type to be used by the interface, and is based on the `NameDescriptionType` (see 3.3.3).

3.12.9 The name of each `GenericType` child element of a `GenericTypeSet` element shall be unique.

3.12.10 Each `GenericType` element may carry a `baseType` attribute which specifies an existing type. The existing type must be a base (ancestor) type of any concrete type, which is mapped to the generic type when the interface is instantiated.

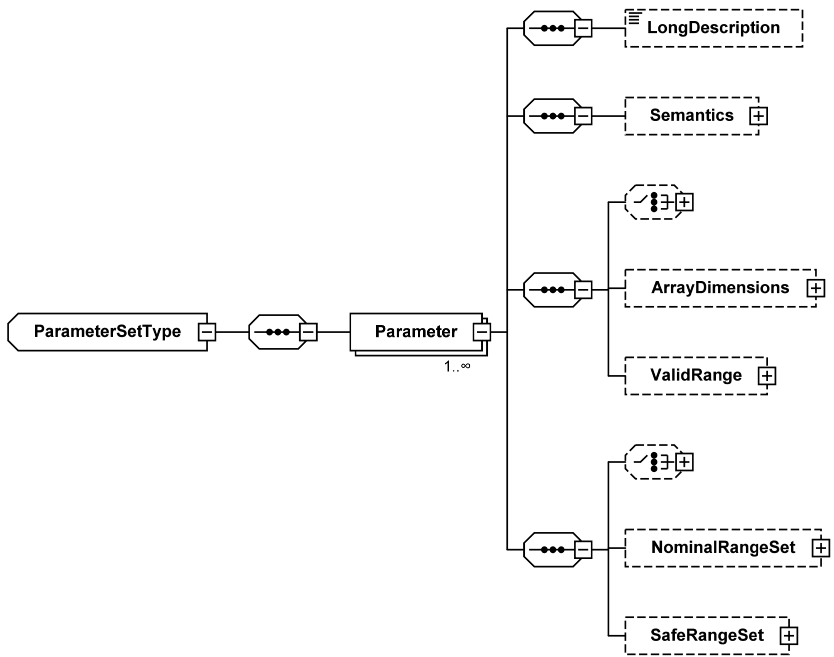

Figure 3-16: Parameters in a ParameterSet Element

3.12.11 Each `Parameter` child element of a `ParameterSet` element shall have the attributes and child elements associated with an external field (see 3.11) with the addition of an optional `mode` attribute, indicating how the parameter data is transmitted across the interface, and an optional `readOnly` attribute, identifying if the parameter is read-only.

3.12.12 The name of each `Parameter` child element of a `ParameterSet` element shall be unique within the set of interfaces reachable by `BaseType` references from the containing interface. (See the note attached to 3.12.16; a similar explanation applies to parameters.)

3.12.13 Valid values for the `mode` attribute of a `Parameter` element shall be '`sync`' (the default) or '`async`'.

NOTE – Subsection 4.5 provides further details.

3.12.14 Valid values for the `readOnly` attribute shall be '`false`' (the default) or '`true`'.

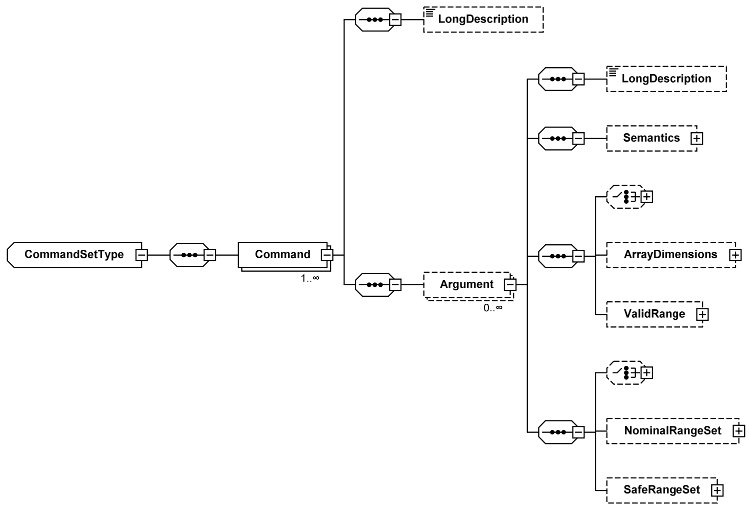

Figure 3-17: Commands in a CommandSet Element

3.12.15 Each `Command` child element of a `CommandSet` element is based on the `NameDescriptionType` (see 3.3.3), plus an optional `mode` attribute, identifying the command mode.

3.12.16 The name of each `Command` child element of a `CommandSet` element shall be unique within the set of interfaces reachable by `BaseType` references from the containing interface.

NOTE – The reason for this restriction is to eliminate the possibility of a derived interface overriding a command of a base interface. Instead, the commands of each base type interface are included unchanged in the derived interface. This is a form of aggregation of interface commands.

3.12.17 Each `Command` child element of a `CommandSet` element identifies a command on an interface and shall contain zero or more `Argument` elements, each of which identifies an argument to the command.

3.12.18 Each `Argument` child element of a `Command` element shall have the attributes and child elements associated with an external field (see 3.11) with the addition of an optional `mode` attribute, identifying the argument mode.

3.12.19 The name of each `Argument` child element of a `Command` element shall be unique within the command.

3.12.20 A command argument may have an attribute `defaultValue`, indicating the value to be used for a call to this command when not otherwise specified.

3.12.21 A command argument may have an attribute `dataUnit`, indicating it is a *service data unit* and so should be passed to/from the device without further interpretation or encoding.

NOTE – Subsection 4.7 provides further details.

Table 3-5: Interface Syntax, Primitives, and Transactions

Interface Element	Parameter/ Command Modes	Argument Modes	Primitive	Transaction
Parameter	**sync**		request indication	yes
	async		indication	no
Command	**sync**	No **out** or **inout**	request indication	yes
		inout, or both **in** and **out**	request indication	yes
		No **in** or **inout** arguments	request indication	yes
	async	No **out** or **inout** arguments	request	no
		inout, or both **in** and **out**	illegal	
		No **in** or **inout** arguments	indication	no

3.12.22 Valid values for the `mode` attribute of a command argument shall be as listed in table 3-5.

NOTE – Subsection 4.5 provides further details.

3.13 COMPONENTS

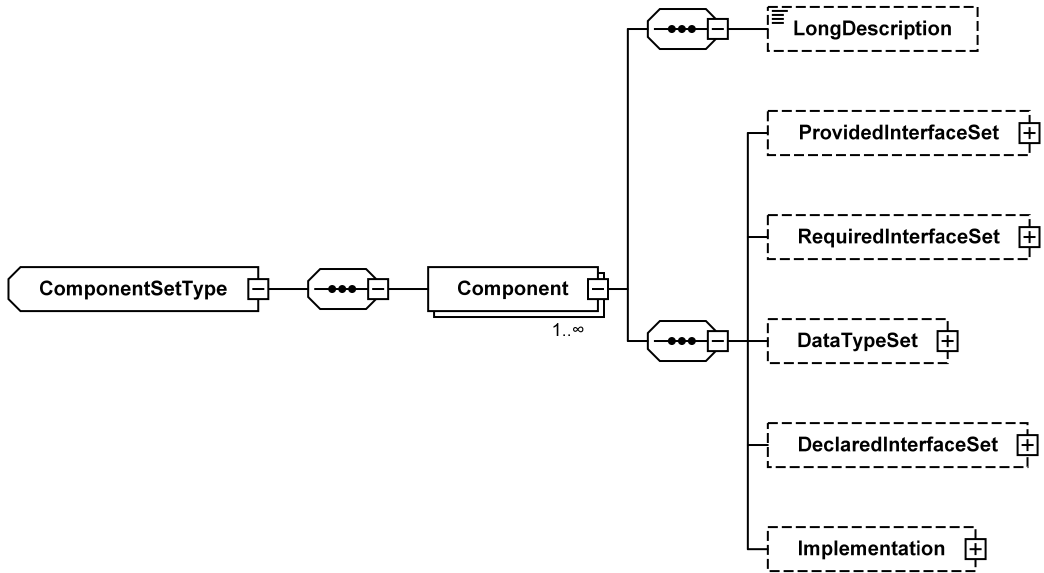

Figure 3-18: Components in a ComponentSet Element

NOTE – Components describe entities that have interfaces and computational behaviour.

3.13.1 The `ComponentSet` element contained in a package shall contain one or more `Component` elements.

3.13.2 Each `Component` child element of a `ComponentSet` element is based on the `NameDescriptionType` (see 3.3.3).

3.13.3 The name of each `Component` child element of a `ComponentSet` element shall be unique within the containing package.

3.13.4 A `Component` element shall contain, in order:

a) zero or one `ProvidedInterfaceSet` element;

b) zero or one `RequiredInterfaceSet` element;

c) zero or one `DataTypeSet` element;

d) zero or one `DeclaredInterfaceSet` element; and

e) zero or one `Implementation` element.

3.13.5 The `ProvidedInterfaceSet` and `RequiredInterfaceSet` element, if present, shall each contain one or more `Interface` elements, each of which identifies a provided or required interface, respectively.

3.13.6 Each `Interface` child element of a `ProvidedInterfaceSet` or `RequiredInterfaceSet` element is based on the `NameDescriptionType` (see 3.3.3).

3.13.7 The name of each `Interface` child element of a `ProvidedInterfaceSet` or `RequiredInterfaceSet` element shall be unique within the containing component.

3.13.8 Each `Interface` element shall carry a `type` attribute which identifies the type of the interface by referencing an element of the `DeclaredInterfaceSet` entry of a `Package` or `Component`.

NOTE – Subsection 4.3 provides further details.

3.13.9 Each `Interface` element may have a `GenericTypeMapSet` element which maps the generic types used to define the interface to the concrete types used in the current component.

