

Draft Recommendation for  
Space Data System Standards

|  |
| --- |
| Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets |

Proposed Draft Recommended Standard

June 2016

AUTHORITY

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Issue: | , |  |
|  | Date: |  |  |
|  | 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 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.
* KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
* Korea Aerospace Research Institute (KARI)/Korea.
* Ministry of Communications (MOC)/Israel.
* 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.
* 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** |
|  | , , |  | Current draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

CONTENTS

Section Page

[1 introduction 1-1](#_Toc438217624)

[1.1 Purpose and scope of this Document 1-1](#_Toc438217625)

[1.2 Applicability 1-1](#_Toc438217626)

[1.3 Rationale 1-1](#_Toc438217627)

[1.4 Document Structure 1-1](#_Toc438217628)

[1.5 Conventions and Definitions 1-2](#_Toc438217629)

[1.6 NOMENCLATURE 1-4](#_Toc438217630)

[1.7 References 1-4](#_Toc438217631)

[2 OvervieW 2-1](#_Toc438217632)

[2.1 The Subject Matter of Electronic Data Sheets 2-2](#_Toc438217633)

[2.2 Purpose and Operation of SOIS Electronic Data SheetS 2-4](#_Toc438217634)

[2.3 Use of W3C Recommendations 2-6](#_Toc438217635)

[3 Basic Structure of the SEDS/XML SCHEMA 3-1](#_Toc438217636)

[3.1 Overview 3-1](#_Toc438217637)

[3.2 Electronic Data Sheets and the Associated Schema 3-2](#_Toc438217638)

[3.3 SEDS/XML Basic Structure 3-3](#_Toc438217639)

[3.4 Device MetaData 3-5](#_Toc438217640)

[3.5 Namespaces 3-5](#_Toc438217641)

[3.6 Data Types 3-2](#_Toc438217642)

[3.7 Scalar Data Types 3-4](#_Toc438217643)

[3.8 Ranges 3-8](#_Toc438217644)

[3.9 Arrays 3-9](#_Toc438217645)

[3.10 Containers 3-10](#_Toc438217646)

[3.11 Type Instances 3-13](#_Toc438217647)

[3.12 Interfaces 3-15](#_Toc438217648)

[3.13 Components 3-20](#_Toc438217649)

[3.14 Component Implementations 3-23](#_Toc438217650)

[3.15 Activities 3-26](#_Toc438217651)

[3.16 State Machines 3-35](#_Toc438217652)

[4 Constructing an SEDS/XML Instance 4-1](#_Toc438217653)

[4.1 Overview 4-1](#_Toc438217654)

[4.2 XML Version 4-1](#_Toc438217655)

[4.3 Type Referencing and Matching 4-1](#_Toc438217656)

[4.4 Primitive Associations 4-5](#_Toc438217657)

[4.5 State Machine Operation 4-8](#_Toc438217658)

[4.6 Encoding and Decoding 4-10](#_Toc438217659)

[ANNEX A Electronic Data Sheet for Onboard Devices Implementation Conformance Statement Proforma (Normative) A-1](#_Toc438135486)

[ANNEX B Security, SANA, and Patent CONSIDERATIONS (Informative) B-1](#_Toc438135487)

[ANNEX C Abbreviations and Acronyms (informative) C-1](#_Toc438135488)

[ANNEX D Informative References (informative) D-1](#_Toc438135489)

[ANNEX E Example SEDS/XML Schema Instantiations (Informative) E-1](#_Toc438135490)

Figure Page

[Figure 2‑1: SEDS Concept Map 2-1](#_Toc438135491)

[Figure 2‑2: SEDSs Describe Data Interfaces in a Spacecraft 2-2](#_Toc438135492)

[Figure 2‑3: Command and Data Acquisition Services Context 2-3](#_Toc438135493)

[Figure 3‑1: Overview of Selected Key Elements and Abstract Types of a Datasheet 3-1](#_Toc438135494)

[Figure 3‑2: Datasheet, Device and Namespace Elements 3-3](#_Toc438135495)

[Figure 3‑3: Fields, Semantics and Terms 3-4](#_Toc438135496)

[Figure 3‑4: Metadata Element 3-5](#_Toc438135497)

[Figure 3‑5: Data Types within a DataTypeSet Element 3-2](#_Toc438135498)

[Figure 3‑6: Scalar Data Types 3-4](#_Toc438135499)

[Figure 3‑7: Ranges 3-8](#_Toc438135500)

[Figure 3‑8: ArrayDataType and Dimension Elements 3-9](#_Toc438135501)

[Figure 3‑9: Constraints and Entries of a ContainerDataType Element 3-10](#_Toc438135502)

[Figure 3‑10: Type Instance Schema Type 3-13](#_Toc438135503)

[Figure 3‑11: Interfaces within an InterfaceDeclarationSet element 3-15](#_Toc438135504)

[Figure 3‑12: Parameters in a ParameterSet Element 3-17](#_Toc438135505)

[Figure 3‑13: Commands in a CommandSet element 3-18](#_Toc438135506)

[Figure 3‑14: Components in a Namespace 3-20](#_Toc438135507)

[Figure 3‑15: Generic Type Mapping 3-21](#_Toc438135508)

[Figure 3‑16: Implementation element of a Component 3-23](#_Toc438135509)

[Figure 3‑17: Activity Element 3-26](#_Toc438135510)

