The Consultative Committee for Space Data Systems

## Draft Recommendation for Space Data System Standards

# SPACECRAFT ONBOARD INTERFACE SERVICES—XML SPECIFICATION FOR ELECTRONIC DATA SHEETS FOR ONBOARD DEVICES



## PROPOSED DRAFT RECOMMENDED STANDARD

## CCSDS 876.0-R-0

## PROPOSED RED BOOK

**March 2015**

# AUTHORITY

| | |
|---|---|
| Issue: | Proposed Red Book, Issue 0 |
| Date: | March 2015 |
| 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


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

# STATEMENT OF INTENT

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

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

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

- 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.

- 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.

- 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.


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

# 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.


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

# PREFACE

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

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

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

# DOCUMENT CONTROL

| Document | Title | Date | Status |
|---|---|---|---|
| CCSDS 876.0-R-0 | Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets for Onboard Devices, Proposed Draft Recommended Standard, Issue 0 | March 2015 | Current draft |

# CONTENTS

# CONTENTS (continued)

# CONTENTS (continued)

# 1 INTRODUCTION

## 1.1 PURPOSE AND SCOPE OF THIS DOCUMENT

This document is one of a family of documents specifying the Spacecraft Onboard Interface Services (SOIS)-compliant service 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 over the SOIS Subnetwork Services.

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 and protocols used to access the data interfaces.

## 1.2 APPLICABILITY

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

## 1.3 RATIONALE

SOIS provide service interface specifications in order to promote commonality of functionality amongst systems implementing well-defined services. These interfaces do not dictate implementation of interfaces or protocols supporting the services.

## 1.4 DOCUMENT STRUCTURE

This document has several major sections:

– Section 0, this section, contains administrative information, definitions and references.

– Section 2 provides a very brief overview of Electronic Data Sheets for onboard devices. It also provides a very brief overview of XML and the justification for an integrated SEDS/XML schema set.

– 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, one normative and four informative annexes are provided:


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

- 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 instructions on where to find the schema set referenced in this standard on the CCSDS Website. Also provided for illustrative purposes are a number of example instantiations of SEDS.

## 1.5 CONVENTIONS AND DEFINITIONS

### 1.5.1 DEFINITIONS

#### 1.5.1.1 Definitions from the Open Systems Interconnection Reference Model

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

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

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

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

#### 1.5.1.2 Terms defined in this Recommended Standard

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

**application**: Any component of the onboard software that makes use of the Device Virtualisation Service or the Device Access Service. This includes flight software applications and higher-layer services.

**device**: A real hardware component of the spacecraft, such as a sensor or actuator, or a single register within such a component.

**timestamp**: The time associated with a value.

NOTES

1 The format of a timestamp is implementation-specific.

2 The timestamp may indicate the time the value was generated by the device, emitted by the device, or acquired by the service. This is implementation-specific.

**value**: Formatted unit of data that is acquired from or used as a command to a device.

**value identifier**: Abstract identification of a value.

NOTE – The format of a value identifier is implementation-specific.

**virtual device identifier**: Abstract identification of a device.

NOTE – The format of a Virtual Device Identifier is implementation-specific.

## 1.6 NOMENCLATURE

### 1.6.1 NORMATIVE TEXT

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

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

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

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

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

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

### 1.6.2 INFORMATIVE TEXT

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

– Overview;

– Background;

– Rationale;

– Discussion.

## 1.7 REFERENCES

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

[1] *Spacecraft Onboard Interface Services—Device Access Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 871.0-M-1. Washington, D.C.: CCSDS, March 2013.

[2] *Spacecraft Onboard Interface Services—Device Virtualization Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 871.2-M-1. Washington, D.C.: CCSDS, March 2014.

[3] *Spacecraft Onboard Interface Services—Specification for Dictionary of Terms for Electronic Data Sheets for Onboard Components*. Issue 0. Proposed Draft Recommendation for Space Data System Practices (Proposed Red Book), CCSDS 876.1-R-0. Washington, D.C.: CCSDS, March 2105.

[4] 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/.

[5] 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/.

[6] 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/.

[7] 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/.

[8] 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/.

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

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

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

[12] *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.

[13] *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.

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

[15] *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.

[16] "Longitudinal Redundancy Check." Wikipedia. http://en.wikipedia.org/wiki/Longitudinal_redundancy_check.

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

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

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

NOTE – Informative references are contained in annex D.

# 2 OVERVIEW

## 2.1 CONTEXT

The SEDS is defined within the context of the overall SOIS architecture (reference [D2]) as the format of information in a data sheet for an onboard device accessed using the Command and Data Acquisition services of the Application Support Layer and the services of the Subnetwork Layer, as illustrated in figure 2-1.

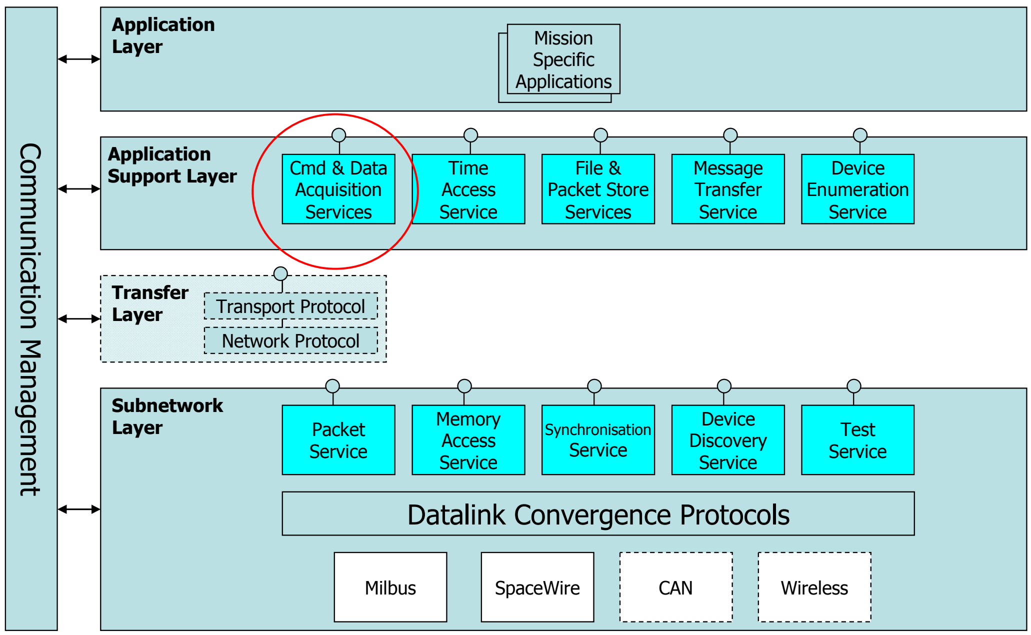



**Figure 2-1: Command and Data Acquisition Services Context**

The relationship between the services of the other Command and Data Acquisition services is illustrated in figure 2-2.

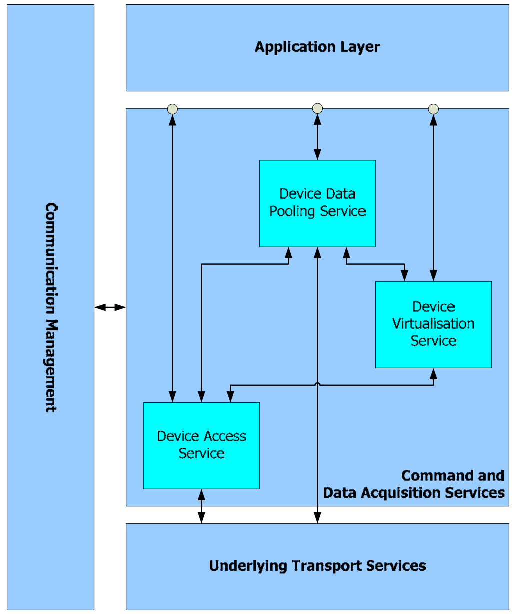



**Figure 2-2: Relationship between Command and Data Acquisition Services**

The Device Access Service (DAS) (reference [1]) provides applications with raw interfaces to simple onboard hardware devices such as sensors and actuators, abstracted from the subnetwork protocols used for exchanging data with the devices. The Device Virtualisation Service (DVS) (reference [2]) provides applications with functional interfaces to devices, abstracted from the protocols used for accessing the devices and the data encodings used in those protocols. The Device Data Pooling Service (DDPS) (reference [D3]) provides a standard interface that enables applications to access pooled data acquired from devices, without explicitly requesting an acquisition from the real device.

The Subnetwork Layer provides standard services mapped onto subnetwork-specific protocols

– to send and receive discrete packets (reference [D4]);

– to access remote memory (reference [D5]);

– to synchronise (reference [D6]) with the subnetwork; and

– to discover (reference [D7]) and test (reference [D8]) devices on the subnetwork.

## 2.2 PURPOSE AND OPERATION OF SOIS ELECTRONIC DATA SHEETS FOR ONBOARD DEVICES

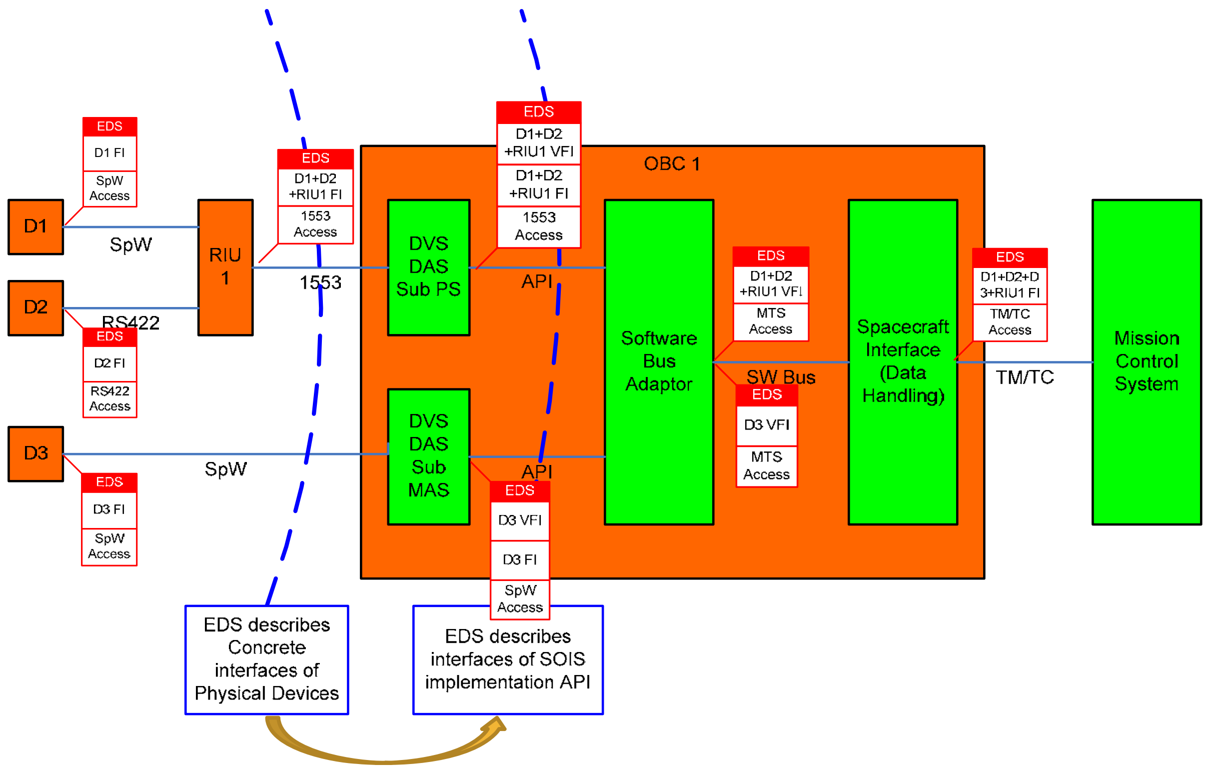



**Figure 2-3: Data Interfaces Described by SEDS**

A SEDS is intended to be a machine-understandable mechanism for describing devices which may be accessed using the SOIS Command and Data Acquisition Services, as identified in figure 2-3 and more fully described in the SOIS Green Book (reference [D2]).

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

a) generating human-readable documentation;

b) specifying interfaces to the device;

c) automatically generating software implementing DVS and DAS for the device;

d) automatically generating software interfacing the device, or device-specific DVS and DAS implementations to an onboard software bus;

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

f) transforming the functional interface, as provided by DVS, into telemetry suitable for processing by a command and data handling system onboard and on the ground;

g) 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:

a) the functional or virtual interface to the device which can be accessed using the DVS service interface;

b) the physical interface to the device which can be accessed using the DAS service interface;

c) the Device Abstraction Control Procedure (DACP) which maps the virtual device interface onto the physical interface;

d) the Device-specific Access Protocol (DAP) which maps the physical device interface onto the SOIS subnetwork services appropriate for the device;

e) information specifying the use of the subnetwork by the device and any constraints placed on the subnetwork;

f) ancillary information.

Each interface (virtual and physical) may actually be composed of multiple interfaces, 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.

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 common Dictionary of Terms (DoT) (reference [3]) 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 common DoT are insufficient, a data sheet author may utilise an additional custom 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.

In addition to this, the SEDS schema, the rules against which SEDS documents are validated, is itself designed to be extensible. Using this mechanism the data sheet language may be extended in the future to accommodate unanticipated data, protocol, and access method patterns. It is likely that such future extensions will include the addition of device-level information not directly relevant to device communications, such as physical device characteristics. This information is currently supported through a generic mechanism for specifying 'metadata' with associated semantics.

## 2.3 USE OF W3C RECOMMENDATIONS

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

a) XML—The Extensible Markup Language (reference [4]) is used to mark up data sheet documents in a machine-readable manner.

b) XSD—The XML Schema Definition language (references [5] and [6]) is used to specify valid construction rules for data sheet documents. It should be noted that version 1.1 of the XSD recommendation is used.

c) XInclude—To permit the construction of data sheet documents from multiple files, some of which may represent standardized data sheet elements, support for the XML Inclusions recommendation (reference [7]) is expected of applications processing SEDS documents.

d) OWL/RDF—In some cases a data sheet author may wish to specify a custom dictionary of terms. This may be accomplished by accompanying the data sheet document with a dictionary of terms document specified according to the Web Ontology Language and using the syntax of the Resource Description Format (reference [8]).

## 2.4 RELATIONSHIP TO XTCE

Early draft versions of the SEDS schema directly referenced some elements of the CCSDS XML Telemetric and Command Exchange (XTCE) standard (reference [D9]). These direct references are no longer present, making the schema entirely standalone.

XTCE remains a historical influence on certain concepts, including the schema naming and construction conventions used.

## 2.5 PRINCIPLES OF AN ELECTRONIC DATA SHEET

### 2.5.1 COMPONENTS

The core element of a SEDS is a **component**. A component is a reusable element of the description of the operation of a device. The device operation corresponding to the DAP, which is an operation to be provided as part of the SOIS DAS, is represented as a component. Similarly, the more abstract device operation corresponding to the DACP, which is an operation to be provided as part of the SOIS Device Virtualisation Service, is also represented as a component. A data sheet therefore usually contains two components, a DAS component and a DVS component, together describing the functionality of the device. Each of these components is an instance of a **component type** described elsewhere in the data sheet. The DVS component, implementing the DACP, will be dependent on the DAS, implementing the DAP. DAS, in turn, is dependent on the SOIS subnetwork services.

### 2.5.2 INTERFACES

The functions of a component are exposed through one or more **interfaces** provided by the component. An interface comprises **parameters** and **commands** which may be accessed using the service interfaces provided for DAS and DVS. The interfaces provided by a component are specified as part of the component type by referencing **interface types**. This permits the types of useful interfaces to be standardised and then provided by component types as appropriate to the functionality offered by a device. To permit the specialisation of existing (possibly standard) interfaces, interface types may extend other interface types by adding parameters and/or commands. Where a component has a dependency on another component, such as the dependency a DVS component has on a DAS component, this is explicitly captured by the component requiring access to an interface of a specific type.