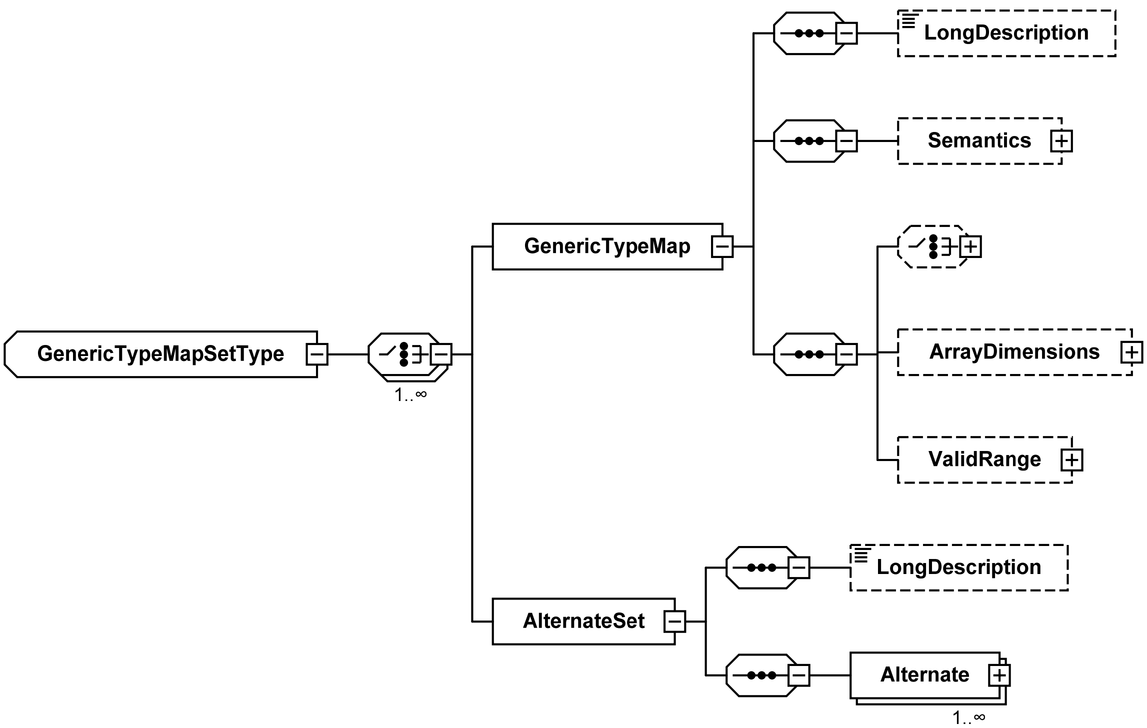

Figure 3-19: Generic Type Mapping

3.13.10 A `GenericTypeMap` element specifies a mapping of a generic type to a concrete type and shall have the attributes and child elements associated with a field (see 3.11), with the optional addition of a `fixedValue` attribute.

3.13.11 The optional `fixedValue` attribute of a `GenericTypeMap` element shall specify a fixed value for the generic type.

NOTE – This is equivalent to specifying a data type with a valid range which contains only the value specified by the `fixedValue` attribute.

3.13.12 An `AlternateSet` child element of a `GenericTypeMapSet` element, where present, specifies a set of alternative mappings of generic types to a concrete type and shall contain one or more `Alternate` elements, each of which shall contain one or more `GenericTypeMap` elements.

NOTE – This construct exists to deal with scenarios like 'if you read a value from address *x*, the result will be of type *A*, whereas if you read from address *y*, the result will be of type *B*'. (See 4.3 for further details.)

3.13.13 The optional `DataTypeSet` child element of a `Component` shall define types local to the component and cannot be referenced outside it.

3.13.14 The optional `DeclaredInterfaceSet` child element of a `Component` shall define interface declaration local to the component and cannot be referenced outside it.

NOTE – Such local interfaces are normally constrained versions of a more generic interface, such as a subnetwork service.

3.13.15 Types and interfaces declared within the `DataTypeSet` and `DeclaredInterfaceSet` child elements of a `Component` element shall only be visible to descendent elements of the `Component` element.

NOTE – Types declared as part of a component type can only be used within the component type and its associated implementation. This makes these types 'private' to the component type declaration.

3.14 COMPONENT IMPLEMENTATIONS

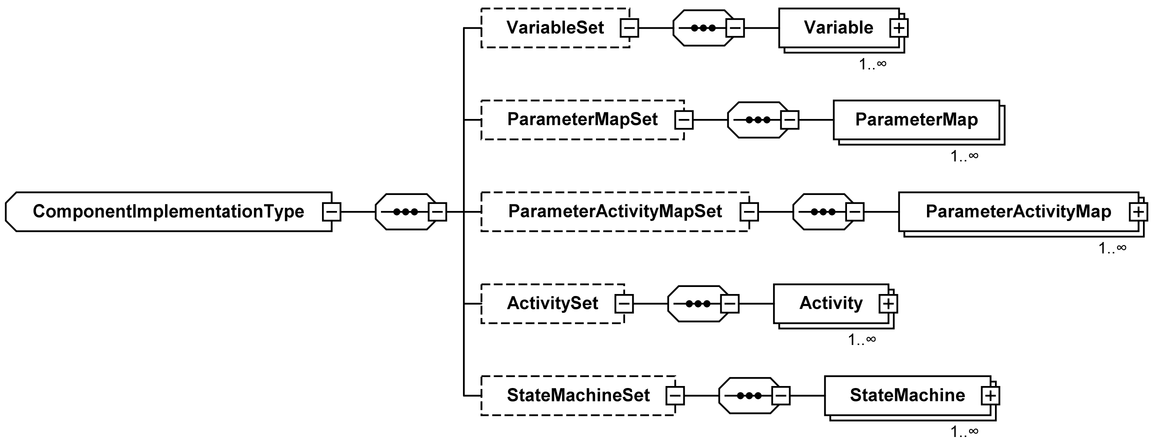

Figure 3-20: Implementation Element of a Component

NOTE – The implementation of a component specifies its behaviour.

3.14.1 The `Implementation` child element of a `Component` element shall contain zero or one of each of the following elements, in order:

a) `VariableSet`;

b) `ParameterMapSet`;

c) `ParameterActivityMapSet`;

d) `ActivitySet`;

e) `StateMachineSet`.

3.14.2 A `VariableSet` element shall contain one or more `Variable` elements. (See figure 3-21.)

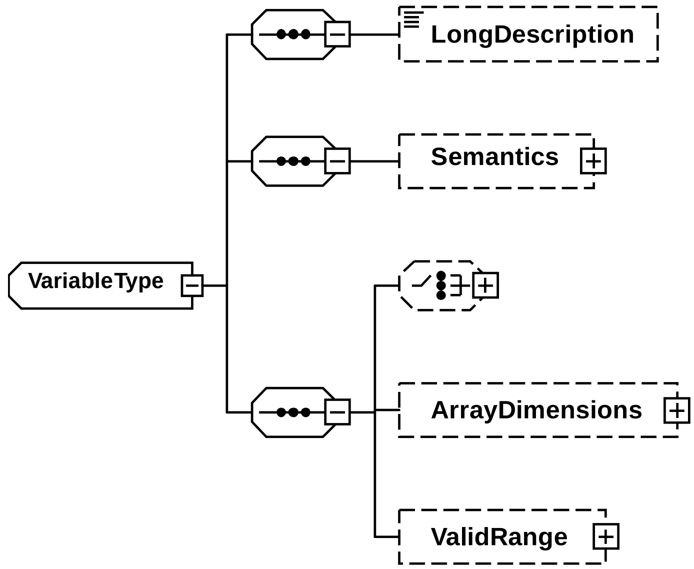

Figure 3-21: Variable Element of a VariableSet

3.14.3 Each `Variable` element shall have the attributes and child elements associated with a field (see 3.11) with the addition of an optional `initialValue` attribute identifying the initial value of the variable, and an optional `readOnly` attribute.

3.14.4 The name of each `Variable` child element of a `VariableSet` element shall be unique.

3.14.5 If the `initialValue` attribute is used to specify an initial value for a variable, the value shall be a literal whose type matches the type of the variable, as specified in table 3-1.

3.14.6 The optional Boolean `readOnly` attribute of a variable shall, if true, indicates that the variable must have an initial value, and may not be subsequently assigned to.

3.14.7 A `ParameterMapSet` element shall contain one or more `ParameterMap` elements.

3.14.8 A `ParameterMap` element shall carry one `parameterRef` attribute and one `variableRef` attribute.

3.14.9 The `parameterRef` attribute of a `ParameterMap` element shall refer to a parameter on an interface provided or required by the component type.

3.14.10 The `variableRef` attribute of a `ParameterMap` element shall refer to a variable declared by the component type.

3.14.11 The types of the elements referred to by the `parameterRef` and `variableRef` attributes shall match. (See subsection 4.3.12 in the SEDS Green Book, reference [D4], for an explanation.)

Table 3-6: Legal Parameter Mappings

Interface Parameter			Component Variable	
Interface	**Mode**	**Read-only**	**Read-only**	**Description**
provided	sync	true	true	Externally-readable constant
provided	sync	true	false	Externally-readable variable
provided	sync	false	false	Externally-updateable variable
required	async	true	false	Externally-supplied variable

3.14.12 The set of properties of the interface parameter and component variable involved in a mapping shall correspond to one of the rows in table 3-6. (See 4.3.12 in the SEDS Green Book, reference [D4], for an explanation.)

3.14.13 A `ParameterActivityMapSet` element shall contain one or more `ParameterActivityMap` elements.

3.14.14 Each `ParameterActivityMap` element maps a parameter on a provided interface to a parameter on a required interface using an activity.

3.14.15 A `ParameterActivityMap` element shall contain a `Provided` element and a `Required` element, each of which carry a `name` attribute and an `interfaceParameterRef` attribute. These elements make the specified interface parameter available within the scope of the activity as a local variable with the specified name.

3.14.16 The interface parameters referenced by the `Provided` and `Required` elements of a `ParameterActivityMap` shall have the same values for the attribute `mode`.

3.14.17 If the interface parameter referenced by the `Required` element of a `ParameterActivityMap` has the `readOnly` attribute set to true, the same must be true of the interface parameter referenced by the `Provided` element.