[Figure 3‑18: Send Parameter Primitive Element 3-27](#_Toc438135511)

[Figure 3‑19: Send Command Primitive Element 3-28](#_Toc438135512)

[Figure 3‑20: Parameter Assignment as Used within an Activity 3-29](#_Toc438135513)

[Figure 3‑21: Polynomial and Spline Calibrations 3-29](#_Toc438135514)

[Figure 3‑22: Math Operations 3-31](#_Toc438135515)

[Figure 3‑23: Conditional Element 3-33](#_Toc438135516)

[Figure 3‑24: Conditional Execution of an Activity or State Machine Transition Guard 3-33](#_Toc438135517)

[Figure 3‑25: State Machines 3-35](#_Toc438135518)

[Figure 3‑26: State Machine Transition Events 3-36](#_Toc438135519)

[Figure 4‑1: Datasheet name reference structure 4-1](#_Toc438135520)

[Figure 4‑2: Primitives that trigger state transition 4-5](#_Toc438135521)

[Figure 4‑3: Primitives sent during activity execution 4-5](#_Toc438135522)

[Figure 4‑4: State Machine Concepts 4-8](#_Toc438135523)

Table Page

[Table 3‑1: Data Types, Encodings, Ranges and Literals 3-5](#_Toc438217821)

[Table 3‑2: MinMaxRange Options 3-8](#_Toc438217822)

[Table 3‑3: Error Control Types 3-12](#_Toc438217823)

[Table 3‑4: Interface Levels 3-16](#_Toc438217824)

[Table 3‑5: Interface Syntax, Primitives, and Transactions 3-19](#_Toc438217825)

[Table 3‑6: Legal Parameter Mappings 3-24](#_Toc438217826)

[Table 3‑7: Arguments to a Primitive 3-28](#_Toc438217827)

[Table 3‑8: Mathematical Operators 3-32](#_Toc438217828)

# introduction

## 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.

## Applicability

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

## Rationale

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

## Document Structure

This document has several major sections:

* Section 1, this section, 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 of 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.

## Conventions and Definitions

### Definitions

#### 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.

#### 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. 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. [D2]

**dictionary of terms, DoT**: Ontology of terms used to describe data in interfaces in Electronic Data Sheets. [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. [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 [D2], compliant with SOIS standards. See 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. [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.

## NOMENCLATURE

### Normative Text

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

1. the words ‘shall’ and ‘must’ imply a binding and verifiable specification;
2. the word ‘should’ implies an optional, but desirable, specification;
3. the word ‘may’ implies an optional specification;
4. 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.

### 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.

## 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 0. Draft Recommendation for Space Data System Practices (Proposed Red Book), CCSDS 876.1-R-0. Washington, D.C.: CCSDS, March 2015.

[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] W3C OWL Working Group, ed. “OWL 2 Web Ontology Language Document Overview.” W3C Recommendation. 2nd ed., 11 December 2012. http://www.w3.org/TR/xinclude/.

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

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

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

[10] *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.

[11] *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.

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

[13] *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.

[] *Information processing -- Use of longitudinal parity to detect errors in information messages*. ISO 1155:1978.

[15] *Space Engineering—SpaceWire—Links, Nodes, Routers and Networks*. ECSS-E-ST-50-12C. Noordwijk, The Netherlands: ECSS Secretariat, 31 July 2008.

[16] *Space Engineering—SpaceWire Protocol Identification*. ECSS-E-ST-50-51C. Noordwijk, The Netherlands: ECSS Secretariat, 5 February 2010.

[17] *Space Engineering—SpaceWire—CCSDS Packet Transfer Protocol*. ECSS-E-ST-50-53C. Noordwijk, The Netherlands: ECSS Secretariat, 5 February 2010.

[20] *CCSDS SANA registry. http://sanaregistry.org/r/sois/*.

NOTE – Informative references are contained in annex D.

# OvervieW

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.