When the commands and/or parameters described by the interface are accessed using the DAS or DVS services, various identifiers are used:

a) Physical device identifiers are used by DAS to uniquely identify physical devices, these correspond to DAS components as described by a SEDS.

b) Virtual device identifiers are used by DVS to uniquely identify virtual devices, these correspond to DVS components as described by a SEDS.

c) Value identifiers are used by both DAS and DVS to identify a parameter (for acquisition or commanding) or a command uniquely within the scope of a device. Each parameter and each command on all of the interfaces provided by a device will have a unique value identifier to permit its use via the DAS or DVS service interface (as appropriate).

The generation and mapping of device identifiers onto SEDS elements, such as components, interface parameters and interface commands, is implementation-specific. The level of information provided by a SEDS is intended to be sufficient to automatically generate a complete implementation of the DVS and DAS layers.

Interface dependencies can go outside the scope of an EDS to interfaces provided by other processes. When designing a space vehicle, the designers match provided interfaces of components to required interfaces of other components. Tool-chain software products can check this feature of a design.

Interface types are extensible through inheritance. It is possible to define a new interface type A which inherits from interface type B. A will then have all of the parameters and commands specified by B and may add further parameters and commands. For simplicity, it is not possible to remove or modify (e.g., overload) inherited parameters or commands. An interface type may choose to inherit from more than one other interface type, in which case the set of parameters and commands specified by the interface is the union of those offered by all of the interface types it inherits from.

It is possible to make interface types which are **generic** across multiple types. This is a similar concept to generics in Ada, Java, or C#, and templates in C++. When an interface type is defined, one or more generic types may be defined. Parameters and command arguments may then be defined using those generic types. When the interface type is instantiated (used), for example by a component type, a mapping must be specified for each generic type to 'replace' it with another data type. This usually maps the generic type to a concrete type which is defined elsewhere in the data sheet.

When a generic interface type is defined, a condition may be attached to each generic type, specifying that the type must have a particular parent type. This permits a component type using the interface to make assumptions about how the type will behave, even before it has been bound to a concrete type.

### 2.5.3 DATA TYPES

Each parameter on an interface or component, or argument to a command, has a type. **Data types** are defined separately and may be shared, re-used, and standardised where appropriate. The scalar types are based on those provided by XTCE: arbitrary **binary** objects, **Booleans**, **enumerations**, **floats**, **integers**, and **strings**.[1] Composite types are fixed and variable-length **arrays**, plus records or structured types referred to as **containers**. Containers have a list of named **entries** representing their contents, plus **constraints** that allow incoming binary data of unknown type to be classified into the matching container.

In all cases the encoding and range can be specified, where encoding means the way that data is represented when transferred to/from the device and range is its valid value range. Additionally, machine readable **semantics** may be associated with a data type, which utilises terms from the Dictionary of Terms. It is also possible to utilise additional terms by creating an additional dictionary to accompany the data sheet using the Web Ontology Language in Resource Description Format syntax (OWL/RDF). The terms in this dictionary can then be referenced from the data sheet using Uniform Resource Identifiers (URIs).

When named data types are referenced directly by other parts of the schema (e.g., parameter definition), the concept of **type instance** is used to allow aspects of the named data type to be customised for that use. This effectively defines a new, anonymous data type in line with the referencing declaration, which references the named data type as its base type.

[1] Unlike XTCE, absolute and relative times are not distinct data types; they are just semantics that can be attached to any appropriate type.

This allows, for example, parameters to be given individual ranges or encodings without requiring corresponding individual data type definitions.

### 2.5.4 COMMAND ARGUMENTS

Commands on an interface can have **arguments** associated with them. Each argument is typed, just like a parameter. Command arguments also have an associated **mode**[2] which specifies whether the argument is to be used to transfer a value to be used by the command (an 'in' mode argument), a value produced by the command (an 'out' mode argument), or a value which is used and then modified by the command (an 'inout' mode argument).

### 2.5.5 PRIMITIVES

The data transfer associated with an interface parameter, or the invocation of a command, is modelled using the basic OSI service **primitives** which are used through the SOIS standards. For example, the acquisition (reading) of the value of a parameter on an interface provided by a component is modelled as the transmission of a parameter get operation **request primitive** to that interface. At some point later the interface will issue a get operation **indication primitive** which contains the parameter value. This pair of request and indication primitives, one transmitted and one received, forms a **transaction**. There are several cases, shown in table 3-5, where a transaction consists of a single primitive. This occurs where the underlying device provides information, such as a telemetry packet, without its being explicitly individually requested.

In this case one or more provided interface parameters may be updated by the component in response to the receipt of data from the device. This asynchronous update can then be passed, again asynchronously, to any components which require the interface. Parameters and commands which operate this way are marked as **async** and cause the transmission of get operation indication primitives without an originating request primitive. Attempting to issue a request primitive for an asynchronous parameter is invalid.

Any further details of the implementation of asynchronous primitives, including initialisation and update registration (if required), is not defined for DVS or DAS services and is consequently outside the scope of SEDS.

### 2.5.6 ACTIVITIES AND STATE MACHINES

The function of the device-specific access protocol or the device abstraction control protocol forms the **implementation** of a component type. The operation of these protocols is specified as a set of sequential, stateless **activities**, each of which can be triggered by a **state machine**. This structure matches the way in which protocols are often specified and cleanly separates stateless and stateful descriptions, making it easier to analyse device operation and generate

[2] The term 'mode' here is used as in programming languages (for example, Ada). This should not be confused with a device mode or any other state-related condition.

code to interact with the device. State machine transitions are typically triggered by the reception of a primitive, for example, indicating the receipt of a packet from the device. The transition triggered by the packet may depend on the packet contents, which could be matched to an appropriate data type. The triggering of a transition may also cause an activity to be invoked. As part of the activity, a primitive may be transmitted, for example, to send an acknowledgement packet back to the device. This division between the transmission and reception of primitives is strict: state machines receive primitives and do not transmit them; activities transmit primitives and do not receive them. State machine transitions may also be triggered by the passage of time; this is useful for specifying timeouts and for catching error conditions.

For example, a device may use a protocol which requires that a command be sent to the device, and the device will always respond with an acknowledgement. The corresponding state machine will have an idle state and a ‘waiting for acknowledgement’ state. The transition away from the idle state will be in response to the need to transmit a command and will typically trigger an activity which transmits the command. There would be two possible transitions from the ‘waiting for acknowledgement’ state: one in response to the receipt of a valid acknowledgement, and a second based on a timeout period. The use of the second transition might trigger an activity to handle the error condition.

Activities and state machines can also interact by sharing data using **component variables**. These are parameters which are contained entirely within the implementation of a component type. The value of a component variable may be exposed through an interface by **mapping** the component variable onto an interface parameter. This parameter mapping is equivalent to creating a pair of state machine/activities which respond to parameter get and set operation requests with appropriate acknowledgement and returning or modifying the component variable. An activity can also be parameterised by requiring arguments. This permits the reuse of activities within one, or across multiple, component types.

### 2.5.7 NAMESPACES

The various elements of a SEDS are designed for reuse. To facilitate this, data types, interface types, and component types are declared in **namespaces**. This permits the type declaration to be qualified in a meaningful way, related types to be collected together, and collections of types to be helpfully encapsulated.

# 3 BASIC STRUCTURE OF THE SEDS/XML SCHEMA

## 3.1 OVERVIEW

This section describes the structure of an electronic data sheet, as required by the SEDS schema. 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.

## 3.2 ELECTRONIC DATA SHEETS AND THE ASSOCIATED SCHEMA

**3.2.1** 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.

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

**3.2.3** Where more than one XML file is used XInclude (reference [7]) shall 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.

**3.2.4** 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.

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, with the schema located at http://sanaregistry.org/xxx/seds/seds-1.0.xsd.

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 [3]).

**3.2.5** A SEDS document can make reference to one or more custom dictionaries of terms. These shall be accompanying XML documents each of which is an RDF/OWL instance (cf. reference [8]).

## 3.3 SEDS/XML BASIC STRUCTURE

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

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

NOTE – This element represents the device which is being described by the SEDS document.

**3.3.3** The `Device` element shall contain zero or one `DVS` elements, zero or one `DAS` elements, zero or one `Subnetwork` elements and zero or one `Metadata` elements. The `DVS` and `DAS` elements are component instances. In this way the document can specify component instances to represent the DVS and DAS interfaces to the device, and the mechanisms used to implement these interfaces (the DACP and DAP respectively).

**3.3.4** The device component instances (the `DAS` and `DVS` elements) are based on `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

NOTE – SEDS elements are frequently based on the `NameDescriptionType`, which is similar to the XTCE `NameDescriptionType` with the `AliasSet` and `AncilliaryDataSet` child elements removed. This means that the element is required to have a `name` attribute; it may also have a `shortDescription` attribute and a `LongDescription` child element. The name is restricted by this regular expression [a-zA-Z][a-zA-Z0-9_]* in `NameType`.

**3.3.5** The device component instances shall each identify the component type it is an instance of using the `type` attribute.

## 3.4 SUBNETWORK

**3.4.1** The `Subnetwork` child element of a `Device` element, if present, specifies the constraints on the configuration of one or more subnetwork interfaces for communication with the device. The `Subnetwork` child element of a `Device` element shall contain one or more child elements, each specifying the constraints for a single subnetwork technology.

NOTES

1 The only valid child element in the current SEDS schema is the `SpaceWireInterface` element, supporting the SpaceWire subnetwork (reference [17]).

2 It is expected that future revisions of the schema will support additional subnetwork types and that alternative child elements of `Subnetwork` will be available to reflect this.

**3.4.2** The `SpaceWireInterface` child element of a `Subnetwork` element shall, if present, contain zero or one of each of the following elements, in the following order:

a) `LinkState`;

b) `ReceiveRate`;

c) `TransmitRate`; and

d) `Protocol`.

**3.4.3** The `LinkState` child element of a `SpaceWireInterface` element shall, if present, describe the default link state of the SpaceWire interface to the device which is one of:

a) `alwaysEnabled`, for a link which is always enabled;

b) `autoStart`, for a link which starts in response to the receipt of NULL characters; and

c) `outOfBandEnable`, if the link is disabled by default and must be enabled by some out-of-band mechanism.

**3.4.4** The `TransmitRate` child element of a `SpaceWireInterface` element describes the various transmit rates that the SpaceWire interface to the device supports and shall, if present, contain one or more `RangeInMbps` elements, each of which specifies a valid range of transmit rates in Mb/s using the `minInclusive` and `maxInclusive` attributes.

**3.4.5** The `ReceiveRate` child element of a `SpaceWireInterface` element describes the various receive rates that the SpaceWire interface to the device supports and shall, if present, contain one or more `RangeInMbps` elements, each of which specifies a valid range of receive rates in Mb/s using the `minInclusive` and `maxInclusive` attributes.

**3.4.6** The `Protocol` child element of a `SpaceWireInterface` element describes the various protocols that the SpaceWire interface to the device supports and shall, if present, contain zero or one of the following elements:

a) `RMAP`;

b) `CPTP`; and

c) any number of `CustomProtocol` elements.

NOTE – These elements represent support for the Remote Memory Access Protocol (RMAP) (reference [14]), the CCSDS Packet Transfer Protocol (CPTP) (reference [19]) and a list of Protocol ID Standard-compliant (reference [18]) custom protocols, respectively.

**3.4.7** An `RMAP`, `CPTP`, or `CustomProtocol` child element of a `Protocol` element shall contain a `LogicalAddress` element containing one or more `ValidRange` elements, each of which specifies a valid range of logical addresses using the `minInclusive` and `maxInclusive` attributes.

**3.4.8** An `RMAP` child element of a `Protocol` element shall contain one or more `AddressRange` elements, each of which specifies the characteristics of a numerically contiguous range of RMAP addresses.

**3.4.9** An `AddressRange` child element of an `RMAP` element shall carry `memoryId`, `minAddress`, and `maxAddress` attributes identifying the numerically contiguous address range described by the element.

**3.4.10** An `AddressRange` child element of an `RMAP` element shall contain zero or one `WriteSupport` elements which contain one or more `WriteType` elements, each of which identifies a type of RMAP write command which is supported in the address range using the `acknowledged` and `verified` Boolean attributes. (See figure 3-1.)

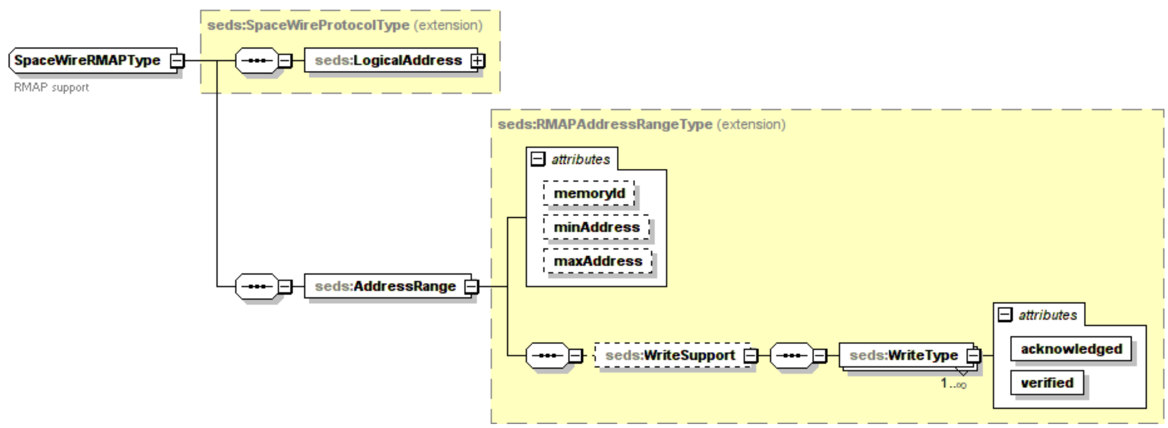



**Figure 3-1: The RMAP Element for a SpaceWire Subnetwork**

**3.4.11** A `CustomProtocol` child element of a `Protocol` element shall carry a `protocolId` attribute specifying the protocol identifier for the custom protocol.

## 3.5 DEVICE METADATA

**3.5.1** The `Metadata` child element of a `Device` element, if present, shall specify a hierarchical set of constant data values, each of which can be associated with machine-understandable semantics.

**3.5.2** The `Category` child element of a `Metadata` or `Category` element, if present, shall specify a categorization or grouping of metadata. The `Category` child element is based on `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.5.3** The `Category` child element of a `Metadata` 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.

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

**3.5.5** The `DateValue`, `FloatValue`, `IntegerValue`, and `StringValue` child elements of either the `Metadata` element or a `Category` element is based on `FieldType`, which is in turn based on `NameDescriptionType`, and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`, one or more `Semantics` elements, and a `value` attribute.

**3.5.6** The value attribute carried by a `DateValue`, `FloatValue`, `IntegerValue`, or `StringValue` element shall contain the value of the metadata.

**3.5.7** The `Semantics` child element of a `Category` element, a `DateValue` element, a `FloatValue` element, an `IntegerValue` element, or a `StringValue` element, if present, may carry a number of attributes specified by the standard dictionary of terms (reference [3]).

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

**3.5.8** The `Semantics` element shall contain zero or more `Term` elements which associate this data type with a term in an accompanying OWL/RDF custom dictionary of terms using a URI. (See figure 3-2.)