3.14.18 Additionally, each `ParameterActivityMap` element shall have either

a) one `GetActivity` child element;

b) one `SetActivity` child element; or

c) one `GetActivity` child element and one `SetActivity` child element.

3.14.19 The `GetActivity` and `SetActivity` elements shall specify a sequence of actions to be used for the parameter mapping during a get or set operation on the provided parameter, respectively. The valid child elements for these elements shall be the same as those for the `Body` child element of an `Activity` element (see 3.15.7).

3.14.20 If the interface parameter referenced by the `Required` element of a `ParameterActivityMap` has the `readOnly` attribute set to true, the `SetActivity` child element must not be present.

3.14.21 The same interface parameter may not be referenced more than once across all `ParameterMap` and `ParameterActivityMap` elements of a component.

NOTE – There is no restriction on using the same variable, or fields thereof, for multiple mappings.

3.15 ACTIVITIES

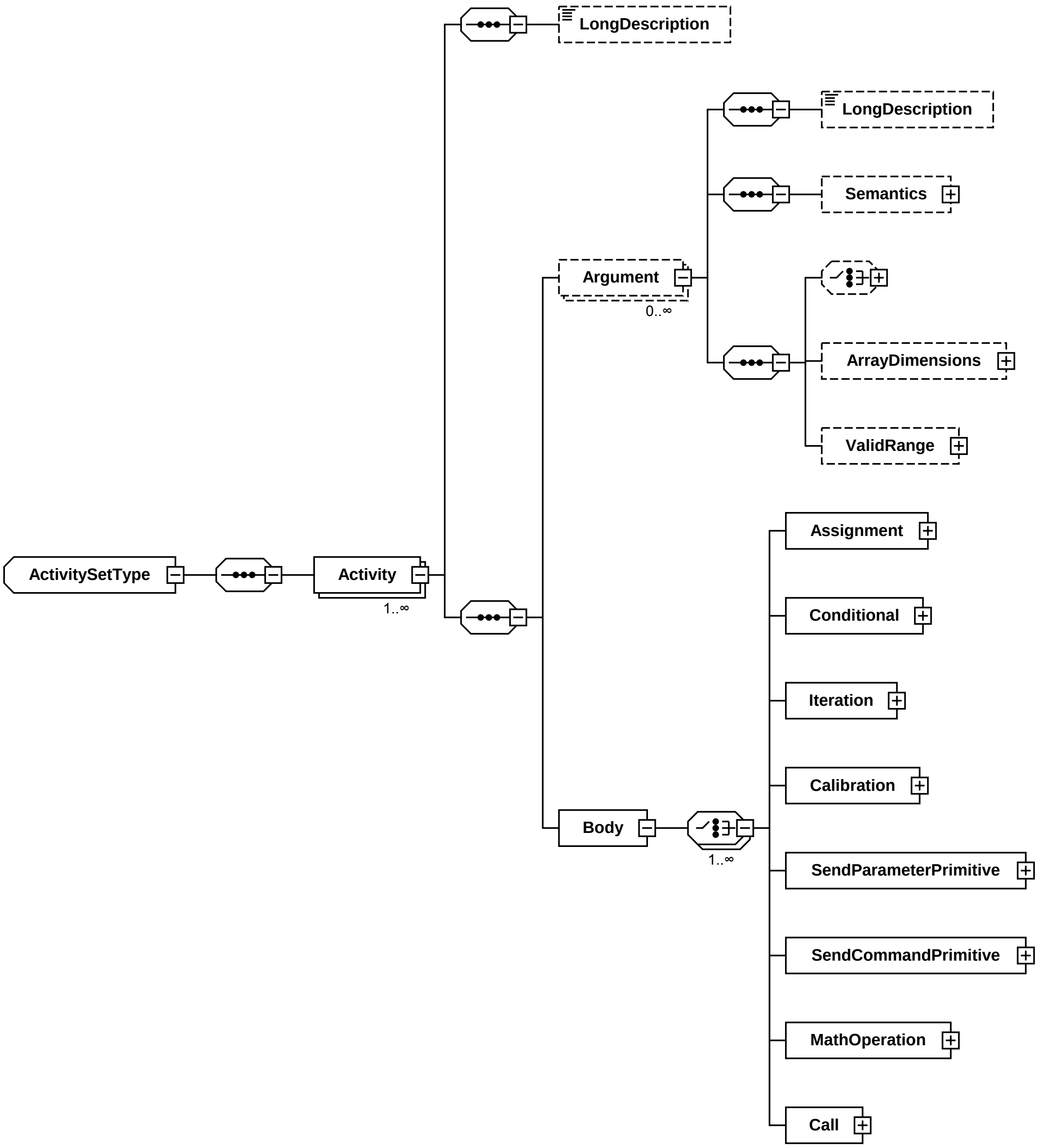

Figure 3-22: Activities within an ActivitySet Element

NOTE – An activity is a block of executable statements whose invocation is controlled by one or more state machines.

3.15.1 The `ActivitySet` element shall contain one or more `Activity` elements.

3.15.2 Each `Activity` element is based on the `NameDescriptionType` (see 3.3.3).

3.15.3 The name of each `Activity` child element of an `ActivitySet` element shall be unique.

3.15.4 Each `Activity` element shall contain zero or more `Argument` elements and one `Body` element.

NOTE – `Argument` elements permit the operation of the activity, specified by the `Body` element, to be parameterized. Parameterization means that invocation of the activity must be accompanied by arguments that provide data values to be used by the activity, and the body of the activity contains statements that refer to those arguments.

3.15.5 Each `Argument` child element of an `Activity` element shall have the attributes and child elements associated with a field (see 3.11).

3.15.6 The name of each `Argument` child element of an `Activity` element shall be unique.

3.15.7 The `Body` child element of an `Activity` element shall contain one or more of the following elements: `SendParameterPrimitive`, `SendCommandPrimitive`, `Calibration`, `MathOperation`, `Assignment`, `Conditional`, `Iteration`, or `Call`.

3.15.8 The sequence of elements specified in the `Body` element shall define the sequence of operations of the activity.

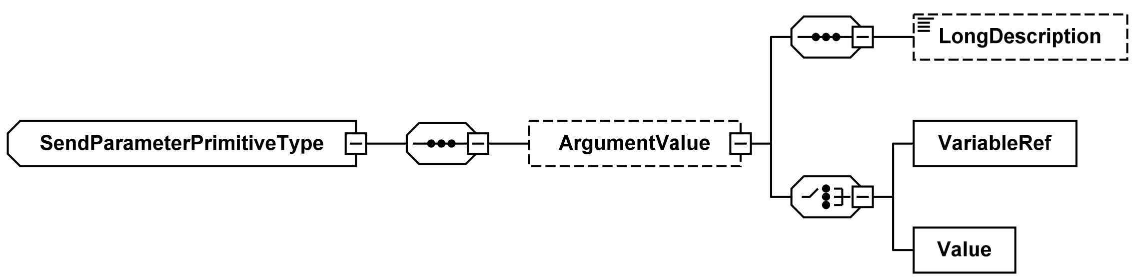

Figure 3-23: Send Parameter Primitive Element

3.15.9 A `SendParameterPrimitive` element shall specify the transmission of a parameter request or indication primitive to an interface provided or required by the component type.

3.15.10 A `SendParameterPrimitive` element shall carry

a) an `interface` attribute, identifying the component interface to which the primitive relates;

b) a `parameter` attribute, identifying the interface parameter to which the primitive relates;

c) an `operation` attribute identifying whether the primitive is for a `get` or `set` operation;

d) an optional `transaction` attribute which permits this primitive to be related to the opposing primitive of the request/indication pair;

e) an optional `failed` attribute, defaulting to false, used in an indication to explicitly report failure of the corresponding request.

3.15.11 The `transaction` attribute of the `SendParameterPrimitive` element shall be present or absent depending on the conditions given in table 3-5.

3.15.12 A `SendParameterPrimitive` element may include an `ArgumentValue` element according to the conditions described in table 3-7.

3.15.13 An `ArgumentValue` element shall include either a `Value` element, specifying a literal value to be associated with the primitive, or a `VariableRef` element, specifying a component variable to associate with the primitive (see table 3-7).

Table 3-7: Arguments to a Primitive

Element	Interface Direction	Parameter Operation	Number of Arguments
`SendCommandPrimitive`	any		0 or more
`OnCommandPrimitive`	any		0 or more
`SendParameterPrimitive`	required	`Get`	0
`SendParameterPrimitive`	provided	`Get`	1
`SendParameterPrimitive`	required	`Set`	1
`SendParameterPrimitive`	provided	`Set`	1
`OnParameterPrimitive`	required	`Get`	1
`OnParameterPrimitive`	provided	`Get`	0
`OnParameterPrimitive`	required	`Set`	1
`OnParameterPrimitive`	provided	`Set`	1

3.15.14 The type of the value specified by either the `VariableRef` or `Value` child element of an `ArgumentValue` element shall match the type of the parameter or command argument to which the primitive relates.

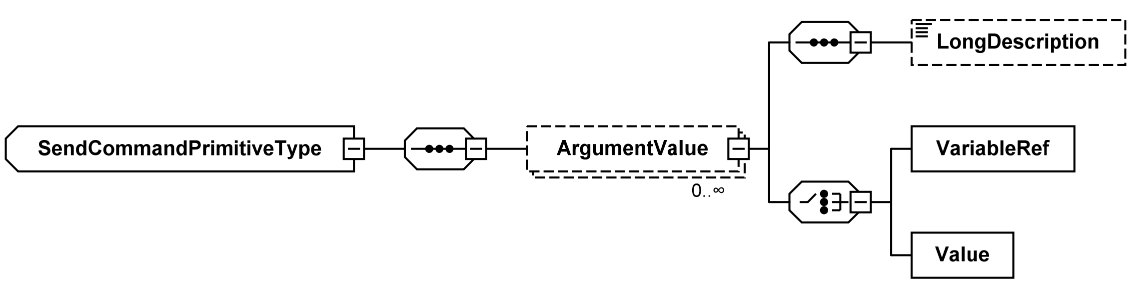

Figure 3-24: Send Command Primitive Element

3.15.15 A `SendCommandPrimitive` element shall specify the transmission of a command request or indication primitive to an interface provided or required by the component type.