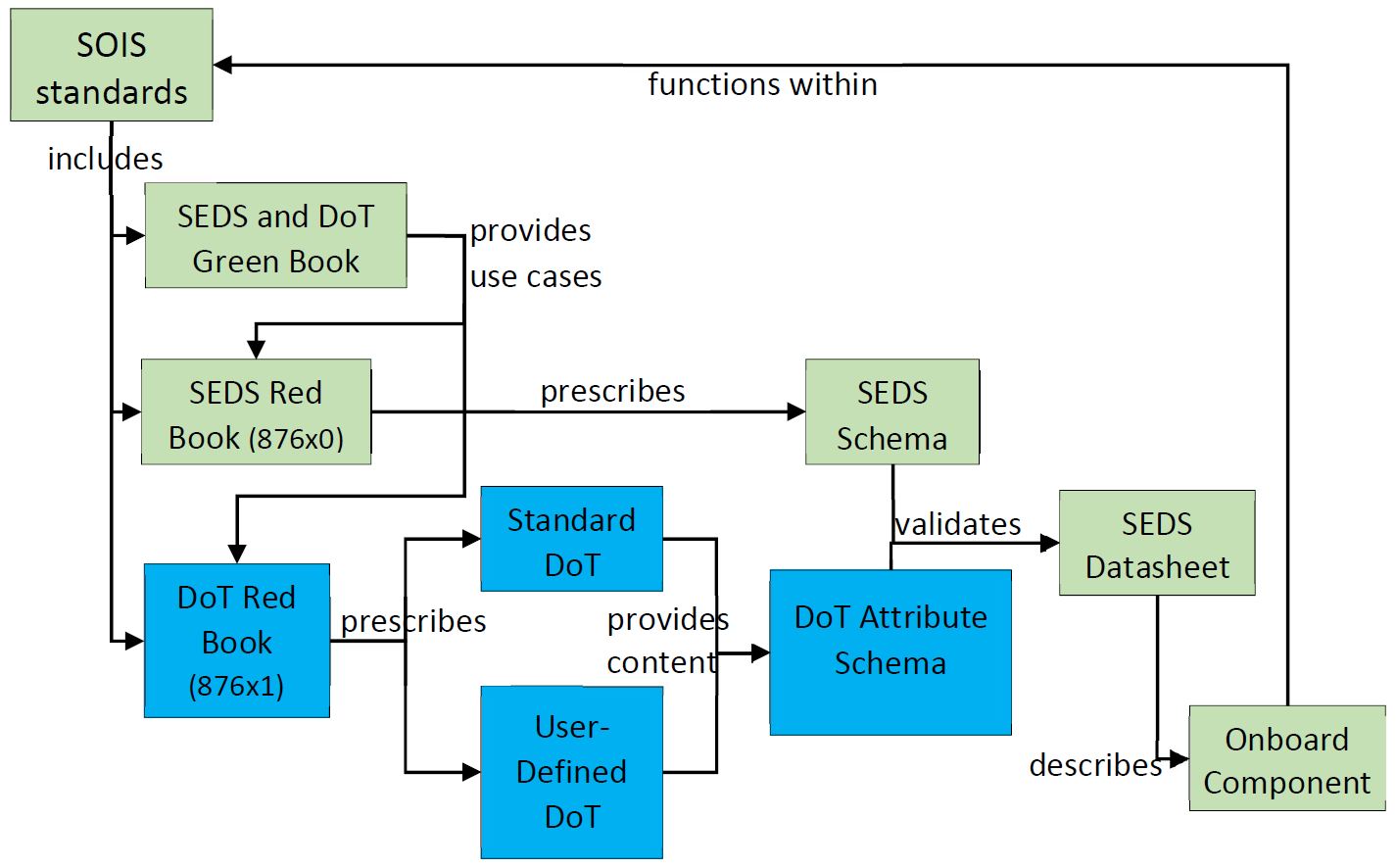


Figure 2‑1: SEDS Concept Map

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 [3].

## 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 SEDSs 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: SEDSs Describe Data Interfaces in a Spacecraft

A SEDS can be used to describe the format of information in a data interface for any onboard device accessed using the Subnetwork Layer. This is illustrated in Figure 2‑3, in which:

* Fixed data interfaces corresponding to CCSDS standard services are represented as ovals.
* Lines between services represent service usage.
* Rectangles represent the OSI primitives passed between services.
* The cloud represents the area where SEDS is used to specify the details of the data interfaces supported by a particular onboard device.



Figure 2‑3: SEDS Describes Data Interfaces of an Onboard Device

The Subnetwork Layer provides standard services mapped onto subnetwork-specific protocols to send and receive discrete packets [D4], to access remote memory [D5], to synchronise [D6] with the subnetwork, and also to discover [D7] and test [D8] devices on the subnetwork.

Some portions of the application layer corresponding to a device are sufficiently universally fixed to be usefully specified as an application-level service, i.e.:

* Clocks may support the Time Access Service.
* Mass Memory Devices may support the File and Packet Store Services
* Dynamically reconfigurable devices may support the Device Enumeration Service.

The remainder is specified by the SOIS Electronic Datasheet for that device. This captures all device-specific aspects, including 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 implementation of other SOIS application services such as the DAS, DVS and MTS (see reference [D2]).

## 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:

1. generating human-readable documentation;
2. specifying interfaces to the device;
3. automatically generating software implementing the relevant parts of the OBSW for the device;
4. automatically generating device interface simulation software for use in test or device-simulation software;
5. transforming the device functional interface into telecommands and telemetry suitable for processing by a command and data handling system onboard and on the ground;
6. 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]).

A full SEDS for a device specifies:

1. the Device-Specific Functional Interface (DSFI) to the device;
2. the Device-Specific Access Interface (DSAI) to the device
3. the Device Abstraction Control Procedure (DACP) which maps the virtual device interface onto the physical interface;
4. the Device-specific Access Protocol (DSAP) which maps the physical device interface onto the SOIS subnetwork services appropriate for the device;
5. information specifying the use of the subnetwork by the device and any constraints placed on the subnetwork;
6. ancillary information, i.e. the device metadata.

Each device interface (virtual and physical) may be composed of multiple interface definitions, some or all of which may be standardised. By implementing additional interfaces, or by extending standard interfaces, a device may offer vendor-specific functions without compromising compatibility with standard functions. Interfaces, data types and protocol elements can be defined in isolation and reused by multiple data sheets. This is enabled by permitting a data sheet document to span multiple files, each of which may contain one or more reusable elements. Although a full data sheet is the most powerful, partial data sheets still offer significant benefits and require less effort to generate. The use of multiple files is especially important in the case where various elements of a data sheet are provided either by different organisations or at different times during the development life cycle of a mission.

Consequently, the SEDS treats the top-level elements enumerated above as optional, allowing validation of partial datasheets.