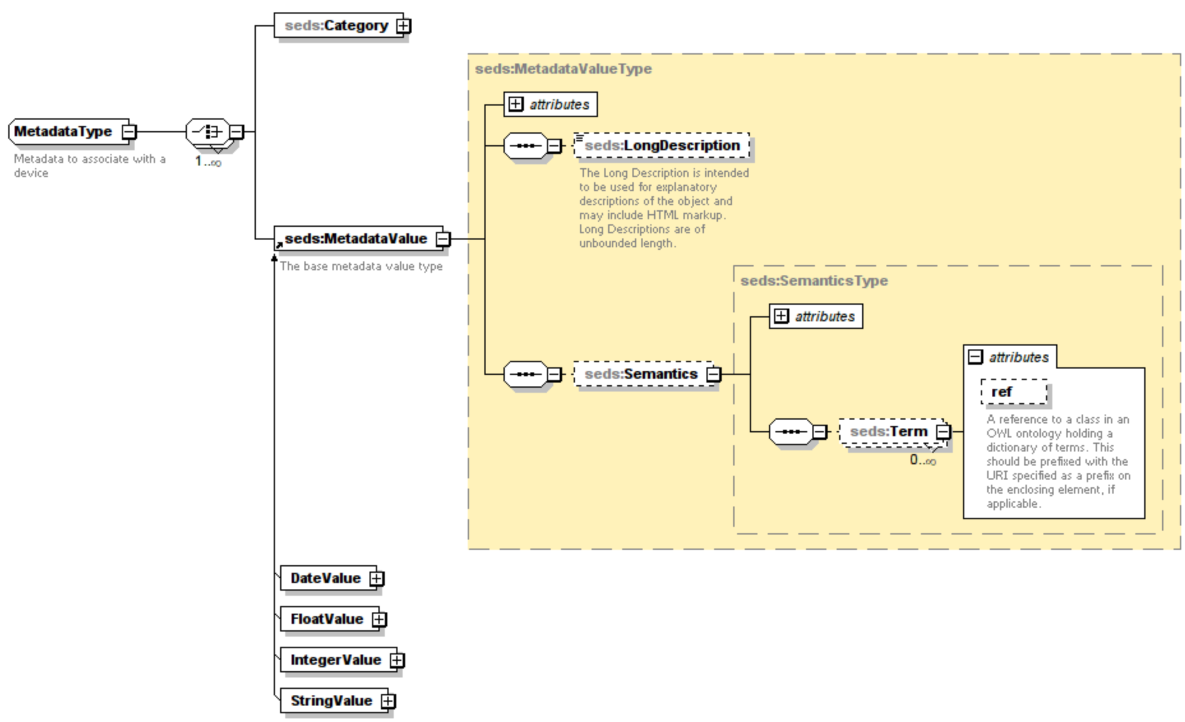



**Figure 3-2: Semantics on Device Metadata Elements**

**3.5.9** 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. (See figure 3-2.)

## 3.6 NAMESPACES AND IDENTIFIER VISIBILITY

**3.6.1** The `DataSheet` element shall contain one or more `Namespace` elements.

**3.6.2** The declaration of types (data types, interface types, and component types) takes place within a namespace. A `Namespace` element shall contain zero or one of the following elements, in the following order `DataTypeSet`, `InterfaceTypeSet`, `ComponentTypeSet`. (See figure 3-3.)

**3.6.3** Additionally, a `Namespace` may declare a hierarchical name. (See figure 3-3.)

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.

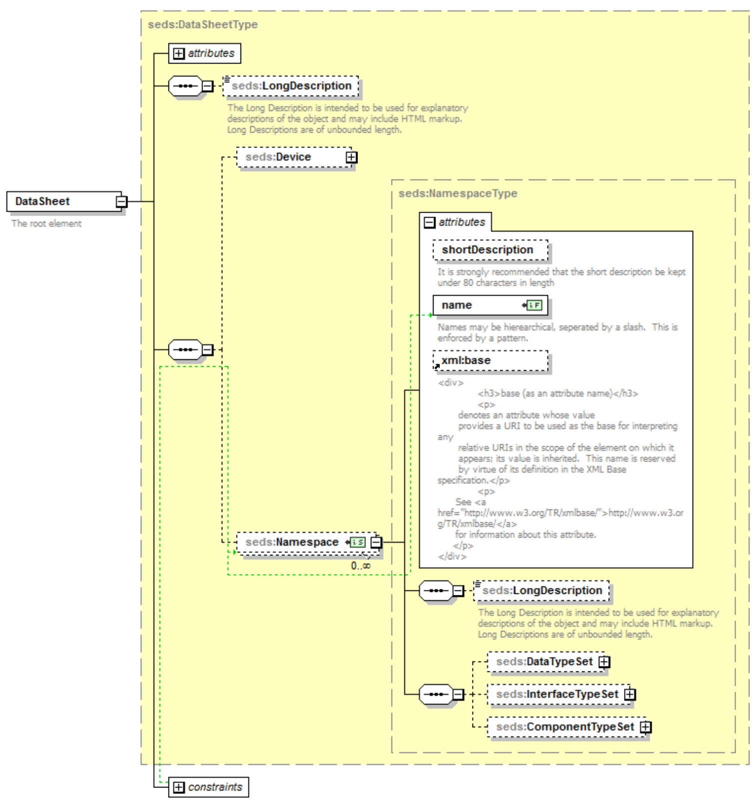



**Figure 3-3: Namespace Elements**

**3.6.4** Each namespace is based on the `NamespaceDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NamespaceDescriptionType`.

NOTE – `NamespaceDescriptionType` is similar to `NameDescriptionType` except that the name attribute is of type `QualifiedNameType`, not `NameType`.

**3.6.5** The name of each `Namespace` element declared shall be unique.

**3.6.6** The declaration of any type shall be identified 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).

NOTES

1 The name without a path is called the local name. The fully qualified name is defined as the namespace hierarchical name in the path form plus the local name separated by a slash character (e.g., a/b/voltage).

2 For references to variables, the dot ('.'), left square bracket ('[') and right square bracket (']') character are allowed local name portion of the fully qualified name to designate fields in a `ContainerDataType` or array indices in an `ArrayDataType`, for example a/b/voltage[20].low is legal.

## 3.7 DATA TYPES

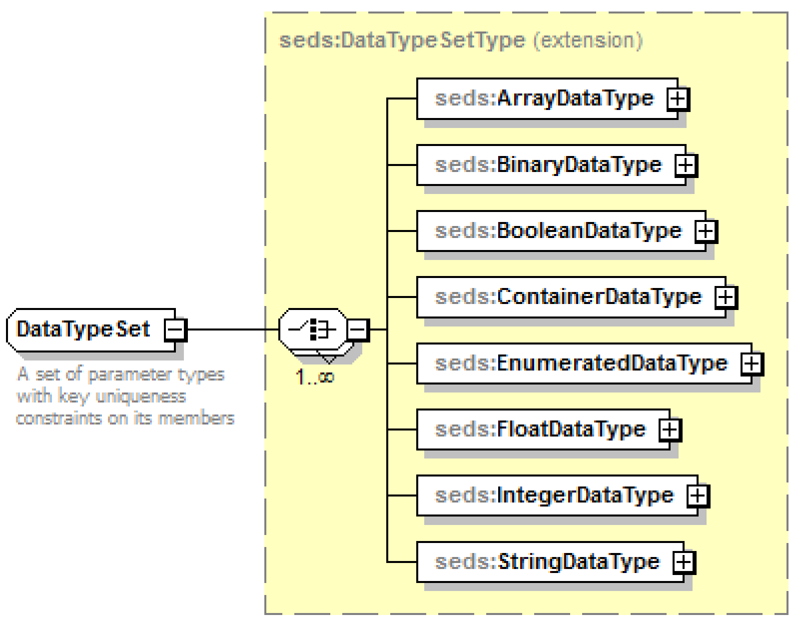



**Figure 3-4: Data Types within a `DataTypeSet` Element**

**3.7.1** The `DataTypeSet` element contained in a namespace shall contain one or more of the following elements:

– `ArrayDataType`;
– `BinaryDataType`;
– `BooleanDataType`;
– `ContainerDataType`;
– `EnumeratedDataType`;
– `FloatDataType`;
– `IntegerDataType`; and
– `StringDataType`.

**3.7.2** Each child element of a `DataTypeSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.7.3** The name of each child element of a `DataTypeSet` element shall be unique.

**3.7.4** It is possible to associate machine-understandable semantics with data types. Each child element of a `DataTypeSet` element shall contain zero or one `Semantics` element which defines the semantics of the data type.

**3.7.5** The `Semantics` element may carry a number of attributes specified by the standard dictionary of terms (reference [3]).

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

**3.7.6** The `Semantics` element shall contain zero or more `Term` elements which associate this data type with a term in an accompanying OWL/RDF custom dictionary of terms using a URI. (See figure 3-5.)

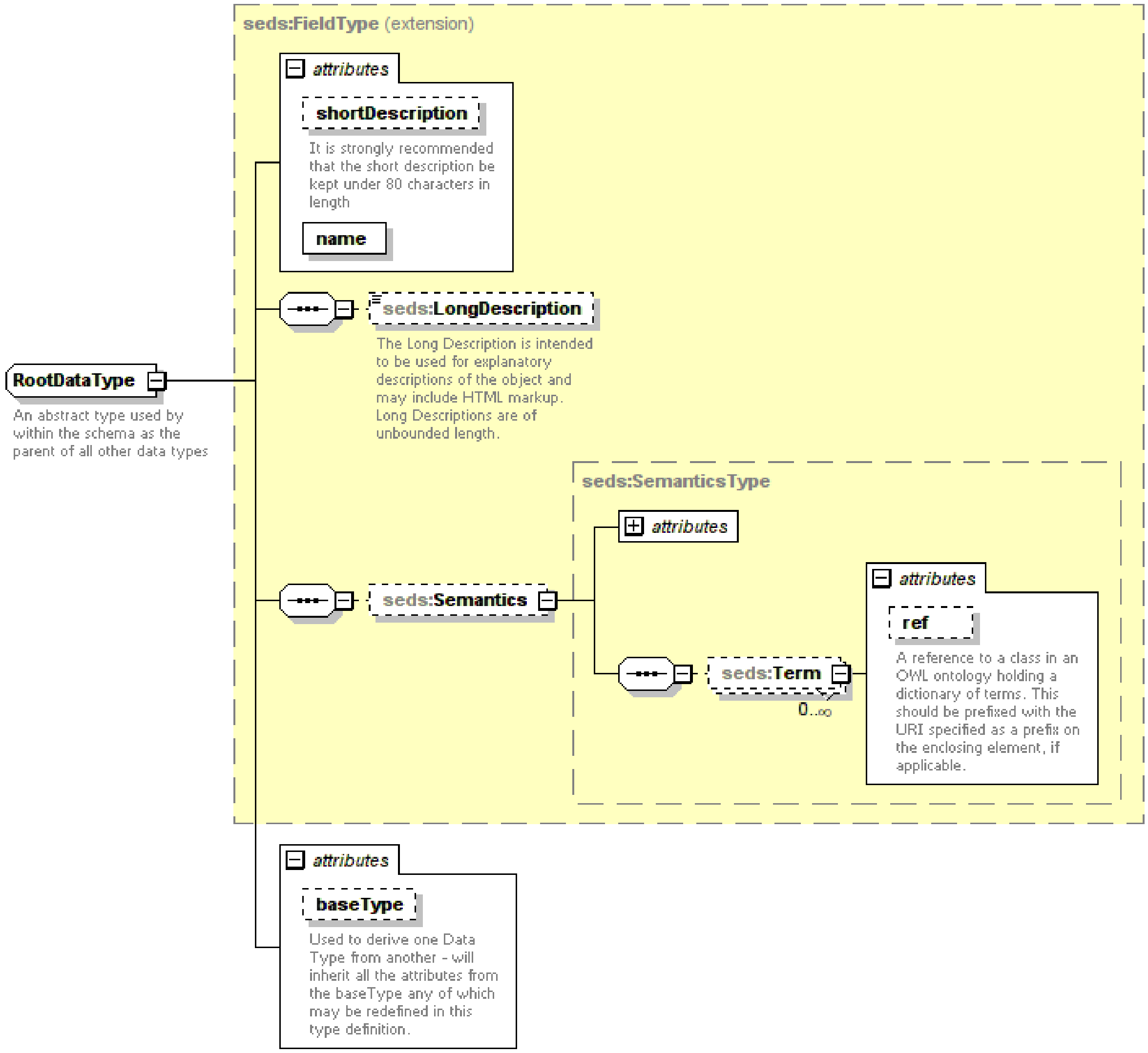



**Figure 3-5: Semantics and Base Types on Data Types**

**3.7.7** The `Semantics` element shall carry zero or one `prefix` attribute which specifies a URI to be used as a prefix on the URIs of all enclosed `Term` elements. (See figure 3-5.)

**3.7.8** It is possible to derive one type from another, where a derived type is more specific than the type it is being derived from (the base type). Each child element of a `DataTypeSet` element may carry a `baseType` element which identifies the type to use a base type. (See figure 3-5.)

## 3.8 SCALAR DATA TYPES

**3.8.1** Scalar data types (i.e., not structured types like arrays or containers) may specify how they are to be encoded, if they are transmitted over a subnetwork. Each `BinaryDataType`, `BooleanDataType`, `EnumeratedDataType`, `FloatDataType`, `IntegerDataType`, or

StringDataType child element of a DataTypeSet shall contain zero or one 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 | BinaryDataEncoding | | |
| BooleanDataType | BooleanDataEncoding | | xs:boolean |
| EnumeratedDataType | IntegerDataEncoding | | xs:string, matching enumeration label. |
| FloatDataType | FloatDataEncoding | PrecisionRange<br>MinMaxRange | xs:float |
| IntegerDataType | IntegerDataEncoding | NumberOfBitsRange<br>MinMaxRange | xs:integer |
| StringDataType | StringDataEncoding | | xs:string |

**3.8.2** Numeric and string data types may specify a range of representable values, which form a constraint on the possible encodings. Each FloatDataType, IntegerDataType, or StringDataType child element of a DataTypeSet shall contain zero or one Range element of a type corresponding to the table 3-1.

**3.8.3** A BinaryDataEncoding, FloatDataEncoding, IntegerDataEncoding, or StringDataEncoding child element of a scalar data type shall carry a byteOrder attribute specifying with a value of bigEndian for values which are to be encoded most significant byte first or littleEndian for values which are to be encoded least significant bytes first.

**3.8.4** A BinaryDataEncoding child element of a scalar data type shall contain a byteSizeInBits attribute which specifies the length of the binary encoding in bits.

NOTE – This allows such data to 'pass-through' the SEDS-defined layers without interpretation.

**3.8.5** A BooleanDataEncoding child element of a scalar data type must carry a sizeInBits attribute which specifies the size, in bits, of the encoded data and may be any positive integer.

**3.8.6** A BooleanDataEncoding child element of a scalar data type may carry a falseValue attribute which specifies the value that corresponds to logical falsehood; if not specified, the default shall be '0'.

**3.8.7** An IntegerDataEncoding child element of a scalar data type may carry an encoding attribute which has a value of:

– unsigned, for an unsigned value;

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

– `twosComplement`, for twos complement;

– `onesComplement`, for ones complement;

– `BCD` for binary coded decimal, where each octet is a decimal digit encoded as binary; and

– `packedBCD`, where each octet contains two decimal digits encoded as binary; if not specified, the default encoding shall be unsigned.

**3.8.8** An `IntegerDataEncoding` child element of a scalar data type may carry a `sizeInBits` attribute which specifies the size, in bits, of the encoded data and may be any positive integer. If not specified, the default value is 8.

**3.8.9** A `FloatDataEncoding` child element of a scalar data type may carry an `encodingAndPrecision` attribute which has a value of either:

– IEEE754_2008_single;

– IEEE754_2008_double

– IEEE754_2008_quad

– MILSTD_1750A_simple; or

– MILSTD_1750A_extended.