3.15.16 A `SendCommandPrimitive` element shall carry:

a) An `interface` attribute, identifying the component interface to which the primitive relates;

b) a `command` attribute, identifying the interface command to which the primitive relates;

c) an optional `transaction` attribute which permits this primitive to be related to the opposing primitive of the request/indication pair;

d) an optional `failed` attribute, defaulting to false, used in an indication to explicitly report failure of the corresponding request.

3.15.17 The `transaction` attribute of the `SendCommandPrimitive` element shall be present or absent according to the conditions expressed in table 3-5.

3.15.18 A `SendCommandPrimitive` element may include a number of `ArgumentValue` element according to the conditions described in table 3-7.

3.15.19 Each `ArgumentValue` child element of a `SendCommandPrimitive` element shall carry a `name` attribute identifying the command argument with which this value is associated.

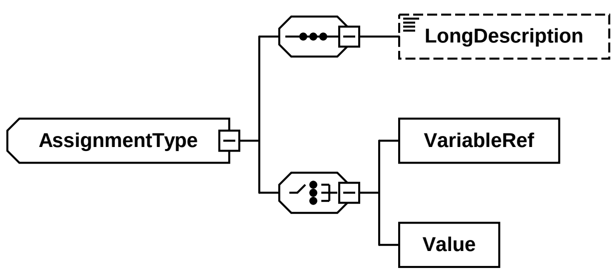

Figure 3-25: Assignment Element

3.15.20 An `Assignment` element shall specify the assignment of a value, specified as either a literal or by referencing a component variable, to a component variable.

3.15.21 An `Assignment` element shall carry an `outputVariableRef` attribute identifying the component variable to which the value should be assigned.

3.15.22 An `Assignment` element shall include either a `Value` element, specifying a literal value to be assigned to the output parameter, or a `VariableRef` element which specifies a component variable to use as the source of the value to assign to the output parameter.

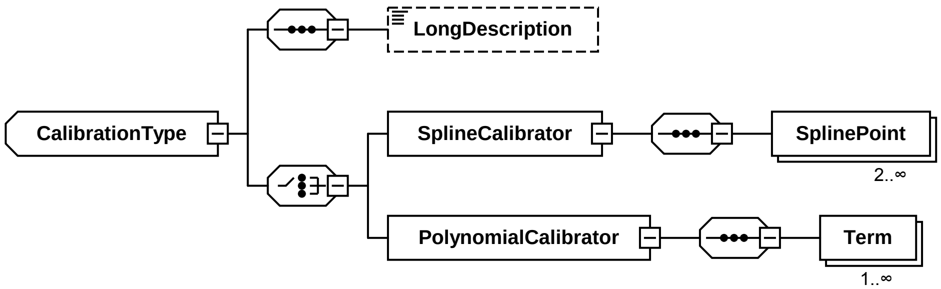

Figure 3-26: Polynomial and Spline Calibrators within a Calibration Element

3.15.23 A `Calibration` element shall specify the assignment of a value, specified as either a literal or by referencing a component variable, to a component variable, translating the value according to a specified calibration operation.

3.15.24 A `Calibration` element shall carry an `outputVariableRef` attribute identifying the component variable to which the calibrated value should be assigned.

3.15.25 A `Calibration` element shall include either a `Value` element, specifying a literal value to calibrate before assignment to the output variable, or an `inputVariableRef` element, specifying a component variable to use as the source of the value to calibrate before assignment to the output parameter.

3.15.26 A `Calibration` element shall include either a `SplineCalibrator` or `PolynomialCalibrator` element.

3.15.27 A `SplineCalibrator` element shall have an attribute `extrapolate`, indicating whether to extrapolate values outside the range of points.

3.15.28 A `SplineCalibrator` element shall have two or more `SplinePoint` child elements.

3.15.29 The attributes of a `SplinePoint` child element of a `SplineCalibrator` shall have attributes `raw` and `calibrated`, which together representing a point on the spline curve used to convert from raw to calibrated values, and order, which represents the algorithm used to interpolate values between this point and the next.

NOTE – A spline of order 1 is linear (i.e., a traditional point calibration). One of order 2 is quadratic, and 3 is cubic. There must be the mathematically necessary number of consecutive points of a given order to support higher-order spline curves.

3.15.30 A `PolynomialCalibrator` element shall have one or more `Term` child elements.

3.15.31 A `Term` child element of a `PolynomialCalibrator` shall have attributes `coefficient` and `exponent`, which together define one term of the polynomial expression used to convert from raw to calibrated values.

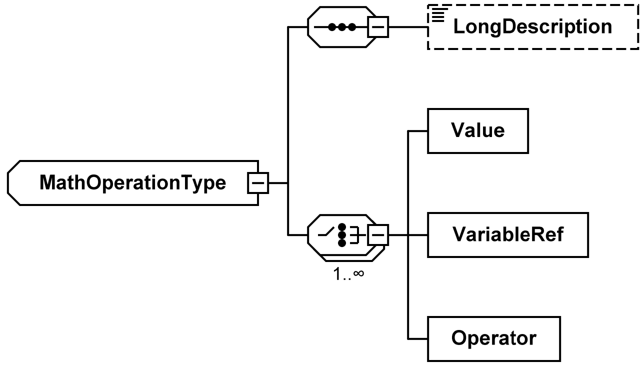

Figure 3-27: Math Operation Element

3.15.32 A `MathOperation` element shall specify a mathematical operation in postfix (Reverse Polish) notation.

NOTE – This means that the sequence of values and operators must be valid when taking account of the 'arity' column of table 3-8.

3.15.33 A `MathOperation` element shall carry an `outputVariableRef` attribute identifying the component variable to which the calculated value should be assigned.

3.15.34 A `MathOperation` element shall include a sequence of the following child elements:

a) `Value`;

b) `VariableRef`; and

c) `Operator`.

3.15.35 The `Value` and `VariableRef` child elements of a `MathOperation` element shall have the same contents and meanings as the elements of the same name an `Assignment element`.

3.15.36 An `Operator` child element of a `MathOperation` element shall have a single attribute `operator`, which shall be one of the values from table 3-8.

Table 3-8: Mathematical Operators

Value	Description	Arity
add	Addition	binary
subtract	Subtraction	binary
multiply	Multiplication	binary
divide	Division	binary
modulus	Remainder	binary
pow	x raised to the power y	binary
ln	Natural (base e) logarithm of x	unary
log	Base 10 logarithm	unary
exp	e raised to a power x	unary
inverse	1/x	unary
tan	Trigonometric function	unary
cos	Trigonometric function	unary
sin	Trigonometric function	unary
atan	Inverse trigonometric function	unary
atan2	Inverse trigonometric function	binary
acos	Inverse trigonometric function	unary
asin	Inverse trigonometric function	unary
tanh	Hyperbolic trigonometric function	unary
cosh	Hyperbolic trigonometric function	unary
sinh	Hyperbolic trigonometric function	unary
atanh	Inverse hyperbolic trigonometric function	unary
acosh	Inverse hyperbolic trigonometric function	unary
asinh	Inverse hyperbolic trigonometric function	unary
swap	Exchange x and y	binary
abs	Absolute value	unary
ceil	Round to integer towards positive infinity	unary
floor	Round to integer towards negative infinity	unary
round	Round to nearest integer, ties as ceil.	unary
min	Minimum	binary
max	Maximum	binary
sqrt	Square root	unary

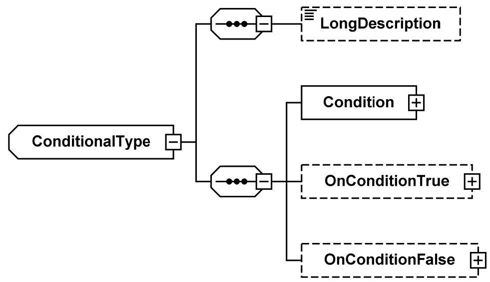

Figure 3-28: Conditional Element

3.15.37 A `Conditional` element shall specify the conditional execution of elements of the activity.

3.15.38 A `Conditional` element shall include one `Condition` element, zero or one `OnConditionTrue` element, and zero or one `OnConditionFalse` element.

3.15.39 A `Condition` element shall specify a Boolean expression as shown in figure 3-29.

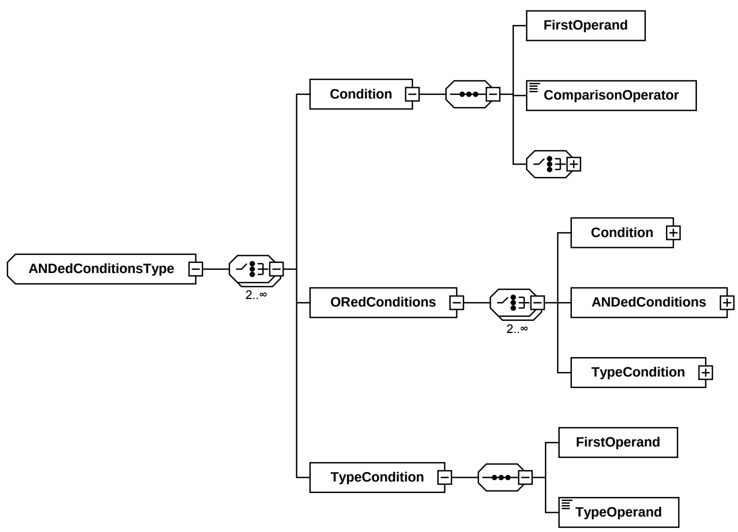

Figure 3-29: Conditional Execution of an Activity or State Machine Transition Guard

3.15.40 A `TypeCondition` element shall be true if the `FirstOperand` value is an instance of the type specified by the `TypeOperand`, or an instance of a derivative of that type.

3.15.41 An `OnConditionTrue` element shall contain one or more of the elements allowed in an activity body specifying the operations to perform if the outcome of the condition expression is ‘true’.

3.15.42 An `OnConditionFalse` element shall contain one or more of the elements allowed in an activity body specifying the operations to perform if the outcome of the condition expression is ‘false’.