In order to be able to relate the elements of the data sheet to physical (and non-physical) concepts, and to promote standardisation 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 utilise 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.

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 they 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;
* customisable to support additional features or requirements of any particular tool or system, without compromising the above.

## Use of W3C Recommendations

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

1. XML—The Extensible Markup Language (reference [2]) is used to mark up data sheet documents in a machine-readable manner.
2. 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.
3. 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.

# Basic Structure of the SEDS/XML SCHEMA

## 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, Section 3.2 covers the nature and relation of the various XML and XSD files that make up a datasheet.

Within those files, a datasheet (Section 3.3) contains device metadata (Section 3.4) plus multiple namespaces (Section 3.5), which:

* define a variety of data types (Section 3.6);
* declare interfaces (Section 3.12) referencing those types;
* contain components (Section 3.13) that specify a behavioral mapping (Section 3.14) between those interfaces.

In turn, components:

* are defined by a set of state machines (Section 3.16);
* that control the execution of a set of activities (Section 3.15).

The data types supported are:

* Single-valued scalars (Section 3.7), which can be limited by ranges (Section 3.8);
* Arrays (Section 3.9);
* Containers (Section 3.10);
* Anonymous inline types (3.11), which can specify the details of a single parameter, argument or field without requiring a separate named type definition.

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.

## Electronic Data Sheets and the Associated Schema

The basic unit of data exchange of SOIS device information is the electronic data sheet. A data sheet shall be captured in an XML document known as a SEDS.

A SEDS document shall be composed of one or more XML files.

Where more than one XML file is used XInclude (reference [5]) may be used to combine the files into a single document.

NOTE – The XInclude dependencies therefore form a tree structure of XML files making up the SEDS document.

Both a complete SEDS document, and also any individual files intended to be included in such a document, shall be an instance of (i.e., compliant to) the SEDS schema, or an extension of it.

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 [20]).
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]).

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 [6].

## SEDS/XML Basic Structure



Figure 3‑2 : Datasheet, Device and Namespace Elements

The root element of a SEDS document shall be the DataSheet element.

The DataSheet element shall contain zero or one Device elements.

NOTE – This element represents the device which is being described by the SEDS document. The schema allows an XML file to contain zero devices in order to allow validation of datasheet fragments representing define interfaces or shared declarations.

The Device element shall contain and zero or one Metadata elements.

NOTE – In the diagram notation above, this is shown by the Device element having an optional child element Metadata.

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

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

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

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

The Device element shall be based on the NameDescriptionType (see 3.3.4).



Figure 3‑3 : Fields, Semantics and Terms

Where any SEDS element is based on the FieldType, that element is required to be based on the NameDescriptionType (see 3.3.4), and may have an optional Semantics child element.

NOTES

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

– These XML schema types use inheritance to share attribute and element definitions. So ‘an element based on the FieldType’ means ‘any element with an XML schema type derived from FieldType.

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.

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.

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

## MetaData



Figure 3‑4 : Metadata Element

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

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

The Category element is based on NameDescriptionType (see 3.3.4).

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

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.

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

DateValue and StringValue elements shall contain a value attribute specifying the value of the metadata as a literal per Table 3‑1

FloatValue and IntegerValue elements may contain a value attribute specifying the value of the metadata as a literal per Table 3‑1

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.

## Namespaces

The DataSheet element shall contain zero or more Namespace elements.

NOTES

1. Here, a namespace is simply a section of a datasheet used to define a package of related data types, interfaces and components.
2. There is no relation between such a namespace and the XML schema concept of namespaces.

A Namespace element may contain the following optional elements, in the order DataTypeSet, DeclaredInterfaceSet, ComponentSet.

The name of each Namespace element declared shall be unique within the datasheet.

A Namespace name may be hierarchical, in which case it will consist of multiple name segments separated by the slash character.

NOTES

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

A namespace may have shortDescription attribute and a LongDescription child element.

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

The declaration of any type or interface within a namespace shall be referencable by using a forward slash (‘/’) separated ‘path’ structure, where the first slash is not given (e.g., a/b); if the path portion is not given, the name shall refer to an item within the local namespace only (e.g., voltage).

NOTE - For details of referencing types or interfaces within a namespace from other elements, see Section 4.3.

## Data Types

DataTypes

Figure 3‑5 : Data Types within a DataTypeSet Element

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

Each child element of a DataTypeSet element is based on the FieldType (see 3.3.6).

The name of each child element of a DataTypeSet element shall be unique within the containing namespace.

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

## Scalar Data Types

ScalarTypes1ScalarTypes2

Figure 3‑6: Scalar Data Types

NOTES

* A Scalar data type is a single-valued data type, as opposed to structured types like arrays or containers.
* Scalar types may specify how they are to be encoded. This information is used when they are transmitted over a subnetwork.
* All encoding specifications should be considered as complete and standalone, with no inheritance mechanism.
* Numeric scalar data types may specify a range of representable values, which form a constraint on the possible encodings. See Section 3.8 for details.
* Scalar types may have values specified in a datasheet as literals (see Table 3‑1).

Each BooleanDataType, EnumeratedDataType, FloatDataType, IntegerDataType, StringDataType or SubRangeDataType element may contain an optional encoding element of a type corresponding to the 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 |

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

1. bigEndian, the default, for values which are to be encoded most significant byte first;
2. littleEndian for values which are to be encoded least significant bytes first.