NOTE – These represent the supported sizes of IEEE (reference [9]) and MIL-STD-1750A (reference [10]). If not specified, the default encoding is IEEE754_2008_single.

**3.8.10** In addition to the attributes and child elements defined for a scalar data type, a `StringDataType` must carry a `length` attribute which defines the maximum possible length of the string, in characters.

**3.8.11** In addition to the attributes and child elements defined for a scalar data type, 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; if not set, the default shall be 'false'.

**3.8.12** A `StringDataEncoding` child element of a string data type may carry an `encoding` attribute which has a value of either:

– `UTF-8`, the default, specifying Unicode UTF-8 encoding (reference [11]); or

– `ASCII`, specifying US ASCII encoding (reference [12]).

**3.8.13** A `StringDataEncoding` child element of a string data type may carry a `characterSizeInBits` attribute which specifies the size, in bits, of the encoded data and may be any positive integer divisible by 8.

**3.8.14** The `terminationCharacter` attribute of a `StringDataEncoding` element, if present, shall specify the termination character for the string. For example, a termination character of zero (null) is used by C-language strings.

**3.8.15** A `StringDataType` element may have an optional `CharacterRange` element.

**3.8.16** A `CharacterRange` child element of a `StringDataType` must have attributes `min` and `max`, indicating the minimum and maximum character values.

**3.8.17** In addition to the attributes and child elements defined for a scalar data type, an `EnumeratedDataType` shall contain an `EnumerationList` element, consisting of a list of one or more Enumeration elements.

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

**3.8.19** An Enumeration element may have an optional child `Semantics` element.

**3.8.20** The valid attributes and child elements of a `Semantics` child element of an `Enumeration` element, if present, shall be identical to those valid for a `Semantics` child element of any data type element (see 3.7.5, 3.7.6, and 3.7.7).

## 3.9 RANGES

**3.9.1** A `PrecisionRange` child element within a `Range` shall be either `SINGLE` or `DOUBLE`, representing the full supported representation range of the corresponding IEEE float data encodings.

**3.9.2** A `MinMaxRange` child element within a `Range` element shall have an attribute `rangeType`, one of the options listed in table 3-2.

**Table 3-2: `MinMaxRange` Options**

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

**3.9.3** A `MinMaxRange` child element within a `Range` element may have attributes `min` and `max`, whose presence and values shall be consistent with the table 3-2.

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

**3.9.5** A `NumberOfBitsRange` child element within a `Range` element must have an attribute `numberOfBits`, which indicates the number of binary bits logically required to represent the range without loss.

**3.9.6** A `NumberOfBitsRange` child element within a `Range` element may have an attribute `signed`, which if 'true' implies that the range calculation should use signed arithmetic.

NOTE – Table 3-3 gives example values for `NumberOfBitsRange`.

**Table 3-3: Example Values for `NumberOfBitsRange`**

| Interval Notation | Signed | `numberOfBits` |
|---|---|---|
| `[0..255]` | 'false' | 8 |
| `[-128..127]` | 'true' | 8 |
| `[0..1]` | 'false' | 1 |
| `[0..65535]` | 'false' | 16 |
| `[-2.147483648E9..2.147483647E9]` | 'true' | 32 |

## 3.10 COMPOSITE DATA TYPES

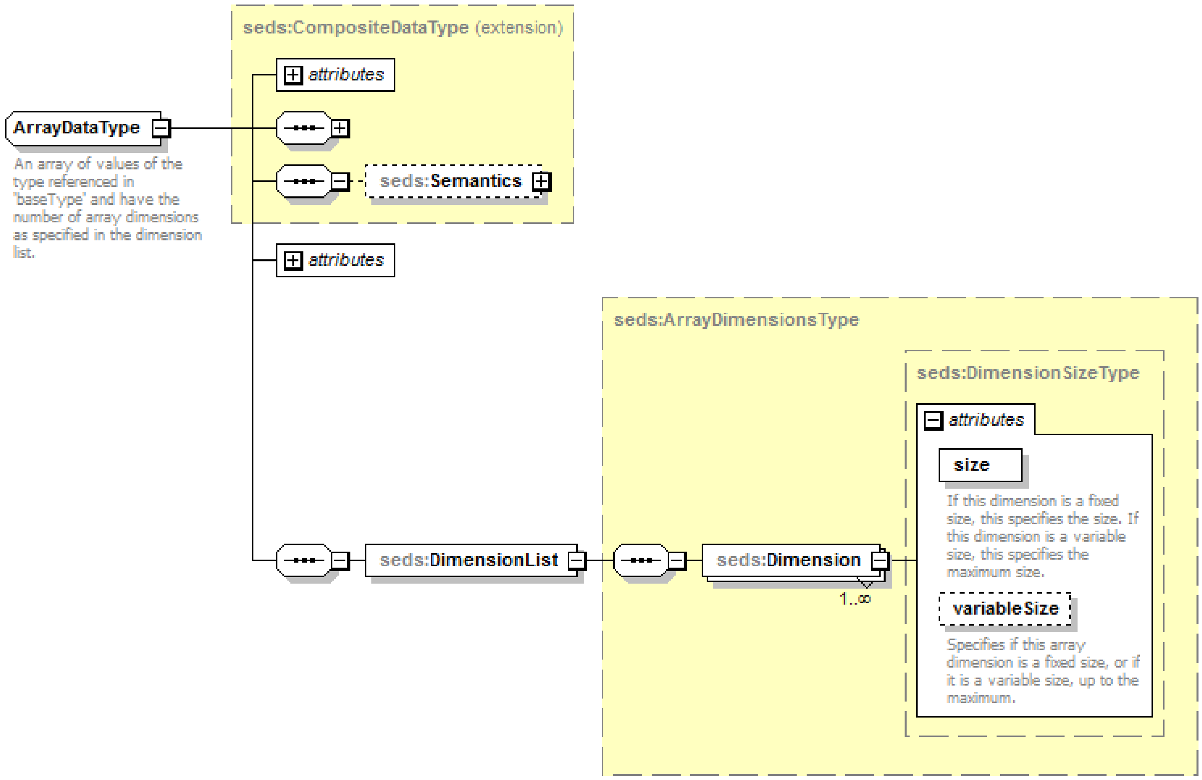



**Figure 3-6: `ArrayDataType` and Dimensions**

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

**3.10.2** A `Dimension` child element of a `DimensionList` element determines the length of the array dimension, in elements, and shall have attribute size, indicating the maximum length, and may have attribute `variableSize`, indicating if the actual length can be shorter than that value.

**3.10.3** A `ContainerDataType` element may carry an `abstract` attribute which, if present and set to 'true', indicates that the container is not to be used directly, only referenced as the base type of other containers.

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

**3.10.5** The `ConstraintSet` element of a `ContainerDataType` element specifies the criteria which apply to the entries of the container type which is the base type of this container in order for the type to be valid. 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`. (See figure 3-7.)

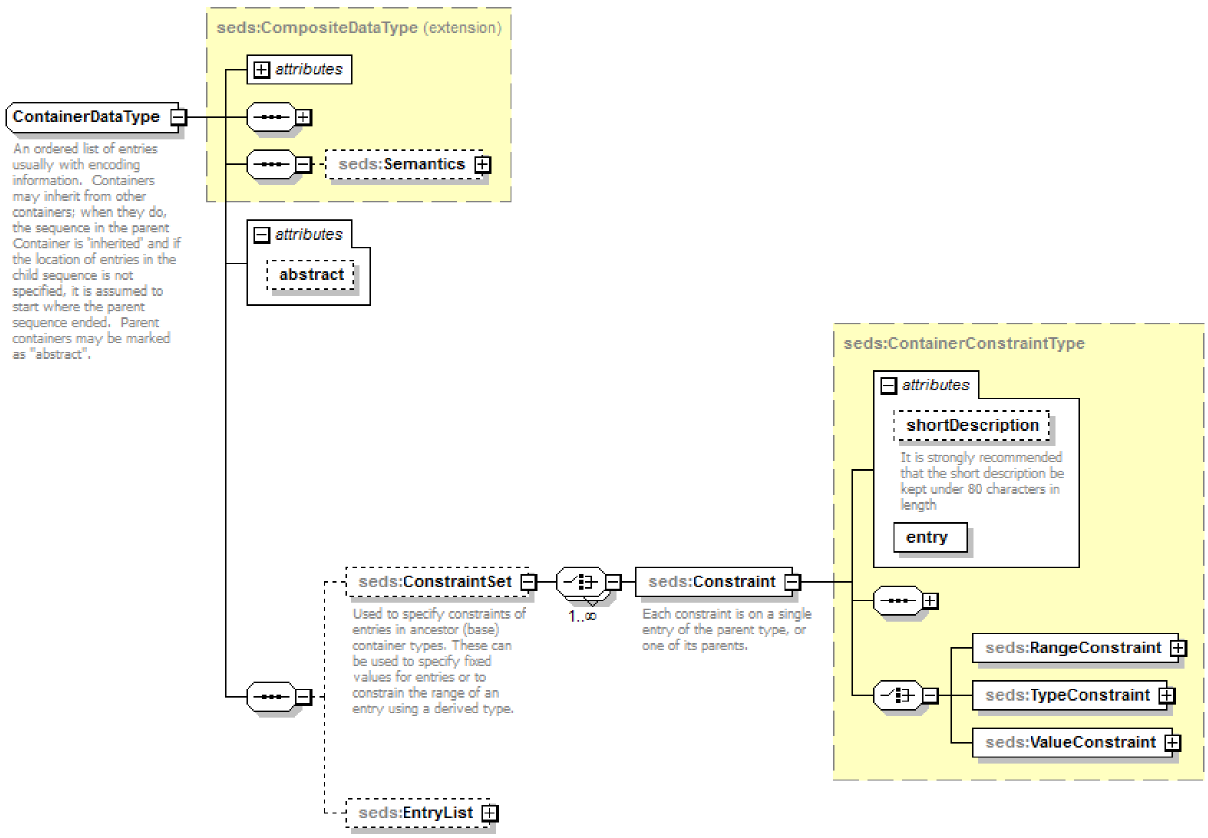



**Figure 3-7: Constraints on a `ContainerDataType`**

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

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

**3.10.8** A `TypeConstraint` child element of a `ConstraintSet` shall have an attribute type, which shall reference a type which has the type of the constrained entry as a base type.

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

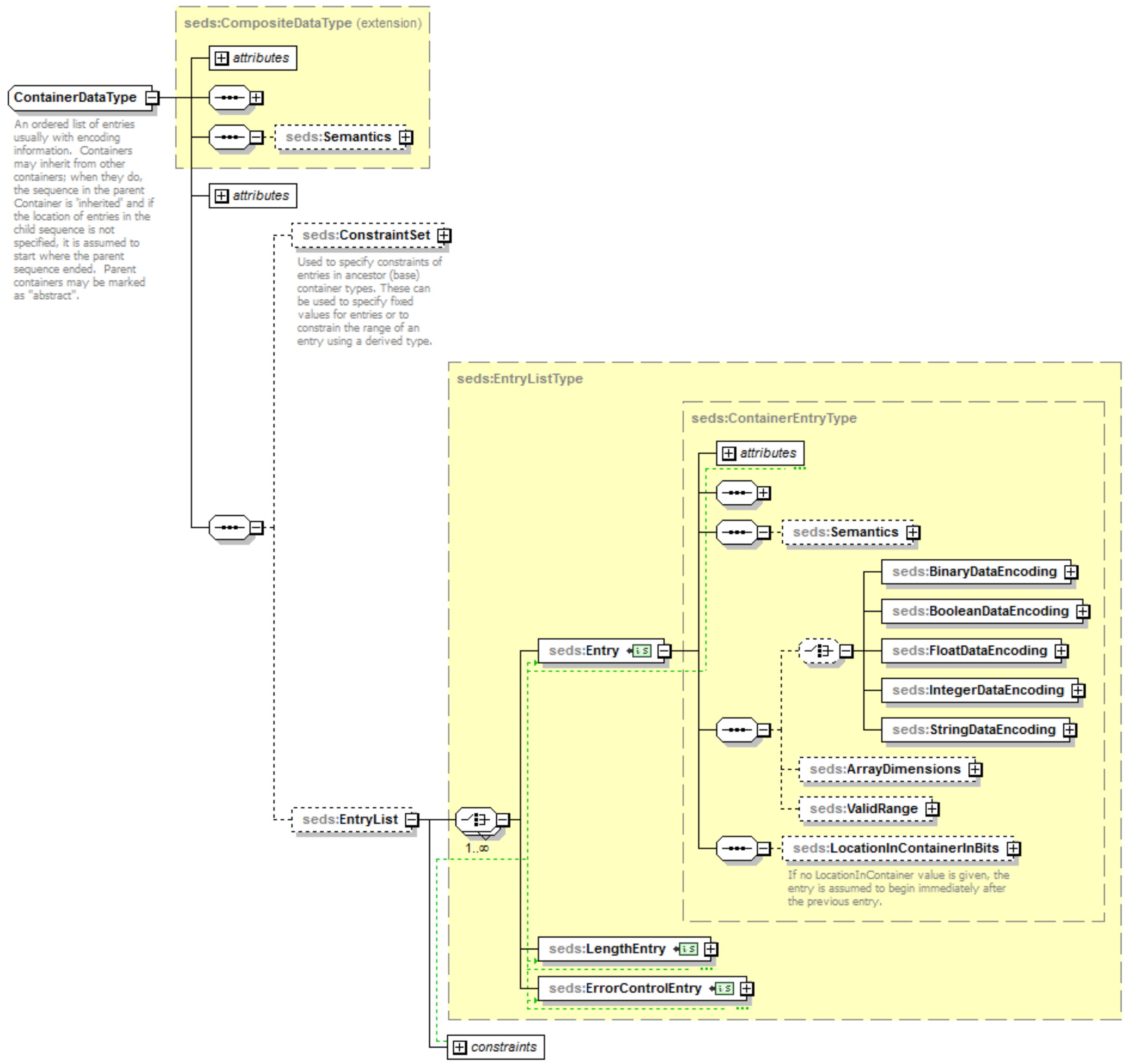



**Figure 3-8: Entries in a Container**

**3.10.10** The `EntryList` element of a `ContainerDataType` element shall contain one or more `Entry`, `LengthEntry`, and `ErrorControlEntry` child elements.

**3.10.11** The `Entry` child element of an `EntryList` element shall have the attributes and child elements associated with a data type instance (see 3.11) with the addition of an optional `fixedValue` attribute, and zero or one of the following element: `LocationInContainerInBits`.

**3.10.12** The `fixedValue` attribute of container entry shall specify, if present, the value to which the container entry should be fixed.

NOTE – The container entry therefore has a constant value and is effectively read-only.

**3.10.13** 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.

**3.10.14** The `LocationInContainerInBits` element of container entry, if present, specifies the location of the entry within the container when encoded and shall carry an offset attribute, specifying the offset, in bits, and an optional `referenceLocation` attribute, which has the possible values `previousEntry`, `containerStart`, and `containerEnd`; if the `referenceLocation` attribute is not present, the value `previousEntry` shall be assumed.