3.15.43 An `Iteration` element shall specify the repeated execution of elements of the activity.

3.15.44 An `Iteration` element shall carry an `iteratorVariableRef` attribute identifying the component variable to use to hold the iteration value.

3.15.45 An `Iteration` element shall either contain either: an `OverArray` element, or a `StartAt` element, zero or one `Step` element, and an `EndAt` element, in that order.

3.15.46 An `Iteration` element shall contain a `Do` element after all other elements.

3.15.47 The `OverArray` element of an `Iteration` element shall specify an array over which to iterate, assigning the value of each array element, in turn, to the iteration parameter.

3.15.48 The `StartAt` element shall include either a `Value` element, specifying a literal value to be assigned as the initial value of the iteration parameter, or a `VariableRef` element, specifying a component variable to use as the source of the value to use as an initial value of the iteration parameter.

3.15.49 The `EndAt` element shall include either a `Value` element, specifying a literal value to be used as the final value of the iteration parameter (inclusive), or a `VariableRef` element, specifying a component variable to use as the source of the value to use as the final value of the iteration parameter (inclusive).

3.15.50 A `Step` element shall include either a `Value` element, specifying a literal value to be used as the difference in value of the iteration parameter between iterations, or a `VariableRef` element, specifying a component variable to use as the source of the value to be used as the difference in value of the iteration parameter between iterations.

3.15.51 The `Do` element shall contain one or more of any of the elements allowed in an activity body.

3.15.52 A `Call` element shall identify a nested activity to be called at this point in the activity execution.

3.15.53 A `Call` element shall include zero or more `ArgumentValue` elements, each of which, in turn, shall carry a `name` attribute, identifying the name of an activity argument and including either a `Value` element, specifying a literal value to be associated with the named activity argument, or a `variableRef` element, specifying a component variable to associate with the named activity argument.

3.16 STATE MACHINES

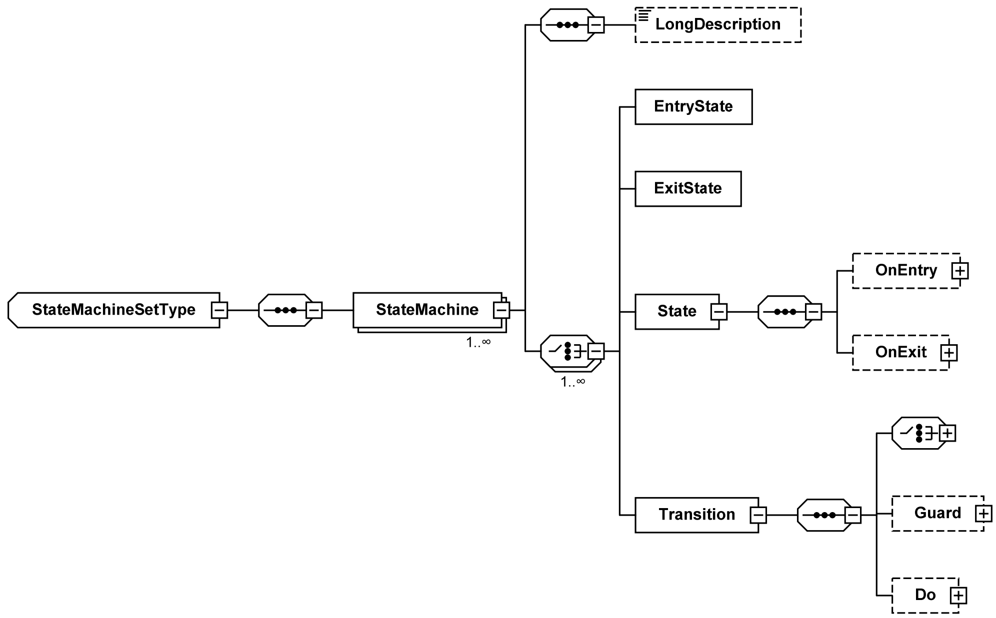

Figure 3-30: State Machines within a StateMachineSet Element

NOTE – A state machine responds to events and schedules the execution of activities.

3.16.1 Each `StateMachine` element may carry a `defaultEntryState` attribute identifying the name of the state to transition to with no action immediately on initialization.

3.16.2 Each `StateMachine` element shall include one or more of the following elements: `EntryState`, `ExitState`, `State`, and `Transition`.

3.16.3 Each child element of a `StateMachine` element shall carry a `name` attribute identifying the name of that element.

3.16.4 The name of each child element of a `StateMachine` element shall be unique within the state machine.

3.16.5 Each `State` element shall include zero or one of the following elements: `OnEntry`, `OnExit`.

3.16.6 The `OnEntry`, `OnExit` and `Do` elements shall each specify the name of an activity, using the `activity` attribute, to be invoked on entry to the state, immediately before exit from the state, and when performing a transition between states, respectively.

3.16.7 The `OnEntry`, `OnExit` and `Do` elements shall each include zero or more `ArgumentValue` elements each of which, in turn, carries a `name` attribute, identifying the name of an activity argument and includes either a `Value` element, specifying a literal value to be associated with the named activity argument, or a `VariableRef` element, specifying a component variable to associate with the named activity argument.

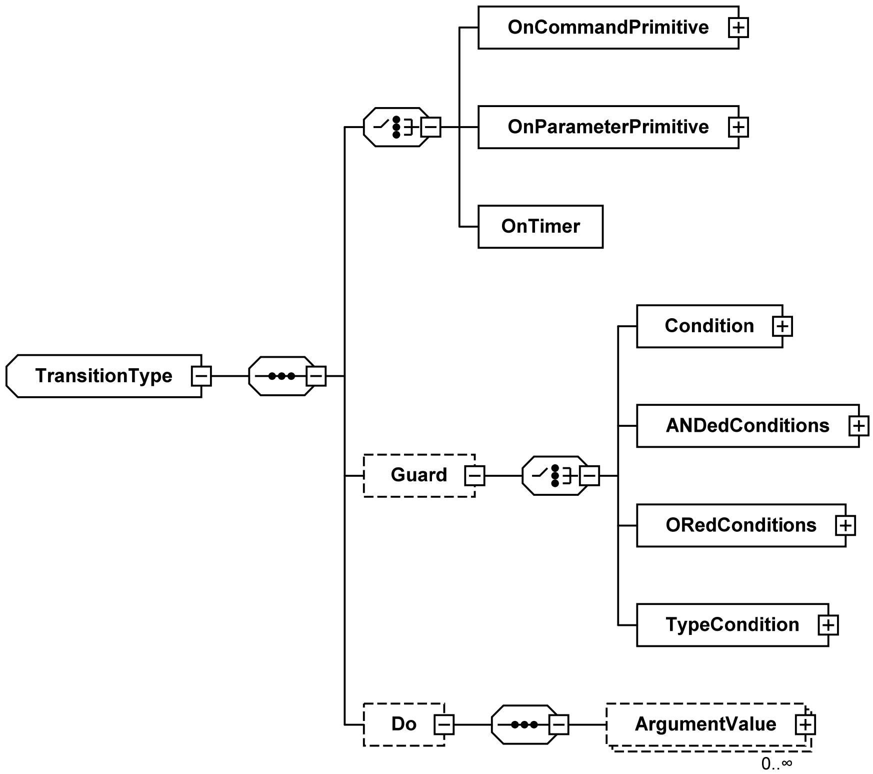

Figure 3-31: State Machine Transition Element

3.16.8 Each `Transition` element shall carry

a) a `fromState` attribute, identifying the name of the state that this transition starts from;

b) a `toState` attribute, identifying the name of the state that this transition ends at;

3.16.9 A transition shall not start from an exit state or end at an entry state.

3.16.10 Each `Transition` element shall include one of the following elements: `OnCommandPrimitive`, `OnParameterPrimitive` and `OnTimer`.

3.16.11 Each `Transition` element shall include zero or one of each of the following elements: `Guard` and `Do`.

3.16.12 An `OnTimer` element shall contain a `nanosecondsAfterEntry` attribute which indicates the number of nanoseconds that must elapse between state entry and triggering the transition, providing that the guard condition is met.

3.16.13 An `OnCommandPrimitive` or `OnParameterPrimitive` element shall identify the primitive that must be received to trigger the transition, providing that the guard condition is met.

3.16.14 An `OnParameterPrimitive` element shall carry:

a) an `interface` attribute, identifying the component interface to which the primitive relates;

b) a `parameter` attribute, identifying the parameter to which the primitive relates;

c) an `operation` attribute, identifying whether the primitive is for a `get` or `set` operation.

d) an optional `transaction` attribute, permitting the primitive reception to be matched to the corresponding primitive transmission using a string identifier;

e) an optional `failed` attribute, defaulting to false, identifying whether the transition should be triggered on successful or failed indications.

3.16.15 The `transaction` attribute of an `OnCommandPrimitive` or `OnParameterPrimitive` element shall be present according to the conditions defined in table 3-5.

3.16.16 An `OnParameterPrimitive` element may, according to the conditions defined in table 3-7, include a `VariableRef` element specifying a component variable to receive the value associated with the primitive.

3.16.17 An `OnCommandPrimitive` element shall carry:

a) an `interface` attribute, identifying the component interface to which the primitive relates;

b) a `command` attribute, identifying the command to which the primitive relates;

c) an optional `transaction` attribute, permitting the primitive reception to be matched to the corresponding primitive transmission using a string identifier;

d) an optional `failed` attribute, defaulting to false, identifying whether the transition should be triggered on successful or failed indications.

3.16.18 An `OnCommandPrimitive` element shall include zero or more `ArgumentValue` elements, each of which, in turn, includes a `VariableRef` element which specifies a component variable to associate with a command argument to the primitive.

3.16.19 A `Guard` child element of a transition shall identify the guard condition that must be met to trigger the transition, providing that the trigger event has been received.

NOTE – If no `Guard` element is present no condition need be met to trigger the transition.

3.16.20 A `Guard` element shall specify a Boolean expression as shown in figure 3-29.

4 CONSTRUCTING AN SEDS/XML INSTANCE

4.1 OVERVIEW

The section describes the rules which must be followed in order to construct a valid electronic data sheet over and above those laid out by the electronic data sheet schema described in section 3.