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

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

1. zeroIsFalse (the default);
2. nonZeroIsFalse.

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

1. unsigned, for an unsigned value;
2. signMagnitude, for an encoding with a separate sign bit (most significant bit is the sign bit, with 1 indicating negative);
3. twosComplement, for twos complement;
4. onesComplement, for ones complement;
5. BCD, for a binary coded decimal, where each octet is a decimal digit encoded as binary;
6. packedBCD, where each octet contains two decimal digits encoded as binary, preceded by an optional ‘-‘ sign in ASCII.

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

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

1. IEEE754\_2008\_single;
2. IEEE754\_2008\_double;
3. IEEE754\_2008\_quad;
4. MILSTD\_1750A\_simple;
5. MILSTD\_1750A\_extended.

NOTE – These represent the supported sizes of IEEE (reference [7]) and MIL-STD-1750A (reference [8]).

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

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.

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

1. UTF-8 specifying Unicode UTF-8 encoding (reference [9]); or
2. ASCII, the default, specifying US ASCII encoding (reference [10]).

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

NOTES

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

– This termination character is not included in the count of characters that make up the string.

– UTF-8 characters use a variable length encoding that can contain as much as 4 bytes per character.

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

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

An Enumeration element may have an optional child Semantics element.

## Ranges



Figure 3‑7: Ranges within a SubRangeDataType element

Each EnumeratedDataType, FloatDataType, IntegerDataType, EnumeratedDataType or SubRangeDataType element shall contain zero or one Range element of a type corresponding to the table 3‑1.

A SubRangeDataType element shall contain a baseType attribute, referring to the type which defines all properties other than range.

A PrecisionRange element shall be either SINGLE or DOUBLE, representing the full supported representation range of the corresponding IEEE float data encodings.

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 |

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

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

## Arrays

ArrayDataType

Figure 3‑8 : ArrayDataType and Dimension Elements

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

NOTE – See Section 4.3 for details.

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

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.

Where an array has multiple dimension elements, this shall be treated as equivalent to defining an array with only the first Dimension element, and an element type with the remaining dimensions.

NOTES

* The maximum length of an array with a specified index type can be found by looking at the maximum value of the index range.
* 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.
* Variable-length arrays, i.e. lists, are only supported within the context of a container. For details, see Section 3.10.

## Containers



Figure 3‑9 : Constraints and Entries of a ContainerDataType Element

NOTES

* Containers are aggregate data types with named entries which each can be of any type.
* There is no distinct concept of a ‘container encoding’. Any container can specify encoding information; this should be ignored where not applicable.
* When used for multi-level self-identifying structured data (e.g. CCSDS space packets) containers are organised in a hierarchy where the concrete container representing a specific packet has a base type of the container representing the packet header and trailer.
* Constraints on header fields represent an implicit algorithm for classifying raw packet data step by step until the final concrete container is selected. See Section 4.7 for details.

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.

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

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

An abstract ContainerDataType element may include zero or one TrailerEntryList elements.

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.

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.

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.

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).

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.

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.

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

The first entry in a 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.

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

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

Each Entry, FixedValueEntry, ListEntry, LengthEntry, and ErrorControlEntry element shall have the attributes and child elements associated with an external data type instance (see section 3.11.

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.

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.

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.

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

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.

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.

If a LengthEntry element has a calibration (see 3.11.7), that 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).

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

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 the table 3‑3.

Table 3‑3 : Error Control Types

|  |  |  |
| --- | --- | --- |
| **Value** | **Description** | **Reference** |
| CRC16\_CCITT | G(X) = X^16 + X^12 + X^5 + 1 | [11], subsection 4.1.4.2 |
| CRC8 | G(x) = x^8 + x^2 + x^1 + x^0 | [12], clause 5.2 |
| CHECKSUM | modulo 2^32 addition of all 4‑octet | [13], subsection 4.1.2 |
| CHECKSUM\_LONGITUDINAL | Longitudinal redundancy check, bitwise XOR of all octets | [14] |

## Type Instances



Figure 3‑10: Type Instance Schema Type

NOTES

* 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 subsection describes these attributes and elements such that they may be referenced whenever a data type instantiation is described elsewhere in this document.
* This concept permits the definition of parameter, arguments or container entries that are a sub-range, encoding or array of the referenced type; this avoids the need to define artificial explicit named types.
* An extension of the type instance concept is the External Type Instance schema type, which is used in cases where the value of the type instance may appear on an interface (i.e. parameters, command arguments and container entries).

A type instance is based on the FieldType (see 3.3.6).

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

NOTE – See Section 4.3 for details.

A type instance shall have zero or one ValidRange elements, zero or one ArrayDimensions elements, and zero or one encoding elements.

Any ValidRange and encoding child elements of a type instance shall match the type attribute according to table 3‑1.



Figure 3‑11: External Type Instance Schema Type

A NominalRangeSet child element of an external type instance specifies the nominal operating limits of the parameter, command argument or container entry. It has the same child elements and attributes as other Range elements (see section 3.8).

A SafeRangeSet child element of an external type instance specifies the safe operating limits of the parameter, command argument or container entry. It has the same child elements and attributes as other Range elements (see section 3.8).

A SplineCalibratororPolynomialCalibrator child element of an external type instance 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 parameter, command argument or container entry (see section 3.15).