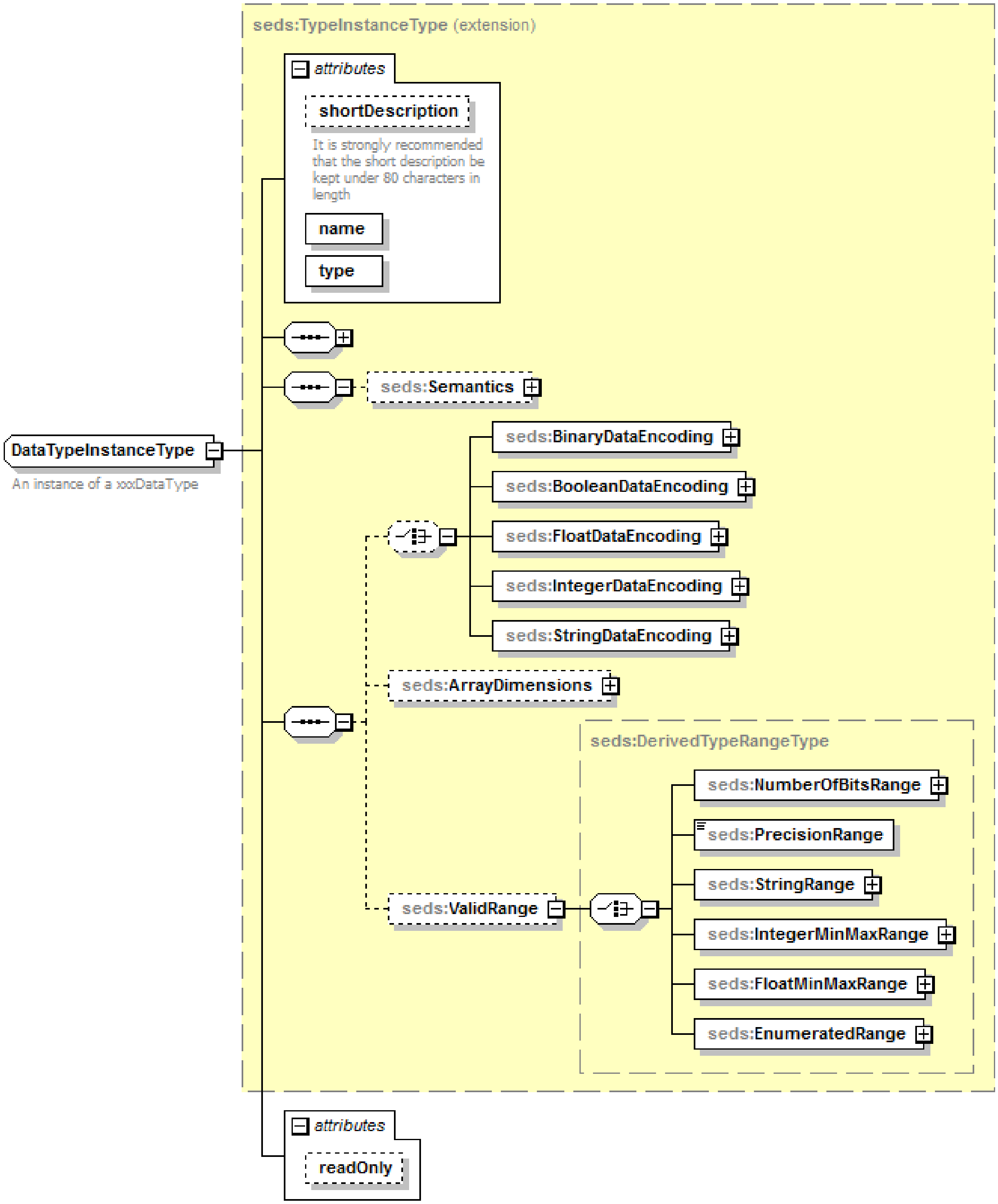
**3.10.15** A `LengthEntry` child element of a container shall specify an entry whose value is constrained, or derived, based on the length of the container in which it is present. As well as a subset of the attributes and elements supported for a regular container entry, it has attributes `coefficient` and `offset`, which are used in the formula: `container length = (entry value * coefficient) + offset`.

**3.10.16** An `ErrorControlEntry` child element of 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 3-4.

**Table 3-4: Error Control Types**

| Value | Description | Reference |
|---|---|---|
| CRC16_CCITT | G(X) = X^16 + X^12 + X^5 + 1 | [13], subsection 4.1.4.2 |
| CRC8 | G(x) = x^8 + x^2 + x^1 + x^0 | [14], clause 5.2 |
| CHECKSUM | modulo 2^32 addition of all 4-octet | [15], subsection 4.1.2 |
| CHECKSUM_LONGITUDINAL | Longitudinal redundancy check, bitwise XOR of all octets | [16] |

## 3.11 TYPE INSTANCES




**3.11.1** A type instance is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

NOTE – 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.

**3.11.2** A type instance shall carry a `type` attribute identifying the name of the data type it is an instance of.

**3.11.3** A type instance shall have zero or one `Semantics` element, zero or one `ValidRange` element, zero or one `ArrayDimensions` element, and zero or one encoding element.

**3.11.4** The `ValidRange` and encoding elements, if present, shall match the `type` attribute according to table 3-1.

**3.11.5** The valid attributes and child elements of a `Semantics` child element of type instance, if present, shall be identical to those valid for a `Semantics` child element of any parameter type element (see 3.7.5, 3.7.6, and 3.7.7).

**3.11.6** The valid child elements of an `ArrayDimensions` child element of type instance, if present, shall be identical to those valid for a `DimensionList` child element of an `ArrayDataType` element (see 3.10.2).

NOTE – This element permits the definition of a parameter which is an array of types at instantiation time; this is often more compact than defining an additional array type.

## 3.12 INTERFACE TYPES

**3.12.1** The `InterfaceTypeSet` element contained in a namespace shall contain one or more `InterfaceType` elements.

**3.12.2** Each `InterfaceType` child element of an `InterfaceTypeSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.12.3** The name of each `InterfaceType` child element of an `InterfaceTypeSet` element shall be unique.

**3.12.4** An `InterfaceType` 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. (See figure 3-9.)

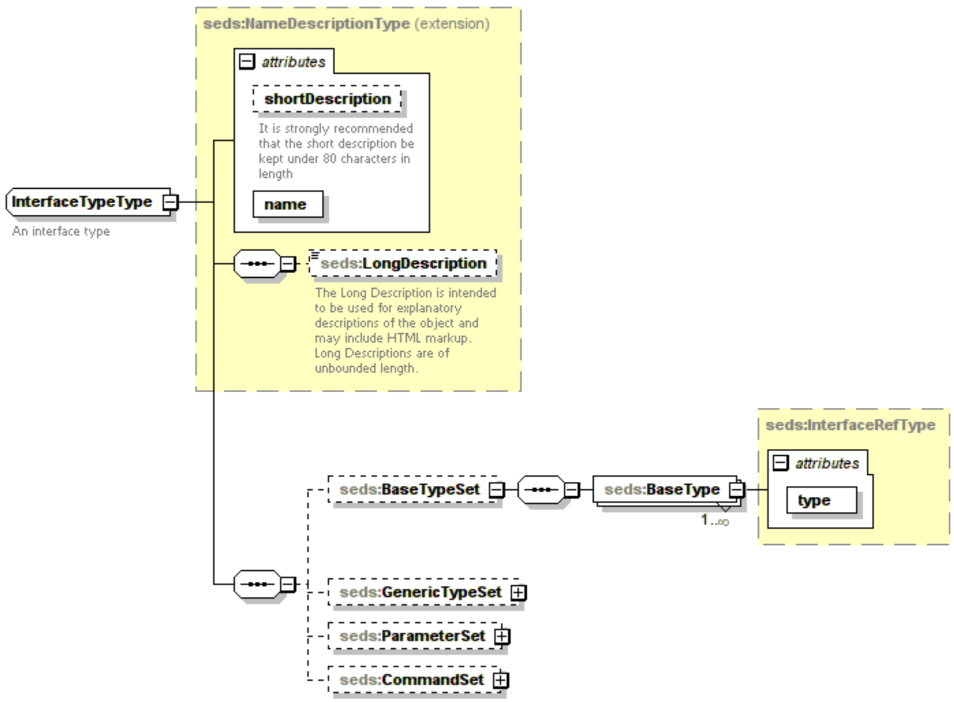



**Figure 3-9: Interface Types**

**3.12.5** Each `BaseType` child element of a `BaseTypeSet` element shall identify one existing interface type which should 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).

**3.12.6** Each `GenericType` child element of a `GenericTypeSet` element specifies a generic type to be used by the interface, and is based on the `NameDescriptionType`, and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

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

**3.12.8** Each `GenericType` child element of a `GenericTypeSet` 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.

**3.12.9** Each `Parameter` child element of a `ParameterSet` element shall have the attributes and child elements associated with a 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.

**3.12.10** The name of each `Parameter` child element of a `ParameterSet` element shall be unique.

**3.12.11** Valid values for the `mode` attribute shall be '`sync`' (the default) or '`async`' indicating how the parameter or command data is updated by the device.

NOTE – A mode of '`sync`' indicates that the parameter value is determined when it is requested through the interface. This may require a query from the actual device. A mode of '`async`' indicates that the device asynchronously issues data which updates this parameter. By marking the parameter as '`async`' on the interface, this asynchronous update is continued across a provided interface to any component requiring the interface.

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

**3.12.13** Each `Command` child element of a `CommandSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`, plus an optional `mode` attribute, identifying the command mode.

**3.12.14** The name of each `Command` child element of a `CommandSet` element shall be unique.

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

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

**3.12.17** The name of each command argument element shall be unique.

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

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

| Interface Element | Options | Argument Modes | Primitive | Transaction |
|---|---|---|---|---|
| Parameter | **sync** | | request<br>indication | yes |
| | **async** | | indication | no |
| Command | **sync** | No **out** or **inout** | request<br>indication | yes |
| | | **inout**, or both **in** and **out** | request<br>indication | yes |
| | | No **in** or **inout** arguments | request<br>indication | yes |
| | **async** | No **out** or **inout** arguments | request | no |
| | | **inout**, or both **in** and **out** | illegal | |
| | | No **in** or **inout** arguments | indication | no |

## 3.13 COMPONENT TYPES

**3.13.1** The `ComponentTypeSet` element contained in a namespace shall contain one or more `ComponentType` elements.

**3.13.2** Each `ComponentType` child element of a `ComponentTypeSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.13.3** The name of each `ComponentType` child element of a `ComponentTypeSet` element shall be unique.

**3.13.4** A `ComponentType` element shall contain, in order,

a) one `ProvidedInterfaceSet` element;

b) one `RequiredInterfaceSet` element

c) zero or one `DataTypeSet` element;

d) zero or one `InterfaceTypeSet` element; and

e) zero or one `Implementation` element.

**3.13.5** The `ProvidedInterfaceSet` and `RequiredInterfaceSet` elements shall each contain zero or more `Interface` elements, each of which identifies a provided or required interface, respectively.

**3.13.6** Each `Interface` child element of a `ProvidedInterfaceSet` or `RequiredInterfaceSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.13.7** The name of each `Interface` child element of a `ProvidedInterfaceSet` or `RequiredInterfaceSet` element shall be unique within the interface set.

**3.13.8** Each `Interface` element shall carry a `type` attribute which identifies the type of the interface.

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

**3.13.10** A `GenericTypeMap` child element of a `GenericTypeMapSet` element, if present, 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.

**3.13.11** The `fixedValue` attribute of a `GenericTypeMap` element shall, if present, specify a fixed value for the generic type. This is equivalent to specifying a data type with a valid range (see `Range` in `IntegerDataType` and `FloatDataType`) which contains only the value specified by the `fixedValue` attribute.

**3.13.12** An `AlternateSet` child element of a `GenericTypeMapSet` element, if 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.

**3.13.13** A `GenericTypeMap` child element of an `Alternate` element specifies a mapping of a generic type to a concrete type and shall be equivalent to the `GenericTypeMap` child element of a `GenericTypeMapSet` element. (See figure 3-10.)

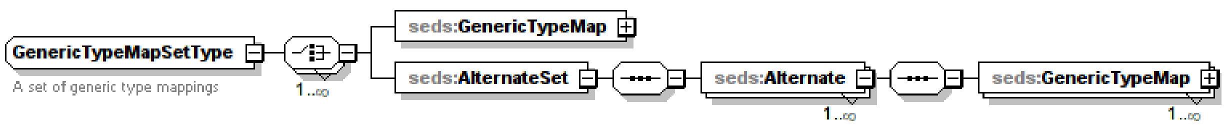



**Figure 3-10: Generic Type Mapping**

**3.13.14** If present, the `DataTypeSet` child element of a `ComponentType` element shall follow identical construction rules as for the `DataTypeSet` child element of a `Namespace` element.

**3.13.15** If present, the `InterfaceTypeSet` child element of a `ComponentType` element shall follow identical construction rules as for the `InterfaceTypeSet` child element of a `Namespace` element.

**3.13.16** Types declared within the `DataTypeSet` and `InterfaceTypeSet` child elements of a `ComponentType` element shall only be visible to descendent elements of the `ComponentType` 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.


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

## 3.14 COMPONENT IMPLEMENTATIONS

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

a) `VariableSet`;

b) `ParameterMapSet`;

c) `ParameterMapActivitySet`;

d) `ActivitySet`;

e) `StateMachineSet`.

**3.14.2** The `VariableSet` child element of an `Implementation` element, if present, shall contain one or more `Variable` elements.

**3.14.3** Each `Variable` child element of a `VariableSet` element shall have the attributes and child elements associated with a type instance (see 3.11) with the addition of an optional `initialValue` attribute identifying the initial value of the variable, and an optional `readOnly` attribute, identifying if the variable is read-only.

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

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

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

**3.14.7** The `ParameterMapSet` child element of an `Implementation` element, if present, shall contain one or more `ParameterMap` elements.

**3.14.8** Each `ParameterMap` child element of a `ParameterMapSet` element shall carry one `parameterRef` attribute and one `variableRef` attribute.

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

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

**3.14.11** The types of the parameters referred to by the `parameterRef` and `variableRef` attributes shall match.

**3.14.12** The `ParameterMapActivitySet` child element of an `Implementation` element, if present, shall contain one or more `ParameterMapActivity` elements.

**3.14.13** Each `ParameterMapActivity` element maps a parameter on a provided interface to a parameter on a required interface using an activity. Therefore a `ParameterMapActivity` 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 parameter with the specified name.

**3.14.14** Additionally, each `ParameterMapActivity` element shall have either:

– one `GetActivity` child element;

– one `SetActivity` child element; or

– one `GetActivity` child element and one `SetActivity` child element.

**3.14.15** The `GetActivity` and `SetActivity` child elements of a `ParameterMapActivity` element, if present, shall specify an activity 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 `Implementation` child element of an `Activity` element (see 3.15.5).

**3.14.16** The `ActivitySet` child element of an `Implementation` element, if present, shall contain one or more `Activity` elements.