The phrase ‘to have left the scope of nominal behaviour’ indicates that the device or service described by a SEDS instance has failed. Treatment of the failure must be determined by the designers of the mission where the device or service is used.

4.2 XML VERSION

The first line of each XML file used as part of a SEDS document shall specify the XML version, exactly as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
```

4.3 TYPE REFERENCING AND MATCHING

4.3.1 OVERVIEW

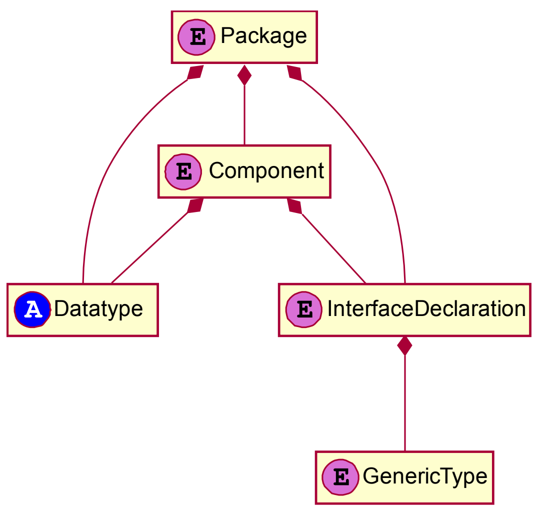

Figure 4-1: Elements and Abstract Types Relevant to Type and Interface Referencing

Datasheets both define and use types and interfaces. This subsection covers how those definitions and their usages can be connected together. In short:

– Types and interfaces can be referenced by their name, which is unique within a package.

– It is possible to reference type and interface definitions defined at the top level of another package, but not those from component or interfaces within that package.

4.3.2 SPECIFICATION

4.3.2.1 Where a data type or interface declaration is referenced from within the same `Package` element, the referencing name shall match the type name exactly.

4.3.2.2 Where a data type or interface declaration is referenced from within the same `Component` element, declarations from the respective `DataTypeSet` and `DeclaredInterfaceSet` elements of that component shall be possible matches.

4.3.2.3 Where a data type or interface declaration is referenced across packages, the referencing name shall use the following syntax:

```
{package name} / {name}
```

4.3.2.4 Where a data type is expected by a `type` attribute on an element, the type referenced shall be a data type.

4.3.2.5 Where an interface type is expected by a `type` attribute on an element, the type referenced shall be an interface type.

4.3.2.6 Where a data type is referenced from within an interface declaration, the type referenced may be a generic type defined on that interface, or any of its base interfaces (as recursively identified by the `BaseType` child element).

4.3.2.7 Where a generic type mapping is specified for a generic type with a base type, the concrete type being specified shall be the same as that specified as the base type or a descendant of the base type.

4.3.2.8 If a mapping for a generic type is necessary, as that generic type is used as the type for an interface parameter or command argument which is, in turn, used within the data sheet, that generic type shall have a valid mapping.

NOTE – It is permissible to leave generic types unbound if they are not used within the data sheet.

4.3.2.9 Where alternate generic type mappings are provided, as alternate sets, the correct set shall be determined using the types and values associated with the relevant primitive.

4.3.2.10 Should multiple alternate generic type sets match a primitive, the most restrictive set shall be chosen.

NOTE – More restrictive means that any valid ranges associated with the type are smaller and/or the type is a closer relation.

4.3.2.11 Where a component variable, interface parameter, or argument is used in relation to a destination component variable, interface parameter, or argument, the types of the source and destination shall match.

4.3.2.12 Where a source literal is used in relation to a destination component variable, interface parameter, or argument, the value of the literal shall be valid according to table 3-1.

4.3.2.13 Activity or state machine operations which reference nonliteral values shall reference component variables only, not interface parameters.

NOTE – Interface parameters can only be accessed using the dedicated operations described in 4.5.

4.3.2.14 Activity or state machine operations which reference a parameter, variable or argument which is an instance of a container parameter type may select a single entry from the container using the following syntax:

```
{parameter name}.{entry name}
```

4.3.2.15 Activity or state machine operations which reference a parameter, variable or argument which is an array may select a single element from the array using the following syntax:

```
{parameter name}[{0-based element index}]
```

NOTE – The element index may be specified as the current value of a variable or argument of integer type.

4.3.2.16 The above two rules may be chained together to access nested array and container entries.

4.4 EXTERNAL REFERENCES

4.4.1 OVERVIEW

To support applications where an electronic data sheet is used to describe the interfaces of a software component or smart device, it may be necessary to reference externally stored values from any part of the EDS.

For example, a reusable software component may be designed to support an arbitrary number of instances of a particular piece of logic. The specific number of instances chosen may have an impact on the EDS, for instance it may affect the `Dimension` element on an `ArrayType` used for describing telemetry messages produced by that software component.

The specific value for this deployment characteristic is driven by the requirements of the project or mission deploying the software, but the EDS file describing the interface(s) of the software component would be authored by the software component vendor. Therefore, the software vendor cannot dictate a specific value in the EDS, but must include a placeholder such that the user can supply the value.

To solve this issue, the EDS may reference external values in place of absolute values in any other EDS element or attribute.

4.4.2 REQUIREMENTS

4.4.2.1 External references in EDS definitions shall use the following syntax to indicate a placeholder where the actual value is to be supplied:

```
${name}
```

4.4.2.2 The names of external references may contain alphanumeric characters and the '_' and '.' symbols.

4.4.2.3 Validation of EDS files utilizing external references shall be performed after the substitution of absolute values has taken place.

NOTE – This may be performed by a dedicated preprocessing tool, or as an intermediate output of the toolchain.

4.5 PRIMITIVE ASSOCIATIONS

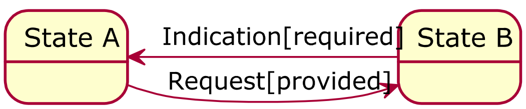

Figure 4-2: Primitives That Trigger State Transition

4.5.1 Where a parameter primitive is to be received (to trigger a state machine transition), the primitive shall be:

a) a get operation primitive from an interface provided by the component identifying a parameter value read request;

b) a set operation primitive from an interface provided by the component identifying a parameter value write request;

c) a get operation primitive from an interface required by the component identifying a parameter value read indication;

d) a set operation primitive from an interface required by the component identifying a parameter value write indication.

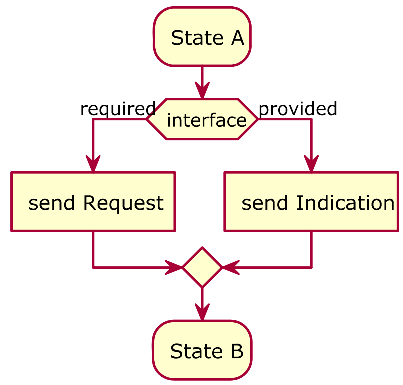

Figure 4-3: Primitives Sent During Activity Execution

4.5.2 Where a parameter primitive is to be transmitted (by an activity), the primitive shall be:

a) a get operation primitive to an interface provided by the component identifying a parameter value read indication;

b) a set operation primitive to an interface provided by the component identifying a parameter value write indication;

c) a get operation primitive to an interface required by the component identifying a parameter value read request;

d) a set operation primitive to an interface required by the component identifying a parameter value write request.

4.5.3 The reception of a get operation parameter indication primitive or a set operation parameter request primitive shall specify a component variable into which the parameter value can be received.

4.5.4 The transmission of a set operation parameter request primitive or a get operation parameter indication primitive shall specify a value for the parameter.

4.5.5 In the case of interface parameters marked as synchronous (having their `mode` attribute set to '`sync`') primitives shall

a) always be transferred in pairs: one transmitted primitive and one received primitive (in the appropriate order); and

b) be associated using an identical string specified as the `transaction` attribute.

4.5.6 In the case of interface parameters marked as asynchronous (having their `mode` attribute set to '`async`') primitives shall always be a single get operation indication primitive:

a) transmitted to a component provided interface;

b) received from a component required interface.

4.5.7 An attempt to transmit a get operation request primitive to an asynchronous interface parameter on required interface of the component shall be invalid.

4.5.8 An attempt to receive a get operation request primitive from an asynchronous interface parameter on a provided interface of the component shall be invalid.

4.5.9 Where a command primitive is to be received (to trigger a state machine transition), the primitive shall be:

a) from an interface *provided* by the component identifying a command execution request;

b) from an interface *required* by the component identifying a command execution indication.

4.5.10 Where a command primitive is to be transmitted (by an activity), the primitive shall be:

a) to an interface *provided* by the component identifying a command execution indication;

b) to an interface *required* by the component identifying a command execution request.

4.5.11 The reception of a command request primitive may specify the component variable into which the value of any arguments of modes `in` or `inout` can be stored.

4.5.12 The reception of a command indication primitive may specify the component variable into which the value of any arguments of modes `out` or `inout` can be stored.

4.5.13 The transmission of a command request primitive shall specify a value for all arguments of modes `in` or `inout`.

4.5.14 The transmission of a command indication primitive shall specify a value for all arguments of modes `out` or `inout`.

4.6 STATE MACHINE OPERATION

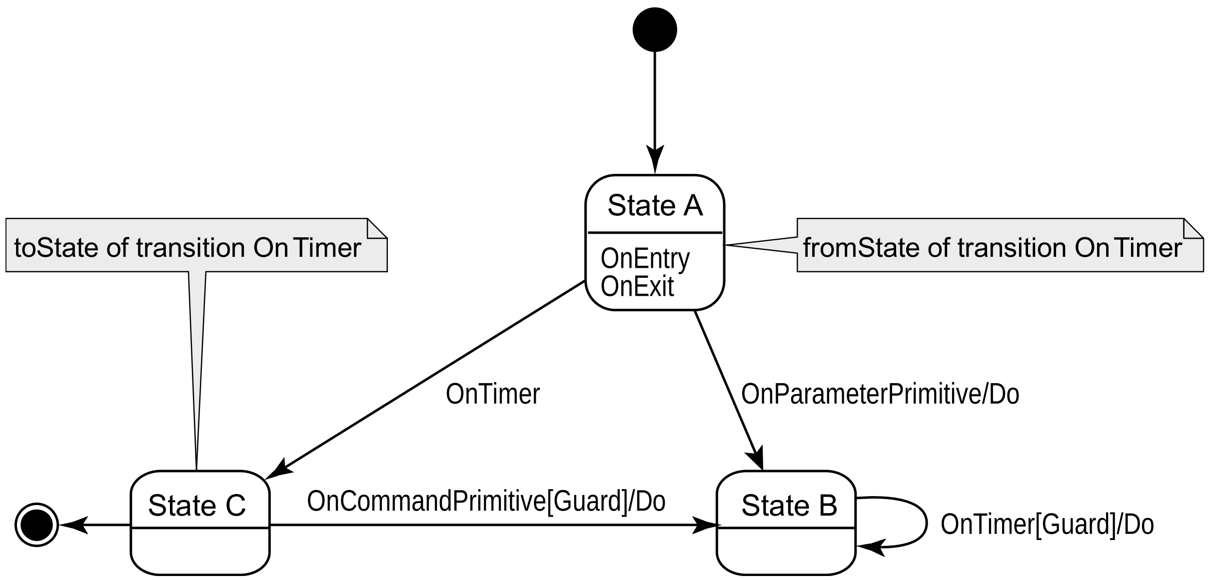

Figure 4-4: State Machine Concepts

4.6.1 A state machine transition shall trigger only if the state machine is in the state identified as the `fromState` of the transition.

4.6.2 A state machine transition with an `OnXPrimitive` element shall trigger only when the corresponding primitive is received.

4.6.3 A state machine transition with an `OnTimer` element shall trigger only when the corresponding time after state entry is reached.

4.6.4 If a state machine transition guard is present, the transition shall trigger only if the guard condition is met.

4.6.5 If multiple transitions from a state have identical conditions, then a transition without a guard shall trigger only if the guard conditions of all other transitions are false.

NOTE – Such a transition effectively represents an 'ELSE' condition.

4.6.6 An external transition is a transition where the `fromState` of the transition is not equal to its `toState`.

4.6.7 When a transition is triggered, the following actions shall be performed in order:

a) for an external transition, the state machine shall exit the state identified as the `fromState` of the transition;

b) for all transitions, any activity specified in the Do child element of the transition shall be executed.

c) for an external transition, the state machine shall enter the state identified as the `toState` of the transition.

4.6.8 When a state is exited, this shall result in execution of the state `onExit` activity, if such an activity is specified.

4.6.9 When a state is entered, this shall result in execution of the state `onEntry` activity, if such an activity is specified.

4.6.10 In order to determine the required logic of a state machine specified in a data sheet, activities shall be assumed to complete instantaneously.

NOTE – Timing considerations should be modelled by states with `OnTimer` elements.

4.6.11 Any possible incoming primitive must meet the trigger conditions of only a single transition.

NOTE – If not, the data sheet is invalid, although this cannot necessarily be statically detected. When developing a SEDS for a device, a mapping from primitives to state machine transitions is implied by the guard conditions. The mapping could contain complex expressions that depend upon data appearing in the interface, so testing all combinations could be impractical.

4.6.12 Only one `EntryState` element shall be present in a given state machine, and if it is present the `defaultEntryState` attribute shall not be set.

4.6.13 If an explicit `EntryState` element is present, it shall be used as the starting state.

NOTE – This allows explicit specification of initialization actions.

4.6.14 If the `defaultEntryState` attribute is present, a default starting state shall be used, causing an immediate and unconditional transition, with no action, into the specified state.

4.6.15 If a state machine transitions to an exit state, the device should be considered to have left the scope of the nominal behaviour documented by the datasheet.

NOTES

1 This would be used in the case of a failure to respond within a reasonable time, an explicit error report, etc. This would normally be an indication that higher-level, mission-specific or operator-driven fault recovery mechanisms should be initiated.

2 Similar considerations would apply if a device behaved in a way not documented in the data sheet, e.g., sending a message not anticipated by the current set of active states.

3 A state machine can include logic to explicitly detect and respond to failures in lower layers. In the absence of such specification, error events will not be handled, and so the state machine will transition to the error event as above.

4.7 ENCODING AND DECODING

4.7.1 A value is encoded when a value of a nonbinary data type is used for an outgoing argument of a command on an interface, and that argument has the `dataUnit` attribute set to true. A toolchain must use encoding specifications of a data type to encode an item of data for sending outbound through an interface. This encoding ensures that a protocol data unit will have the expected format at the receiving endpoint.

4.7.2 A value is decoded when a value of a nonbinary data type is used for an incoming argument of a command on an interface, and that argument has the `dataUnit` attribute set to true. Parameters are implicitly treated the same as an argument with the `dataUnit` attribute set to true. A toolchain may use encoding specifications to decode an item of data received through an interface. This decoding allows for variation in memory words and for variation in alignment of entries of structures across different platforms and compilers.

4.7.3 If a scalar value is encoded or decoded, it must have an encoding specification set.

4.7.4 If a scalar value with a valid range is encoded, the encoding specification must provide for the full extent of the range.

NOTE – This means, for example, that a value with a valid range that includes negative numbers cannot use an `unsigned` integer encoding.

4.7.5 If a scalar value with a valid range is decoded, and the decoded value is outside the valid range, the device should be considered to have left the scope of the nominal behaviour documented by the datasheet.

4.7.6 If an enumerated value is decoded, and the decoded value does not correspond to any of the values of the enumeration, the device should be considered to have left the scope of the nominal behaviour documented by the datasheet.

4.7.7 The constraints applicable to a container are those contained within its `ConstraintSet` child element, plus all those specified on containers referenced as a direct or indirect base type.

4.7.8 If a container is decoded, and the incoming data does not match all applicable constraints, the device should be considered to have left the scope of the nominal behaviour documented by the datasheet.

4.7.9 When an abstract container is encoded or decoded, the concrete container type to use shall be selected from the set of all non-abstract containers that have that container as a direct or indirect base type.

4.7.10 If an abstract container is decoded, and the incoming data does not match any of the candidate concrete container types, the device should be considered to have left the scope of the nominal behaviour documented by the datasheet.

4.7.11 If the `sizeInBits` attribute of `FloatDataEncoding` element disagrees with the standard identified in the `encodingAndPrecision` attribute in the same element, the condition shall be treated as an error.

4.8 DISCUSSION

4.8.1 OVERVIEW

The following considerations can help in decoding a string data type.

4.8.2 DETERMINING THE LENGTH OF THE STRING

4.8.2.1 General

The attribute `fixedlength="true"` indicates a fixed amount of space allocated for a string, specified by the length attribute. The string in that space could vary in length by termination byte or by Pascal length. The content of any space after the string is undefined. If there is no termination byte and there is no Pascal length, then the string exactly fills the fixed amount of space.

The attribute `fixedlength="false"` implies that the string length needs to be determined by one of the methods below. The length attribute is the maximum number of bytes in the string.

– If the data type is in a container of known length, and it is the only variable-length entry in the container, then the lengths of all fixed-length entries are subtracted from the length of container, and the difference is the length of the string.

– Pascal length: An integer in the same container with the string can be designated to represent the number of bytes (not necessarily the same as characters) in the string. The advantage of this representation is that it can be determined in constant time.

– The space allocated for the string can be scanned, seeking a termination byte. If that byte is present, it is just beyond the end of the string, and it is not counted in the length of the string. If that byte is absent or undefined, and there is no Pascal length, then the string exactly fills the maximum space defined by the length attribute.

4.8.2.2 Decoding the String

After the length of the string is known, then it is possible to interpret the content of the string as characters, according to the encoding attribute.

4.8.2.3 Specifying a Pascal Length

To indicate that an integer in a container contains the length of a string data type in the same container, the following element can be inserted into the entry for the integer:

<Semantics unit="count" quantityKind="quantityQK" subject="…"/>

The ellipsis in the element above is replaced with the name of the entry for the string.