## Interfaces

InterfaceType

Figure 3‑12 : Interfaces within an InterfaceDeclarationSet Element

NOTES

* Standardised interfaces, including those to the subnetwork, are defined with this interface construct to allow them to be treated symmetrically with user-defined interfaces.
* Any interface declared within a data sheet is implicitly scoped to the data sheet, bypassing any complexities associated with having multiple devices.
* Consequently, when these device-scoped interfaces are connected up to services that support multiple devices, like the DAS, DVS or subnetwork, a mapping layer will be necessary to translate device and address information. This would normally be a simple static table (see reference [1]).
* An interface may be defined in terms of generic types, to avoid placing undue restrictions on its implementation or use. Such interfaces must have those generic types mapped to fully-specified types before use (see 3-163.12.7).

An InterfaceDeclarationSet element shall contain one or more Interface elements.

Each Interface child element of an InterfaceDeclarationSet is based on the NameDescriptionType (see 3.3.4).

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

An Interface element shall contain zero or one BaseTypeSet element containing one or more BaseType 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.

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

An Interface element must have an attribute level, with value taken from Table 3‑4, which indicates the system level it operates at.

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. |

Each BaseType child element of a BaseTypeSet 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).

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 3.3.4).

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

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



Figure 3‑13 : Parameters in a ParameterSet Element

Each Parameter child element of a ParameterSet element shall have the attributes and child elements associated with an external type instance (see 3.11) with the addition of an optional mode attribute, identifying the parameter mode, and an optional readOnly attribute, identifying if the parameter is read-only.

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.

Valid values for the mode attribute of a Parameter element shall be ‘sync’ (the default) or ‘async’ indicating how the parameter data is transmitted across the interface.

NOTE – See Section 4.5 for further details.

Valid values for the readOnly attribute shall be ‘false’ (the default) or ‘true’.



Figure 3‑14 : Commands in a CommandSet Element

Each Command child element of a CommandSet element is based on the NameDescriptionType (see 3.3.4), plus an optional mode attribute, identifying the command mode.

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.

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.

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

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

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

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 – See Section 4.7 for 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 |

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

NOTE – See Section 4.5 for further details.

## Components

ComponentType

Figure 3‑15: Components in a ComponentSet Element

The ComponentSet element contained in a namespace shall contain one or more Component elements.

Each Component child element of a ComponentSet element is based on the NameDescriptionType (see 3.3.4).

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

A Component element shall contain, in order,

1. zero or one ProvidedInterfaceSet elements;
2. zero or one RequiredInterfaceSet elements;
3. zero or one DataTypeSet elements;
4. zero or one DeclaredInterfaceSet element; and
5. zero or one Implementation elements.

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

Each Interface child element of a ProvidedInterfaceSet or RequiredInterfaceSet element is based on the NameDescriptionType (see 3.3.4).

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

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 Namespace or Component.

NOTE – See Section 4.3 for details.

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 3‑16 : Generic Type Mapping

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 type instance (see 3.11) with the optional addition of a fixedValue attribute.

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.

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. The alternate sets shall specify mutually exclusive alternative type mappings for the specified generic types.

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

The optional DeclaredInterfaceSet child element of a Component shall define interfaces declaration local to the component, which cannot be referenced outside it.

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

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.

## Component Implementations



Figure 3‑17: Implementation Element of a Component

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

1. VariableSet;
2. ParameterMapSet;
3. ParameterActivityMapSet;
4. ActivitySet;
5. StateMachineSet.

A VariableSet element shall contain one or more Variable elements.

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

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

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.

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

A ParameterMapSet element shall contain one or more ParameterMap elements.

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

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

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

The types of the elements referred to by the parameterRef and variableRef attributes shall match.

Table 3‑6 : Legal Parameter Mappings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interface Parameter** | | | **Component Variable** | **Description** |
| **Interface** | **Mode** | **Read-only** | **Read-only** |
| 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 |

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.

A ParameterActivityMapSet element shall contain one or more ParameterActivityMap elements.

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

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.

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

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.

Additionally, each ParameterActivityMap element shall have either:

1. one GetActivity child element;
2. one SetActivity child element; or
3. one GetActivity child element and one SetActivity child element.

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).

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.

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.

## Activities



Figure 3‑18: Activities within an ActivitySet Element

The ActivitySet element shall contain one or more Activity elements.

Each Activity element is based on the NameDescriptionType (see 3.3.4).

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

Each Activity element shall contain zero or more Argument elements and one Implementation element.

NOTE – Argument elements permit the operation of the activity, specified by the Body element, to be parameterised.

Each Argument child element of an Activity element shall have the attributes and child elements associated with a type instance (see Section 3.11).

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

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.

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



Figure 3‑19 : Send Parameter Primitive Element

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

A SendParameterPrimitive element shall carry

1. An interface attribute, identifying the component interface to which the primitive relates;
2. a parameter attribute, identifying the interface parameter to which the primitive relates;
3. an operation attribute identifying whether the primitive is for a get or set operation;
4. an optional transaction attribute which permits this primitive to be related to the opposing primitive of the request/indication pair.

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

A SendParameterPrimitive element may include an ArgumentValue element according to the conditions described in Table 3‑7.

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 figure 3‑19.).