**3.14.17** Each `Activity` child element of an `ActivitySet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

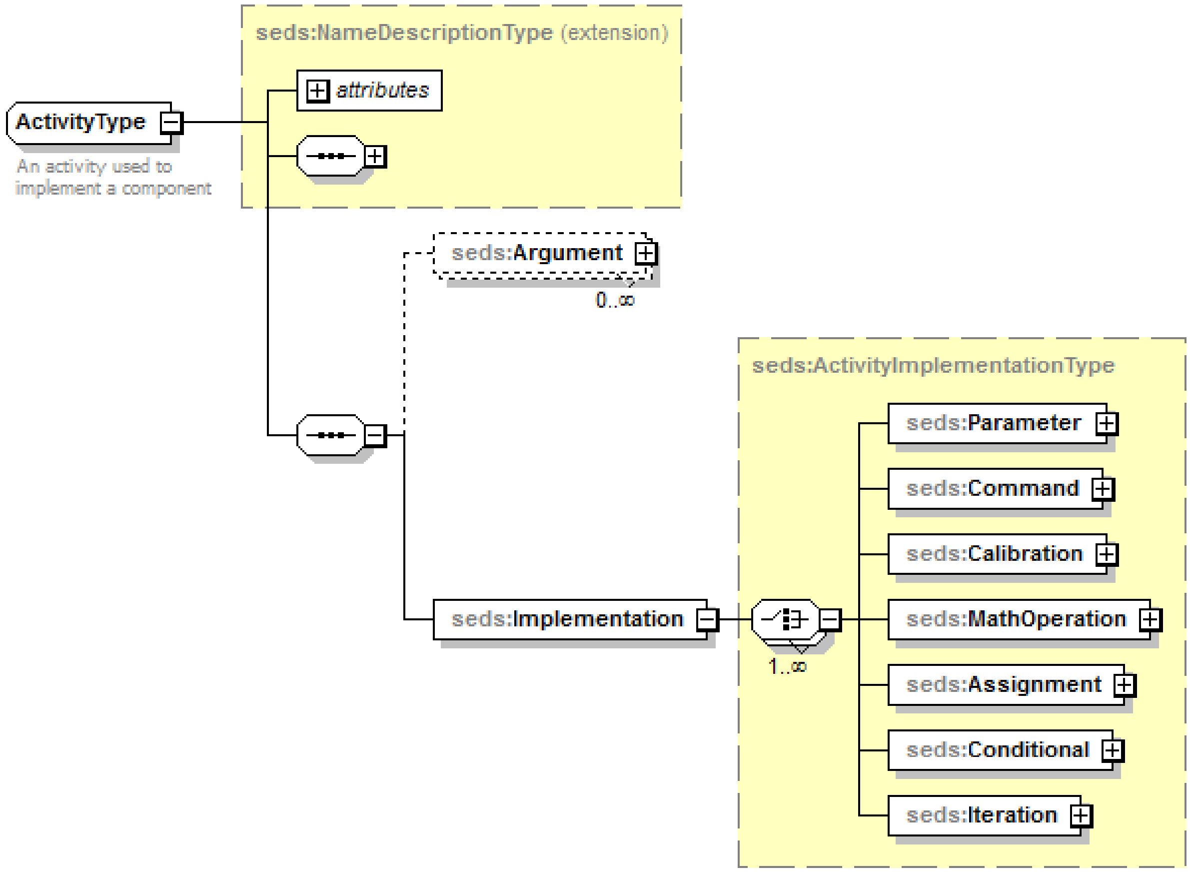
**3.14.18** The name of each `Activity` child element of an `ActivitySet` element shall be unique.

**3.14.19** The `StateMachineSet` child element of an `Implementation` element, if present, shall contain one or more `StateMachine` elements.

**3.14.20** Each `StateMachine` child element of a `StateMachineSet` element is based on the `NameDescriptionType` and shall therefore carry a name (the `name` attribute) together with the other optional attributes and elements of the `NameDescriptionType`.

**3.14.21** The name of each `StateMachine` child element of a `StateMachineSet` element shall be unique.

## 3.15 ACTIVITIES




**3.15.1** Each `Activity` child element of an `ActivitySet` element shall contain zero or more `Argument` elements and one `Implementation` element. `Argument` elements permit the operation of the activity, specified by the `Implementation` element, to be parameterised.

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

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

**3.15.4** Each `Argument` child element of an `Activity` element shall carry a `type` attribute specifying the parameter type of the argument.

**3.15.5** The `Implementation` child element of an `Activity` element shall contain one or more of the following elements:

– `Parameter`;

– `Command`;

– `Calibration`

– MathOperation;

– Assignment;

– Conditional; and

– Iteration.

**3.15.6** The sequence of elements specified in the Implementation element shall define the sequence of operations of the activity.

**3.15.7** A Parameter child element of the Implementation element shall specify the transmission of a parameter request or indication primitive to an interface provided or required by the component type.

**3.15.8** A Parameter child element of the Implementation element shall carry

– a parameter attribute, identifying the parameter to which the primitive relates;

– an operation attribute identifying whether the primitive is for a get or set operation; and

– a transaction attribute which permits this primitive to be related to the opposing primitive of the request/indication pair.

**3.15.9** The operation attribute of the Parameter element shall specify the operation to be carried out in response to the primitive which is either the value 'get' to acquire the value of a parameter or 'set' to command the value of a parameter.

**3.15.10** The transaction attribute of the Parameter element shall permit the primitive transmission to be matched to the corresponding primitive reception using a string identifier and shall be present or absent depending on the conditions given in table 3-5.

**3.15.11** A Parameter child element of the Implementation element shall include an ArgumentValue element which, in turn, includes 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 3-11.)

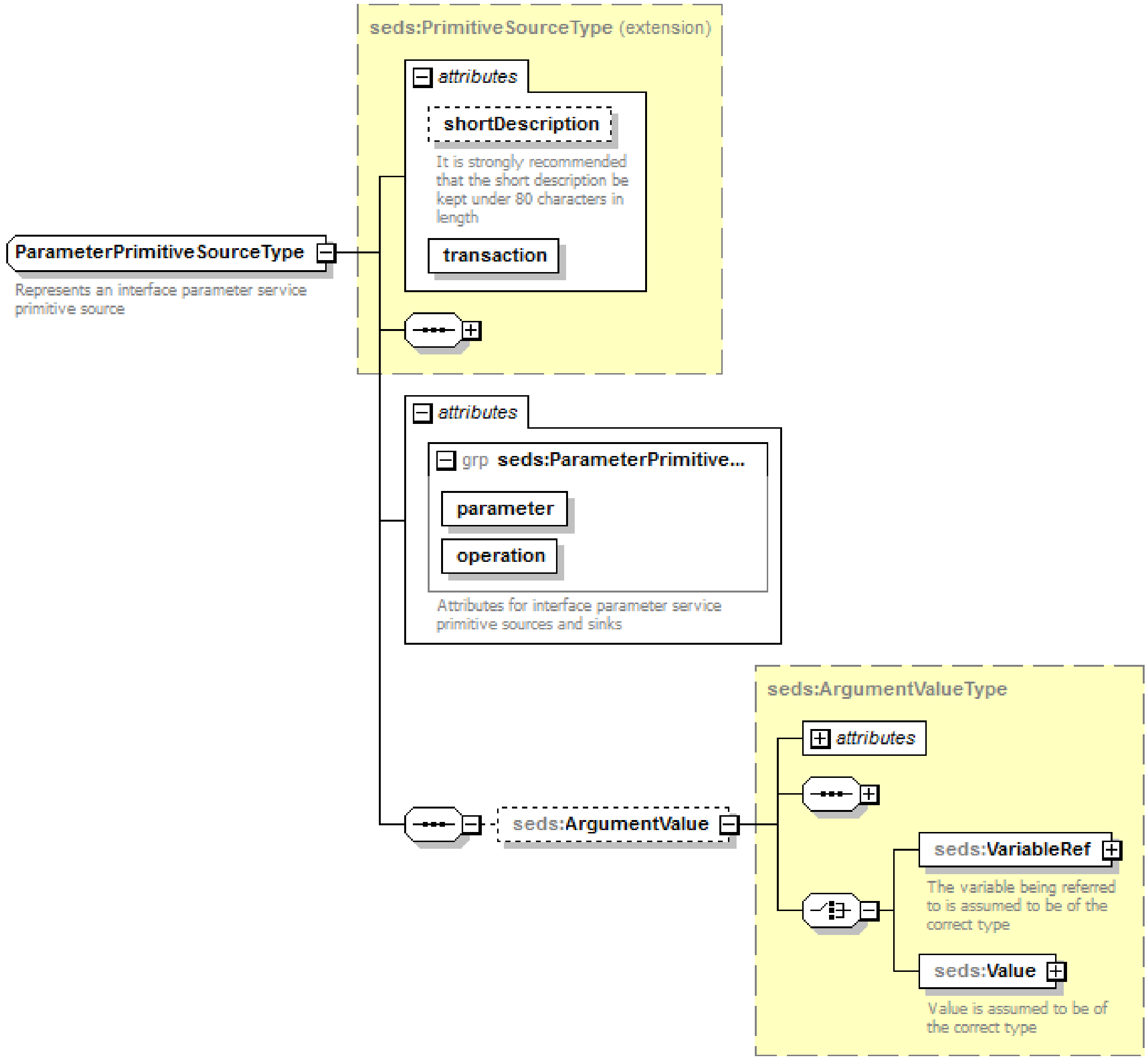



**Figure 3-11: Parameter Primitive Source as Used within an Activity**

**3.15.12** 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 to which the primitive relates.

**3.15.13** A `Command` child element of the `Implementation` element shall specify the transmission of a command request or indication primitive to an interface provided or required by the component type.

**3.15.14** A `Command` child element of the `Implementation` element shall carry a `command` attribute, identifying the command to which the primitive relates and may have a `transaction` attribute which permits this primitive to be related to the opposing primitive of the request/indication pair.

**3.15.15** The `transaction` attribute of the `Command` element shall permit the primitive transmission to be matched to a corresponding primitive reception using a string identifier and shall be present or absent according to the conditions expressed in table 3-5.

**3.15.16** A `Command` child element of the `Implementation` element shall include zero or more `ArgumentValue` elements each of which, in turn, includes either a `Value` element, specifying a literal value to be associated with a command argument to the primitive, or a `VariableRef` element which specifies a component variable to associate with a command argument to the primitive. (See figure 3-12.)

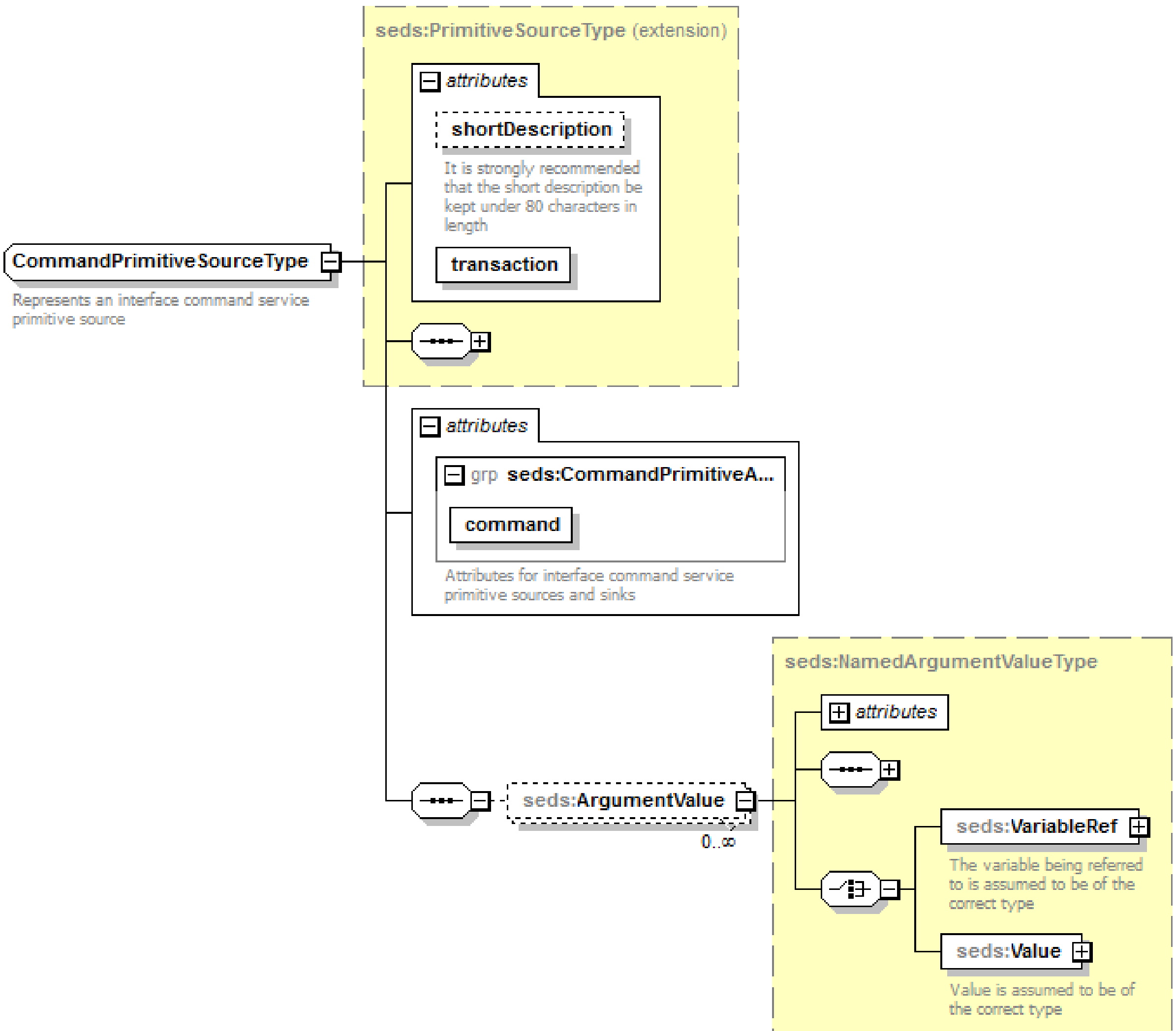



**Figure 3-12: Command Primitive Source as Used within an Activity**

**3.15.17** Each `ArgumentValue` child element of a `Command` element shall carry a `name` attribute identifying the command argument with which this value should be associated.

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

**3.15.19** An `Assignment` child element of the `Implementation` element shall carry an `outputVariableRef` attribute identifying the component variable to which the value should be assigned.

**3.15.20** An `Assignment` child element of the `Implementation` 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. (See figure 3-13.)

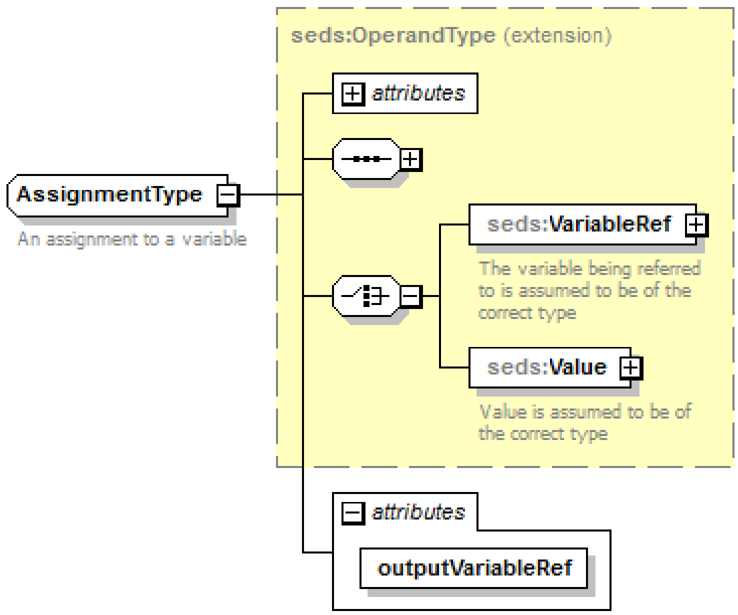



**Figure 3-13: Parameter Assignment as Used within an Activity**

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

**3.15.22** A `Calibration` child element of the `Implementation` element shall carry an `outputVariableRef` attribute identifying the component variable to which the calibrated value should be assigned.

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

**3.15.24** A `Calibration` child element of the `Implementation` element shall include either a `SplineCalibrator` or `PolynomialCalibrator` element.