4.8.3 EXTENSIBLE ENUMERATIONS

The following attributes can be extended by means of auxiliary schemas, which the SEDS schema includes:

a) errorControlType attribute of an ErrorControlEntry element;

b) encodingAndPrecision attribute of FloatDataEncoding element;

c) encoding attribute of IntegerDataEncoding element;

d) encoding attribute of StringDataEncoding element;

e) operator attribute of Operator element.

The SEDS schema includes the following auxiliary schemas:

a) seds-core-semantics.xsd is generated from the SOIS Dictionary of Terms (DoT). This auxiliary schema contains extensions that have been normalized by the curators of the DoT and published in SANA. Normalization identifies external conventions and standards that are of general use in describing interoperable systems across space agencies.

b) seds-extension-semantics.xsd is provided for use within single projects to identify conventions for use by the project toolchain. This auxiliary schema is maintained by the project personnel.

SEDS instances that use the auxiliary schema seds-extension-semantics.xsd cannot be interoperable across agencies, because that auxiliary schema is not managed by a standards organization.

ANNEX A

ELECTRONIC DATA SHEET FOR ONBOARD DEVICES IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

(NORMATIVE)

A1 INTRODUCTION

This annex provides the Implementation Conformance Statement (ICS) Requirements List (RL) for implementation of the SEDS, CCSDS 876.0-R-3, August 2018. The ICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation shall satisfy the mandatory conformance requirements of the base standards referenced in the RL.

The RL in this annex is blank. An implementation's complete RL is called an ICS. The ICS states which capabilities and options of the services have been implemented. The following can use the ICS:

– The service implementer, as a checklist to reduce the risk of failure to conform to the standard through oversight;

– The supplier and acquirer or potential acquirer of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard ICS proforma;

– The user or potential user of the implementation, as a basis for initially checking the possibility of interoperability with another implementation;

– A service tester, as a basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

A2 NOTATION

The following are used in the RL to indicate the status of features:

<u>Status Symbols</u>

M mandatory

O optional

Support Column Symbols

The support of every item as claimed by the implementer is stated by entering the appropriate answer (Y, N or N/A) in the Support column:

Y	Yes, supported by the implementation
N	No, not supported by the implementation
N/A	Not applicable

A3 REFERENCED BASE STANDARDS

The base standards references in the RL are:

– Electronic Data Sheet for Onboard Device – this document.

A4 GENERATION INFORMATION

A4.1 IDENTIFICATION OF ICS

Ref	Question	Response
1	Date of Statement (DD/MM/YYYY)	
2	ICS serial number	
3	System Conformance statement cross-reference	

A4.2 IDENTIFICATION OF IMPLEMENTATION UNDER TEST (IUT)

Ref	Question	Response
1	Implementation name	
2	Implementation version	
3	Special configuration	
4	Other information	

A4.3 IDENTIFICATION

Ref	Question	Response
1	Supplier	
2	Contact Point for Queries	
3	Implementation name(s) and Versions	
4	Other information necessary for full identification, e.g., name(s) and version(s) for machines and/or operating systems: System Name(s)	

A4.4 SERVICE SUMMARY

Ref	Question	Response
1	Service Version	
2	Addenda implemented	
3	Amendments implemented	
4	Have any exceptions been required? NOTE – A YES answer means that the implementation does not conform to the service. Non-supported mandatory capabilities are to be identified in the ICS, with an explanation of why the implementation is nonconforming.	Yes ______ No ______

A4.5 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the specification by completing the RL; that is, compliance to all mandatory requirements and the options that are not supported are shown. The resulting completed RL is called an ICS. In the Support column, each response shall be selected either from the indicated set of responses or it shall comprise one or more parameter values as requested. If a conditional requirement is inappropriate, N/A shall be used. If a mandatory requirement is not satisfied, exception information must be supplied by entering a reference X*i*, where *i* is a unique identifier, to an accompanying rationale for the noncompliance.

The implementers affected by this RL are writers of software that reads and interprets Electronic Data Sheets for use in computer-assisted engineering.

A5 GENERAL/MAJOR CAPABILITIES

Service Feature	Reference	Status	Support
SEDS Core	3.2, 3.3, 4.2	M	
Device Metadata	3.4	O	
Packages	3.5	M	
Data Types	3.6	M	
Scalar Data Types	3.7	M	
Ranges	3.8	M	
Arrays	3.9	M	
Containers	3.10	M	
Fields	3.11	M	
Interface Types	3.12	M	
Component Types	3.13	M	
Component Implementations	3.14	O	
Activities	3.15	O	
State Machines	3.16	O	
Type Referencing and Matching	4.3	M	
Primitive Associations	4.4	O	
State Machine Operation	4.5	O	
Encoding and Decoding	4.6	O	

A6 UNDERLYING LAYERS PROVIDING SERVICES TO IMPLEMENTATION

This subsection provides identification of the underlying layers providing services to the implementation.

Service Feature	Reference	Status	Support
XInclude	3.2.2–3.2.4	M	
User-defined Ontology	3.2.6	O	

ANNEX B

SECURITY, SANA, AND PATENT CONSIDERATIONS

(INFORMATIVE)

B1 SECURITY CONSIDERATIONS

B1.1 SECURITY BACKGROUND

The SOIS services are intended for use with protocols that operate solely within the confines of an onboard subnet. It is therefore assumed that SOIS services operate in an isolated environment which is protected from external threats. Any external communication is assumed to be protected by services associated with the relevant space-link protocols. The specification of such security services is out of scope of this document.

B1.2 SECURITY CONCERNS

At the time of writing there are no identified security concerns. If confidentiality of data is required within a spacecraft it is assumed it is applied at the Application Layer. More information regarding the choice of service and where it can be implemented can be found in reference [D3].

B1.3 POTENTIAL THREATS AND ATTACK SCENARIOS

Potential threats and attack scenarios typically derive from external communication and are therefore not the direct concern of the SOIS services, which make the assumption that the services operate within a safe and secure environment. It is assumed that all applications executing within the spacecraft have been thoroughly tested and cleared for use by the mission implementer. Confidentiality of applications can be provided by Application Layer mechanisms or by specific implementation methods such as time and space partitioning. Such methods are outside the scope of SOIS.

B1.4 CONSEQUENCES OF NOT APPLYING SECURITY

The security services are out of scope of this document and are expected to be applied at layers above or below those specified in this document. If confidentiality is not implemented, science data or other parameters transmitted within the spacecraft might be visible to other applications resident within the spacecraft resulting in disclosure of sensitive or private information.

B1.5 RELIABILITY

While it is assumed that the underlying mechanisms used to implement the devices operate correctly, the SEDS make no assumptions as to their reliability.

B2 SANA CONSIDERATIONS

The recommendations of this document request SANA to create a registry named 'Spacecraft Onboard Interface Services Electronic Data Sheets and Dictionary of Terms' that consists of a set of files that constitute an ontology and related files. The candidate registry is located at the following URL: http://beta.sanaregistry.org/r/sois/sois.html.

The registration rule for change to this registry requires an engineering review by a designated expert. The expert shall be assigned by the SOIS-APP working group Chair, or in absence, Area Director.

The registry shall contain at least the following items.

Table B-1: SANA Registry Content

File	Description
seds.xsd	The schema for SOIS Electronic Data Sheets.
seds-core-semantics.xsd	The SOIS Dictionary of Terms in the form of a schema to be included by seds.xsd.
seds.xml	A non-normative collection of definitions that can reduce the number of definitions in an electronic data sheet.
sois.0.owl	The ontology for SOIS Dictionary of Terms. This ontology imports sysml-qudv-si-sois.owl.
sysml-qudv.owl	The definition of quantities, units, dimensions, and values.
sysml-qudv-si.owl	The extension of QUDV to the International System of Units. This ontology imports SysML-QUDV.owl.
sysml-qudv-si-sois.owl	An extension of the original QUDV ontologies to support units used in SOIS EDS. This ontology imports SysML-QUDV-SI.owl.
soisOwlToXsd.zip	A compressed project that converts a conformant ontology into a seds-core-semantics.xsd.

B3 PATENT CONSIDERATIONS

The technology used in managing SEDS (xml and xsd) is in the public domain.

ANNEX C

ABBREVIATIONS AND ACRONYMS (INFORMATIVE)

CCSDS	Consultative Committee for Space Data Standards
CPTP	CCSDS Packet Transfer Protocol
DACP	Device Abstraction Control Procedure
DSAP	Device-specific Access Protocol
DAS	Device Access Service
DDPS	Device Data Pooling Service
DoT	Dictionary of Terms
DVS	Device Virtualization Service
ID	Identifier
MAS	Memory Access Service
Mb/s	Mega-bits per second
MTS	Message Transfer Service
OBC	Onboard Computer
OSI	Open Systems Interconnection
OWL	Web Ontology Language
PS	Packet Service
QUDV	Quantities, Dimensions, Units, Values
RDF	Resource Description Framework
RIU	Remote Interface Unit
RMAP	Remote Memory Access Protocol
SANA	Space Assigned Numbers Authority
SEDS	SOIS Electronic Data Sheet
SOIS	Spacecraft Onboard Interface Services
SW	Software
SpW	SpaceWire
TM/TC	Telemetry/Telecommands
XML	Extensible Markup Language
URI	Uniform Resource Identifier

ANNEX D

INFORMATIVE REFERENCES (INFORMATIVE)

[D1] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. 2nd ed. International Standard, ISO/IEC 7498-1:1994. Geneva: ISO, 1994.

[D2] *Spacecraft Onboard Interface Services*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 850.0-G-2. Washington, D.C.: CCSDS, December 2013.

[D3] *The Application of CCSDS Protocols to Secure Systems*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 350.0-G-2. Washington, D.C.: CCSDS, January 2006.

[D4] *Electronic Data Sheets and Dictionary of Terms for Onboard Devices and Components*. Report Concerning Space Data System Standards. Forthcoming.

ANNEX E

EXAMPLE SEDS/XML SCHEMA INSTANTIATIONS

(INFORMATIVE)

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSheet
   xmlns="http://www.ccsds.org/schema/sois/seds"
   xmlns:xi="http://www.w3.org/2001/XInclude"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ccsds.org/schema/sois/seds seds.xsd">
  <Device name="SimpleDevice" shortDescription="Simple arbitrary example of SEDS XML usage">
  </Device>
  <xi:include href="ccsds.sois.subnetwork.xml" xpointer="element(/1/1)"/>
  <Package name="SimpleDemo">
    <DataTypeSet>
      <IntegerDataType name="MyInteger">
        <Range>
            <MinMaxRange min="0" max="4294967296" rangeType="inclusiveMinInclusiveMax" />
        </Range>
    </IntegerDataType>
    </DataTypeSet>
    <DeclaredInterfaceSet>
      <Interface name="DeviceAccessInterface">
        <ParameterSet>
          <Parameter name="DeviceMode" type="MyInteger"/>
        </ParameterSet>
        <CommandSet>
          <Command name="DoSomething">
            <Argument name="WithANumber" type="MyInteger" mode="in"/>
          </Command>
        </CommandSet>
      </Interface>
    </DeclaredInterfaceSet>
    <ComponentSet>
      <Component name="DeviceDACP">
        <ProvidedInterfaceSet>
          <Interface name="VendorSpecificInterface" type="DeviceAccessInterface"/>
        </ProvidedInterfaceSet>
        <RequiredInterfaceSet>
          <Interface name="Subnetwork" type="CCSDS/SOIS/Subnetwork/MASInterfaceType"/>
        </RequiredInterfaceSet>
      </Component>
    </ComponentSet>
  </ Package >
</DataSheet>
```

The above example shows a datasheet defining a device `SimpleDevice` with a single component `DeviceDACP` that in turn provides a single interface, `VendorSpecificInterface`. The interface type `DeviceAccessInterface` has one command `DoSomething` and one parameter `DeviceMode`. Both of those definitions share a single data type, `MyInteger`.

The definition of the subnetwork interface used (*`MASInterfaceType`*) is provided in an external file. It should be noted that in this example, no implementation of the component is defined; a fully specified device datasheet would include the logical transformations needed to map between the required and provided interfaces as state machines.