Table 3‑7 : Arguments to a Primitive

|  |  |  |  |
| --- | --- | --- | --- |
| **Element** | **Interface Direction** | **Parameter Operation** | **Number of Argument Value** **Elements** |
| SendCommandPrimitive | any |  | 0 or more |
| OnCommand | any |  | 0 or more |
| SendParameterPrimitive | required | get | 0 |
| SendParameterPrimitive | provided | get | 1 |
| SendParameterPrimitive | required | set | 1 |
| SendParameterPrimitive | provided | set | 1 |
| OnParameter | required | get | 1 |
| OnParameter | provided | get | 0 |
| OnParameter | required | set | 1 |
| OnParameter | provided | set | 1 |

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 3‑20 : Send Command Primitive Element

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

A SendCommandPrimitive element shall carry:

1. An interface attribute, identifying the component interface to which the primitive relates;
2. a command attribute, identifying the interface command to which the primitive relates;
3. an optional transaction attribute which permits this primitive to be related to the opposing primitive of the request/indication pair.

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

A SendCommandPrimitive element may include a number of ArgumentValue element according to the conditions described in Table 3‑7.

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

AssignmentType

Figure 3‑21 : Assignment Element

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.

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

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.

CalibrationType

Figure 3‑22 : Polynomial and Spline Calibrators within a Calibration Element

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.

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

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.

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

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

A SplineCalibrator element shall have 2 or more SplinePoint child elements.

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 cubic. There must be the mathematically necessary number of consecutive points of a given order to support higher-order spline curves.

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

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.

MathOperation

Figure 3‑23 : Math Operation Element

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

NOTE – This means that sequence of values and operators must be valid when taking account of the ‘arity’ column of Table 3‑8.

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

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

1. Value;
2. VariableRef; and
3. Operator.

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.

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 |

ConditionalType

Figure 3‑24: Conditional Element

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

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

A Condition element shall specify a Boolean expression as shown in figure 3‑25.

ANDedConditionsType

Figure 3‑25 : Conditional Execution of an Activity or State Machine Transition Guard

A TypeCondition element shall be true if the FirstOperand value is compatible with the type specified by the TypeOperand.

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’.

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’.

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

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

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.

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

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.

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.

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).

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.

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

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

A Call element shall include zero or more ArgumentValue elements, each of which, in turn, carries 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.

## State Machines



Figure 3‑26 : State Machines within a StateMachineSet Element

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

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

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

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

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

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.

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 3‑27 : State Machine Transition Element

Each Transition element shall carry

1. a fromState attribute, identifying the name of the state that this transition starts from;
2. a toState attribute, identifying the name of the state that this transition ends at;

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

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

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

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.

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

An OnParameterPrimitive element shall carry:

1. An interface attribute, identifying the component interface to which the primitive relates;
2. a parameter attribute, identifying the parameter to which the primitive relates;
3. an operation attribute, identifying whether the primitive is for a get or set operation.
4. An optional transaction attribute, permitting the primitive reception to be matched to the corresponding primitive transmission using a string identifier.

The transaction attribute of an OnCommandPrimitive or OnParameterPrimitive element shall be present according to the conditions defined in Table 3‑5.

An OnParameterPrimitive element may, according to the conditions defined in Table 3‑7, include an ArgumentValue element which, in turn, includes a VariableRef element specifying a component variable to receive the value associated with the primitive.

An OnCommandPrimitive element shall carry:

1. an interface attribute, identifying the component interface to which the primitive relates.
2. a command attribute, identifying the command to which the primitive relates.
3. An optional transaction attribute, permitting the primitive reception to be matched to the corresponding primitive transmission using a string identifier.

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.

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.

A Guard element shall specify a Boolean expression as shown in figure 3‑25.

# Constructing an SEDS/XML Instance

## 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.

## 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"?>

## Type Referencing and Matching



Figure 4‑1 : Datasheet name reference structure

NOTES

* All elements shown above have names unique within their containing element.
* Elements tagged with <<start> represent the possible starting contexts for name lookup.
* Elements tagged with <<destination> represent the things that can be looked up.
* Relations tagged with fallback represent alternative places to look for referenced names if they are not present.
* It is possible to reference type and interface definitions from another namespace, but not from another component or interface.

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

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

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

{namespace name} **/ {**name}

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

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

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).

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.

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.

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.

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.

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.

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.

Activity or state machine operations which reference non-literal values shall reference component variables only, not interface parameters.

NOTE – Interface parameters can only be accessed using primitive-based operations.

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}

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.

## External References

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 re-usable 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.

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

${name}

The names of external references may contain alphanumeric characters and the ‘\_’ and ‘.’ symbols.

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 pre-processing tool, or as an intermediate output of the toolchain.

## Primitive Associations



Figure 4‑2 : Primitives that trigger state transition

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

1. a get operation primitive from an interface provided by the component identifying a parameter value read request;
2. a set operation primitive from an interface provided by the component identifying a parameter value write request;
3. a get operation primitive from an interface required by the component identifying a parameter value read indication;
4. 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

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

1. a get operation primitive to an interface provided by the component identifying a parameter value read indication;
2. a set operation primitive to an interface provided by the component identifying a parameter value write indication;
3. a get operation primitive to an interface required by the component identifying a parameter value read request;
4. a set operation primitive to an interface required by the component identifying a parameter value write request.

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.

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

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

1. always be transferred in pairs: one transmitted primitive and one received primitive (in the appropriate order); and
2. be associated using an identical string specified as the transaction attribute.