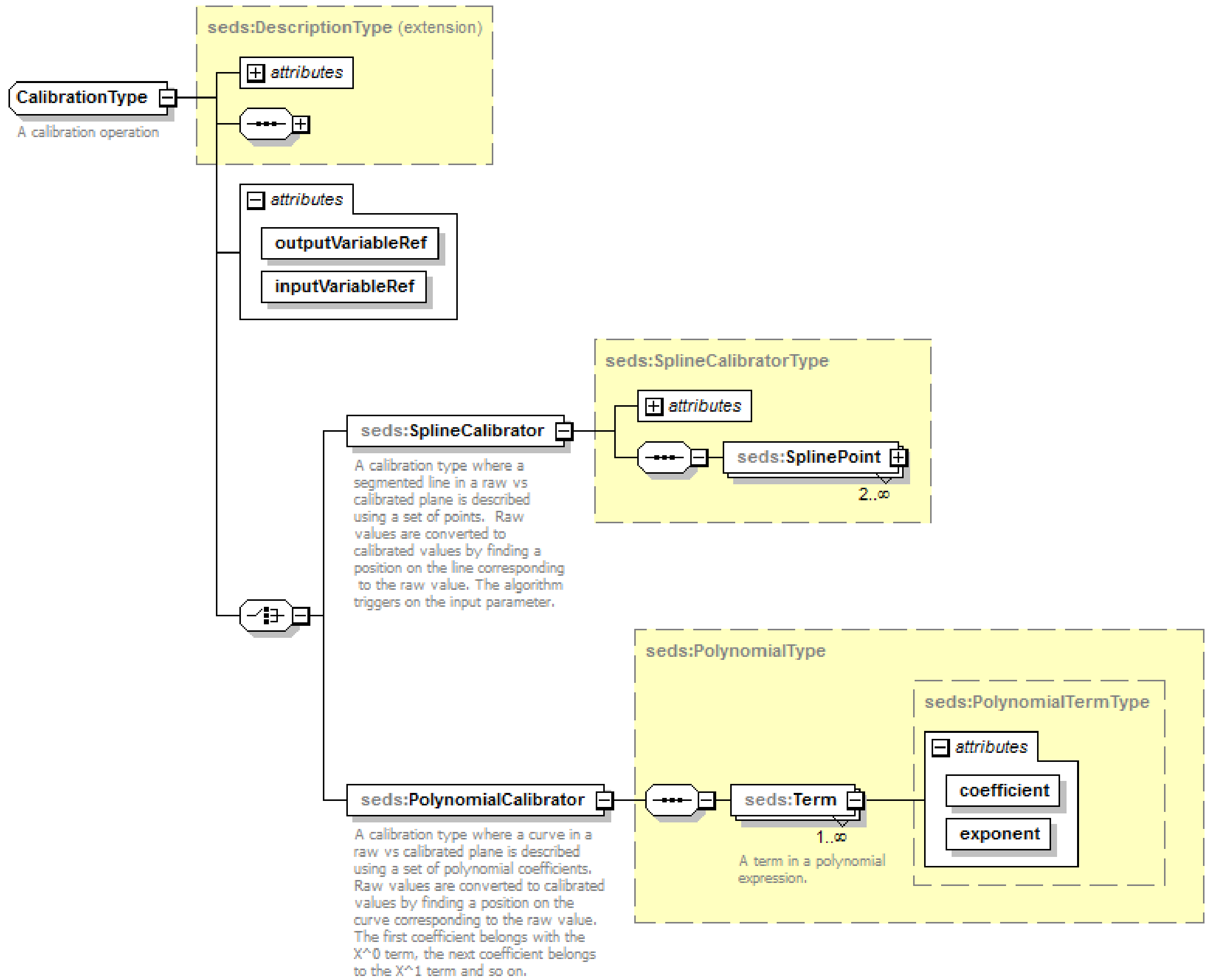



**Figure 3-14: Polynomial and Spline Calibrations**

**3.15.25** The attributes of a `SplineCalibrator` child element of a `Calibration` element shall be `order`, indicating the order in the underlying spline polynomial, and `extrapolate`, indicating whether to extrapolate values outside the range of points.

NOTE – A spline of order 1 is linear, one of order 2 quadratic.

**3.15.26** A `SplineCalibrator` child element of a `Calibration` element shall have 2 or more `SplinePoint` child elements.

**3.15.27** 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.

**3.15.28** A `PolynomialCalibrator` child element of a `Calibration` element shall have one or more `Term` child elements.

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

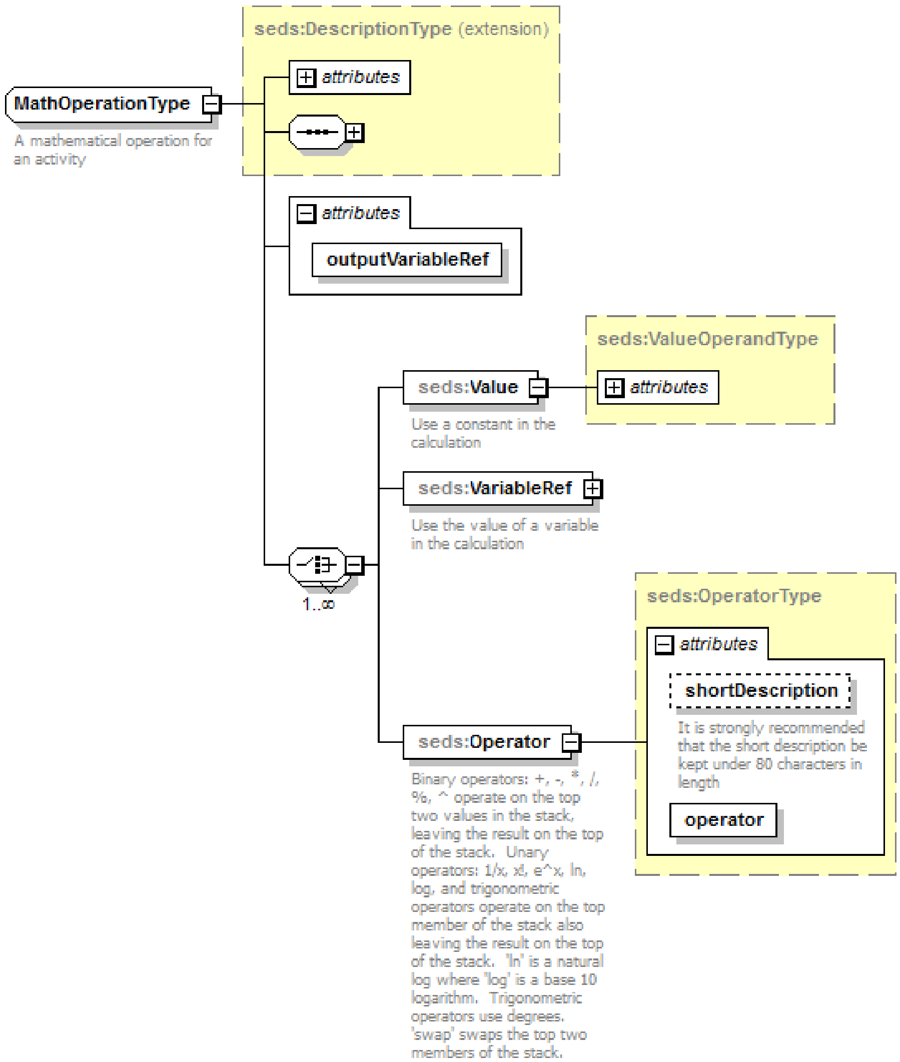



**Figure 3-15: Math Operations**

**3.15.30** A `MathOperation` child element of the `Implementation` element shall specify a mathematical operation in postfix (Reverse Polish) notation.

**3.15.31** A `MathOperation` child element of the `Implementation` element shall carry an `outputVariableRef` attribute identifying the component variable to which the calculated value should be assigned.

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

– `Value`;

– `VariableRef`; and

– `Operator`.

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

**3.15.34** An `Operator` child element of a `MathOperation` element shall have a single attribute `operator`, which shall be one of the values from the table 3-6.

**Table 3-6: 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 |

**3.15.35** A `Conditional` child element of the `Implementation` element shall specify the conditional execution of elements of the activity.

**3.15.36** A `Conditional` child element of the `Implementation` element shall include one `Condition` element, zero or one `OnConditionTrue` element, and zero or one `OnConditionFalse` element.

**3.15.37** A `Condition` child element of a `Conditional` element shall specify a Boolean expression as shown in figure 3-16.

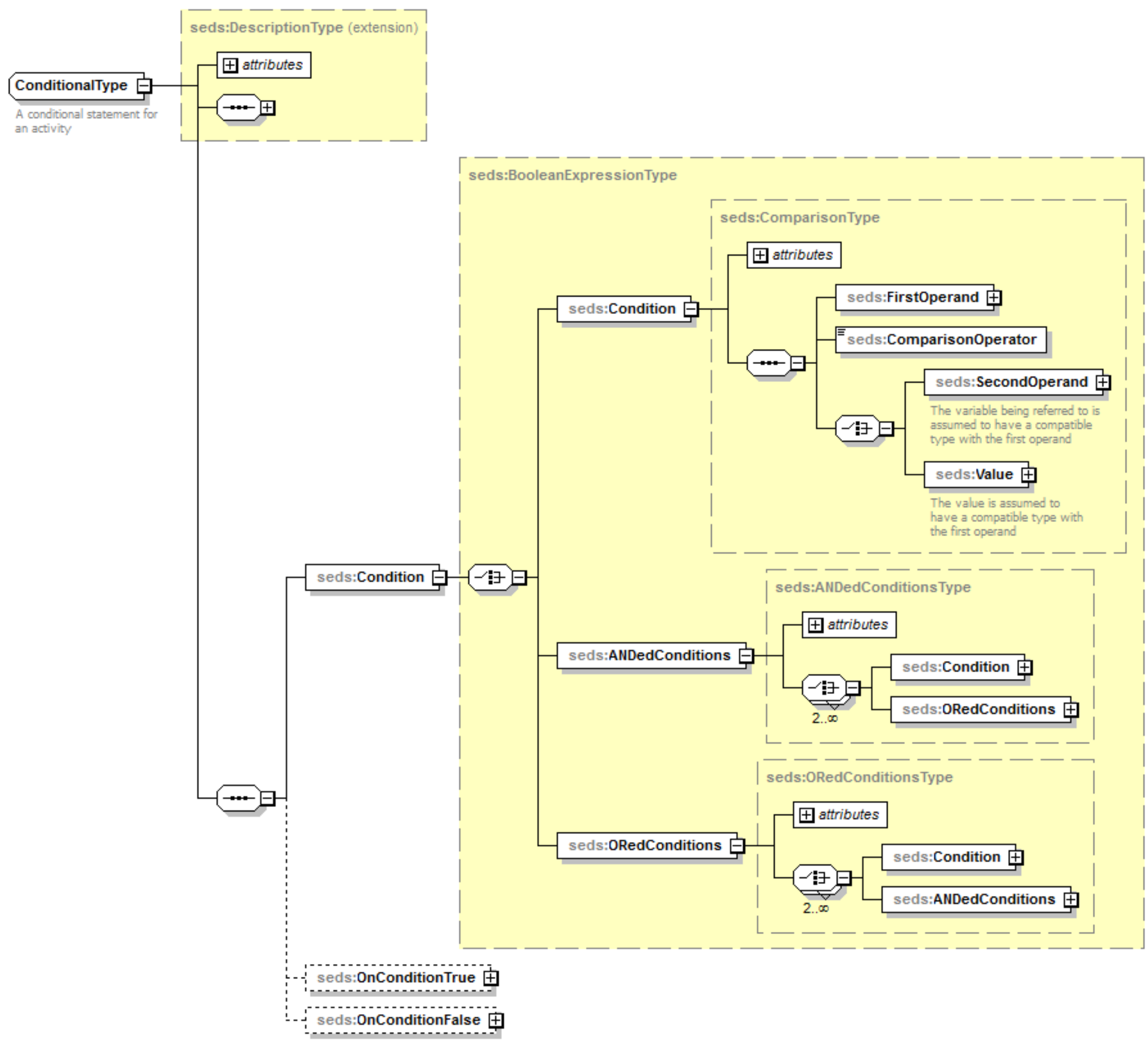



**Figure 3-16: Conditional Execution of an Activity or State Machine Transition Guard**

**3.15.38** The `OnConditionTrue` child element of a `Conditional` element, if present, shall contain one or more of the following elements specifying the operations to perform if the outcome of the condition expression is ‘true’: `Parameter`, `Command`, `Calibration`, `MathOperation`, `Assignment`, `Conditional`, and `Iteration`.

**3.15.39** The `OnConditionFalse` child element of a `Conditional` element, if present, shall contain one or more of the following elements specifying the operations to perform if the outcome of the condition expression is 'false':

- `Parameter`;
- `Command`;
- `Calibration`;
- `MathOperation`;
- `Assignment`;
- `Conditional`; and
- `Iteration`.

**3.15.40** An `Iteration` child element of the `Implementation` element shall specify the repeated execution of elements of the activity.

**3.15.41** An `Iteration` child element of the `Implementation` element shall carry an `iteratorVariableRef` attribute identifying the component variable to use to hold the iteration value.

**3.15.42** An `Iteration` child element of the `Implementation` element shall either contain an `overArray` element or a `StartAt` element, zero or one `Step` element, and an `EndAt` element, in that order.

**3.15.43** An `Iteration` child element of the `Implementation` element shall contain a `Do` element after all other elements.

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

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

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

**3.15.47** The `Step` element of an `Iteration` element, if present, 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


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

use as the source of the value to be used as the difference in value of the iteration parameter between iterations.

**3.15.48** The `Do` child element of an `Iteration` element shall contain one or more of the following elements specifying the operations to perform in each iteration:

– `Parameter`;

– `Command`;

– `Calibration`;

– `MathOperation`;

– `Assignment`;

– `Conditional`; and

– `Iteration`.

## 3.16 STATE MACHINES

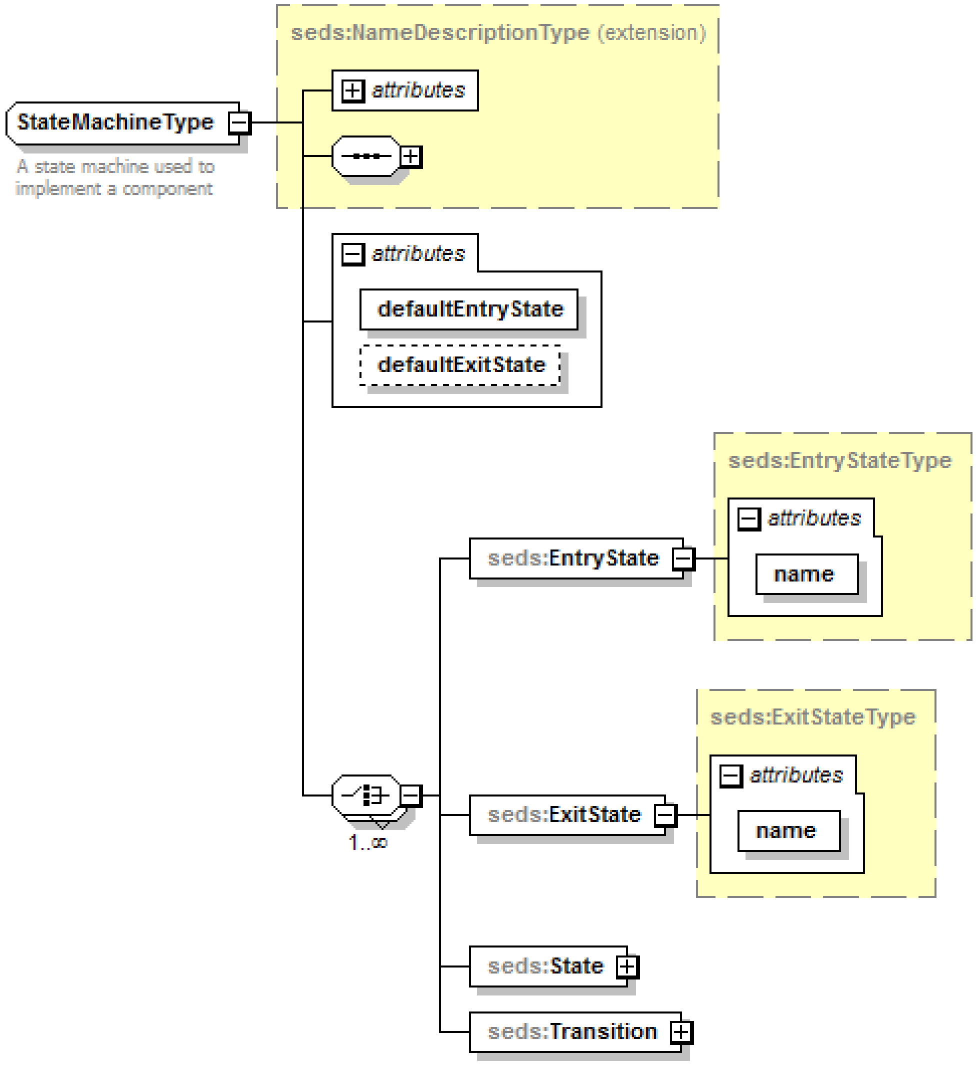



**Figure 3-17: State Machines**

**3.16.1** Each `StateMachine` child element of a `StateMachineSet` element shall carry a `defaultEntryState` attribute identifying the name of the entry state to use as the default entry state and zero or one `defaultExitState` attribute identifying the exit state to use as the default exit state.