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:

1. transmitted to a component provided interface;
2. received from a component required interface.

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

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

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

1. from an interface *provided* by the component identifying a command execution request;
2. from an interface *required* by the component identifying a command execution indication.

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

1. to an interface *provided* by the component identifying a command execution indication;
2. to an interface *required* by the component identifying a command execution request.

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.

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.

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

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

## State Machine Operation



Figure 4‑4 : State Machine Concepts

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

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

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

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

If multiple transitions from a state have identical OnEvent or OnTimer 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.

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

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

1. For an external transition, the state machine shall exit the state identified as the fromState of the transition;
2. For all transitions, any activity specified in the Do child element of the transition shall be executed.
3. For an external transition, the state machine shall enter the state identified as the toState of the transition.

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

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

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.

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

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

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

NOTE - This allows explicit specification of initialisation actions.

If the defaultEntryState attribute is present, a default starting state shall be used, which will immediately and unconditionally transitions, with no action, into the specified state.

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

* 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.
* 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.

## Encoding and Decoding

A value is encoded when a value of a non-binary 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 value is decoded when a value of a non-binary data type is used for an incoming argument of a command on an interface, and that argument has the dataUnit attribute set to true.

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

If a scalar value with a valid range is encoded, the encoding specification must cater 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.

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.

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.

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.

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.

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.

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.

1. Electronic Data Sheet for Onboard Devices Implementation Conformance Statement Proforma  
     
   (Normative)
   1. Introduction

This annex provides the Implementation Conformance Statement (ICS) Requirements List (RL) for implementation of the SEDS, CCSDS 876.0-R-0, December 2015. 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 a 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.
  1. Notation

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

Status Symbols

|  |  |
| --- | --- |
| 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 |

* 1. Referenced Base Standards

The base standards references in the RL are:

* Electronic Data Sheet for Onboard Device – this document.
  1. Generation Information
     1. Identification of ICS

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

* + 1. Identification of Implementation Under Test (IUT)

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

* + 1. 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) |  |

* + 1. 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 non-conforming. | Yes                No |

* + 1. 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 non-compliance.

The implementers affected by this RL are writers of software that reads and interprets electronic data sheets for use in computer-assisted engineering.

* 1. General/Major Capabilities

|  |  |  |  |
| --- | --- | --- | --- |
| Service Feature | Reference | Status | Support |
| SEDS Core | 3.2, 3.3, 4.2 | M |  |
| Device Metadata | 3.4 | O |  |
| Namespaces | 3.5 | M |  |
| Data Types | 3.6 | M |  |
| Scalar Data Types | 3.7 | M |  |
| Ranges | 3.8 | M |  |
| Arrays | 3.9 | M |  |
| Composite Data Types | 3.10 | M |  |
| Type Instances | 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 |  |

* 1. 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 | 3.2.2–3.2.3 | M |  |
| User-defined Ontology | 3.2.5 | O |  |

1. Security, SANA, and Patent CONSIDERATIONS   
     
   (Informative)
   1. Security Considerations
      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.

* + 1. 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 [D9].

* + 1. 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.

* + 1. 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.

* + 1. 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.

* 1. 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.

|  |  |
| --- | --- |
| File | Description |
| [seds.xsd](http://beta.sanaregistry.org/r/sois/seds.xsd) | The schema for SOIS Electronic Data Sheets. |
| [seds-core-semantics.xsd](http://beta.sanaregistry.org/r/sois/seds-core-semantics.xsd) | The SOIS Dictionary of Terms in the form of a schema to be included by seds.xsd. |
| [seds.xml](http://beta.sanaregistry.org/r/sois/seds.xml) | A non-normative collection of definitions that can reduce the number of definitions in an electronic data sheet. |
| [sois.0.owl](http://beta.sanaregistry.org/r/sois/sois.0.owl) | The ontology for SOIS Dictionary of Terms. This ontology imports sysml-qudv-si-sois.owl. |
| [sysml-qudv.owl](http://beta.sanaregistry.org/r/sois/sysml-qudv.owl) | The definition of quantities, units, dimensions, and values. |
| [sysml-qudv-si.owl](http://beta.sanaregistry.org/r/sois/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](http://beta.sanaregistry.org/r/sois/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. |

Table B2-1 SANA Registry Content

* 1. Patent Considerations

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

1. 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 Virtualisation 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 |

1. 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] *Spacecraft Onboard Interface Services—Device Data Pooling Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 871.1-M-1. Washington, D.C.: CCSDS, November 2012.

[D4] *Spacecraft Onboard Interface Services—Subnetwork Packet Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 851.0-M-1. Washington, D.C.: CCSDS, December 2009.

[D5] *Spacecraft Onboard Interface Services—Subnetwork Memory Access Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 852.0-M-1. Washington, D.C.: CCSDS, December 2009.

[D6] *Spacecraft Onboard Interface Services—Subnetwork Synchronisation Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 853.0-M-1. Washington, D.C.: CCSDS, December 2009.

[D7] *Spacecraft Onboard Interface Services—Subnetwork Device Discovery Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 854.0-M-1. Washington, D.C.: CCSDS, December 2009.

[D8] *Spacecraft Onboard Interface Services—Subnetwork Test Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 855.0-M-1. Washington, D.C.: CCSDS, December 2009.

[D9] *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.

1. 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)"*/>

<Namespace 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>

</Namespace>

</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. Note 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.