NOTE – Exit states are most applicable only to sub-state machines. A top-level state machine that specifies one or more exit states will immediately re-enter at the default entry state. The specification of a default exit state will not be used unless the state machine is a sub-state machine.

**3.16.2** Each `StateMachine` child element of a `StateMachineSet` element shall include one or more of the following elements:

- `EntryState`;
- `ExitState`;
- `State`; and
- `Transition`.

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

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

**3.16.5** Each `State` child element of a `StateMachine` element shall include zero or one of the following elements:

- `OnEntry`;
- `Do`; and
- `OnExit`.

**3.16.6** The `OnEntry`, `Do`, and `OnExit` child elements shall each specify the name of an activity, using the `activity` attribute, to be invoked on entry to the state, immediately after entry to the state, and immediately before exit from the state, respectively.

**3.16.7** The `OnEntry`, `Do`, and `OnExit` child 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.

**3.16.8** Each `Transition` child element of a `StateMachine` element shall carry

- a `fromState` attribute, identifying the name of the state that this transition starts from;
- a `toState` attribute, identifying the name of the state that this transition ends at; and
- zero or one `afterTime` attributes, identifying the delay that should occur between the conditions for the transition being met and the state transition taking place.

**3.16.9** Each `Transition` child element of a `StateMachine` element shall include zero or one of the following elements:

- `OnEvent`;
- `Guard`; and
- `Do`.

**3.16.10** An `OnEvent` child element of a `Transition` element, if present, shall identify the primitive that must be received to trigger the transition, providing that the guard condition is met. If an `OnEvent` element is not present no primitive need be received to trigger the transition.

**3.16.11** An `OnEvent` child element of a `Transition` element, if present, shall carry a `transaction` attribute which permits this primitive to be related to the opposing primitive of the request/indication pair.

**3.16.12** The `transaction` attribute of the `OnEvent` element shall permit the primitive reception to be matched to the corresponding primitive reception using a string identifier.

**3.16.13** An `OnEvent` child element of a `Transition` element, if present, shall utilise a schema instance element `type` attribute (see reference [5]) to identify the element as a `ParameterPrimitiveSinkType` or a `CommandPrimitiveSinkType`.

**3.16.14** If an `OnEvent` element is identified as a `ParameterPrimitiveSinkType`, the element shall carry a `parameter` attribute, identifying the parameter to which the primitive relates, and an `operation` attribute, identifying whether the primitive is for a `get` or `set` operation.

**3.16.15** The `operation` attribute of an `OnEvent` element is identified as a `ParameterPrimitiveSinkType` and shall specify the type of operation request primitive which will trigger the transition with either the value '`get`', for a request to acquire the value of a parameter, or the value '`set`', for a request to command the value of a parameter.

**3.16.16** If an `OnEvent` element is identified as a `ParameterPrimitiveSinkType` and the `name` and `operation` attributes identify the operation as a set primitive from a component type provided interface, the element shall include an `ArgumentValue` element which, in turn, includes a `VariableRef` element specifying a component variable to receive the value associated with the primitive.

**3.16.17** If an `OnEvent` element is identified as a `ParameterPrimitiveSinkType` and the `name` and `operation` attributes identify the operation as a get primitive from a component type required interface, the element shall include an `ArgumentValue` element which, in turn, includes a `VariableRef` element specifying a component variable to receive the value associated with the primitive.

**3.16.18** If an `OnEvent` element is identified as a `CommandPrimitiveSinkType`, the element shall carry a `command` attribute, identifying the command to which the primitive relates.

**3.16.19** If an `OnEvent` element is identified as a `CommandPrimitiveSinkType`, the 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. (See figure 3-18.)

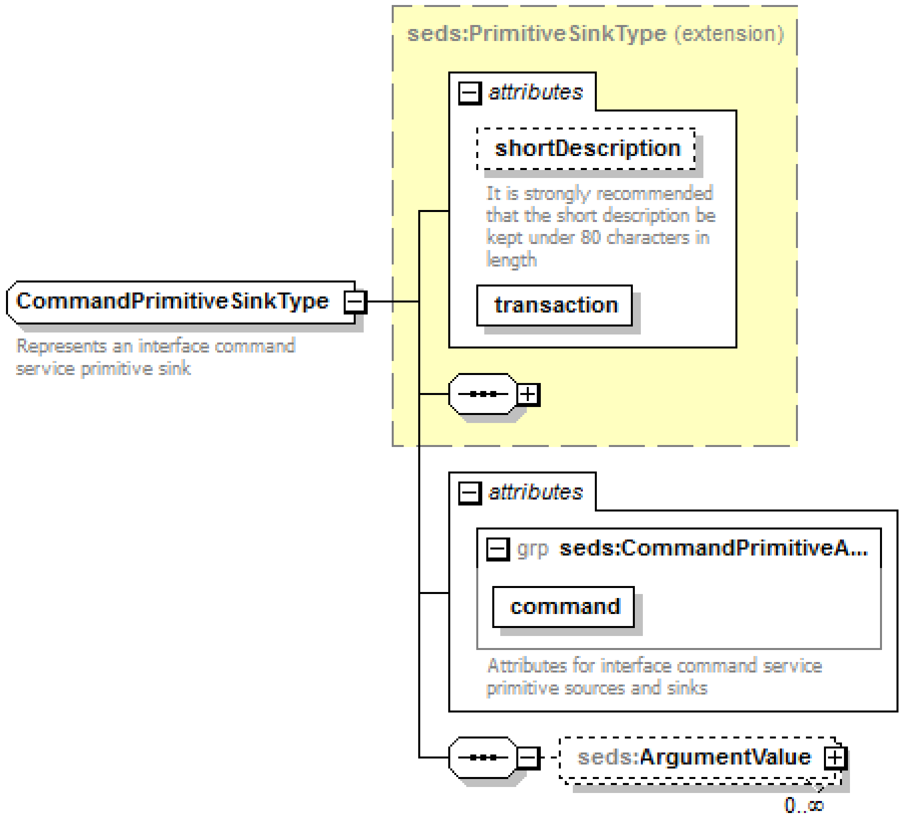



**Figure 3-18: State Machine Transition Events of Type `CommandPrimitiveSinkType`**

**3.16.20** A `Guard` child element of a `Transition` element, if present, shall identify the guard condition that must be met to trigger the transition, providing that the trigger event has been received. If a `Guard` element is not present no condition need be met to trigger the transition.

**3.16.21** A `Guard` child element of a `Transition` element, if present, shall specify a Boolean expression as shown in figure 3-16.

**3.16.22** A `Do` child element of a `Transition` element, if present, shall identify an activity to be performed when the transition is triggered but before the destination state is entered.

**3.16.23** A `Do` child element of a `Transition` element, if present, 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.

# 4 CONSTRUCTING AN SEDS/XML INSTANCE

## 4.1 OVERVIEW

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

## 4.2 XML VERSION

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

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

## 4.3 TYPE REFERENCING AND MATCHING

**4.3.1** Where a data, interface, or component type is referenced locally to a namespace, the referencing name shall match the type name exactly.

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

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

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

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

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

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

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

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

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

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

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

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

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

**4.3.12** Activity or state machine operations which reference a parameter shall reference component variables only, not interface parameters.

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

**4.3.13** Activity or state machine operations which reference an interface parameter or command on an interface provided or required by the component type shall refer to the parameter or command using the following syntax:

{interface name}**/**{parameter or command name}

**4.3.14** Activity or state machine operations which reference a parameter 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}

NOTE – The parameter could be on an interface, in which case this syntax rule would be combined with one of the previous syntax rules.

**4.3.15** Activity or state machine operations which reference a parameter which is an array may select a single element from the array using the following syntax:

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

NOTE – The parameter could be on an interface, in which case this syntax rule would be combined with one of the previous syntax rules.

## 4.4 PRIMITIVE ASSOCIATIONS

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

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

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

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

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

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

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

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

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

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

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

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

**4.4.5** Except in the case of interface parameters marked as asynchronous (having their `mode` attribute set to ‘`async`’) primitives shall

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

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

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

a) transmitted to a component type provided interface;

b) received from a component type required interface.

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

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

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

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

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

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

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

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

**4.4.11** The reception of a command request primitive shall specify the component variable into which the value of all arguments of modes `in` or `inout` can be received.

**4.4.12** The reception of a command indication primitive shall specify the component variable into which the value of all arguments of modes `out` or `inout` can be received.

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

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

## 4.5 STATE MACHINE OPERATION

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

**4.5.2** A state machine transition shall trigger only if the primitive identified by the `onEvent` element is received and the type(s) of the argument(s) associated with the primitive match those of the component variables specified as part of the `onEvent` element.

NOTE – This permits the triggering of state machine transitions to be dependent on the type of data received as part of the primitive. It can be useful for, for example, triggering a state machine transition in response to an incoming packet with a specific value as part of the header.

**4.5.3** If a state machine transition does not specify an `onEvent` element, the reception of a primitive shall be unnecessary to trigger the transition.

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

**4.5.5** The state machine shall exit the state identified as the `fromState` of the transition immediately upon triggering of the transition; exiting shall result in execution of the state `onExit` activity, if such an activity is specified.

**4.5.6** The state machine shall enter the state identified as the `toState` of the transition after the triggering of the transition delayed by the time specified by the `afterTime` attribute. This shall result in execution of the state `onEntry` activity followed by the `Do` activity, if such activities are specified.

**4.5.7** If an `afterTime` attribute is not specified on a state machine transition, the time delay shall be set to zero.

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

NOTE – If a state machine transitions into a state with no specified trigger or guard on a transition from that state, the transition through the state is modelled as instantaneous even if the transition results in one or more activities' being invoked. This has an impact on the ability of incoming primitives to effect transitions.

**4.5.9** If an incoming primitive results in the trigger conditions of multiple transitions to be met and those transitions are on different state machines, then all transitions shall be triggered.

**4.5.10** If an incoming primitive results in the trigger conditions of multiple transitions to be met and those transitions are on the same state machine, then only one transition shall be triggered.

NOTE – Which transition is triggered is undefined and will be implementation-specific. This situation should therefore be avoided when constructing a data sheet.

# ANNEX A

# ELECTRONIC DATA SHEET FOR ONBOARD DEVICES IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

# (NORMATIVE)

## A1 INTRODUCTION

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

## A2 NOTATION

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

<u>Status Symbols</u>

| | |
|---|---|
| M | mandatory |
| O | optional |

Support Column Symbols

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

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

## A3 REFERENCED BASE STANDARDS

The base standards references in the RL are:

– Electronic Data Sheet for Onboard Device – this document.

## A4 GENERATION INFORMATION

### A4.1 IDENTIFICATION OF ICS

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

### A4.2 IDENTIFICATION OF IMPLEMENTATION UNDER TEST (IUT)

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


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

### A4.3 IDENTIFICATION

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

### A4.4 SERVICE SUMMARY

| Ref | Question | Response |
|---|---|---|
| 1 | Service Version | |
| 2 | Addenda implemented | |
| 3 | Amendments implemented | |
| 4 | Have any exceptions been required?<br><br>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 ______ |

### A4.5 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the specification by completing the RL; that is, compliance to all mandatory requirements and the options that are not supported are shown. The resulting completed RL is called a 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.

## A5 GENERAL/MAJOR CAPABILITIES

| Service Feature | Reference | Status | Support |
|---|---|---|---|
| EDS syntax | 3.2.1–3, 3.3.1–3.3.5, 4.2 | M | |
| Subnetwork (at least one of SpaceWire, MilBus, CanBus, or TTE) | 3.4 | M | |
| Namespaces | 3.6, 4.3.1, 4.3.2 | M | |
| Interfaces | 3.12, 4.3.4 | M | |
| Implementations | 3.14, 4.3.5, 4.3.10, 4.3.11, 4.4, 4.5 | O | |

## A6 UNDERLYING LAYERS PROVIDING SERVICES TO IMPLEMENTATION

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

| Service Feature | Reference | Status | Support |
|---|---|---|---|
| XInclude | 3.2.2–3.2.3 | M | |
| Custom Ontology | 3.2.5 | O | |


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

| | | | |
|---|---|---|---|
| Generic Types | 3.12.6–3.12.8, 3.13.9–3.13.13, 4.3.5–4.3.8 | O | |
| SpaceWire | 3.4.2–3.4.11 | O | |
| MilBus | ? | O | |
| CanBus | ? | O | |
| Time-Triggered Ethernet | ? | O | |
| Metadata | 3.5 | M | |
| Data Types | 3.7–3.11, 4.3.3 | M | |
| Interface Types | 3.12, 4.3.4 | M | |
| Component Types | 3.13, 4.3.9 | M | |
| Activities and State Machines | 3.15, 3.16, 4.3.12–4.3.15, 4.4, 4.5 | O | |

# ANNEX B

# SECURITY, SANA, AND PATENT CONSIDERATIONS

# (INFORMATIVE)

## B1 SECURITY CONSIDERATIONS

### B1.1 SECURITY BACKGROUND

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

### B1.2 SECURITY CONCERNS

At the time of writing there are no identified security concerns. If confidentiality of data is required within a spacecraft it is assumed it is applied at the Application Layer. More information regarding the choice of service and where it can be implemented can be found in reference [D10].

### B1.3 POTENTIAL THREATS AND ATTACK SCENARIOS

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

### B1.4 CONSEQUENCES OF NOT APPLYING SECURITY

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


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

### B1.5 RELIABILITY

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

## B2 SANA CONDSIDERATIONS

[To be supplied]

## B3 PATENT CONSIDERATIONS

[To be supplied]

# ANNEX C

# ABBREVIATIONS AND ACRONYMS (INFORMATIVE)

| | |
|---|---|
| CCSDS | Consultative Committee for Space Data Standards |
| CPTP | CCSDS Space Packet Protocol |
| DACP | Device Abstraction Control Procedure |
| DAP | Device-specific Access Protocol |
| DAS | Device Access Service |
| DDPS | Device Data Pooling Service |
| DoT | Dictionary of Terms |
| DVS | Device Virtualisation Service |
| ID | Identifier |
| Mb/s | Mega-bits per second |
| OSI | Open Systems Interconnection |
| OWL | Web Ontology Language |
| RDF | Resource Description Language |
| RMAP | Remote Memory Access Protocol |
| SANA | Space Assigned Numbers Authority |
| SEDS | SOIS Electronic Data Sheet |
| SOIS | Spacecraft Onboard Interface Services |
| XML | Extensible Markup Language |
| URI | Uniform Resource Identifier |


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

# ANNEX D

# INFORMATIVE REFERENCES (INFORMATIVE)

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

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

[D3] *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] *XML Telemetric and Command Exchange (XTCE)*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 660.0-B-1. Washington, D.C.: CCSDS, October 2007.

[D10] *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.


CESG APPROVAL COPY - NOT FOR DISTRIBUTION

# ANNEX E

# EXAMPLE SEDS/XML SCHEMA INSTANTIATIONS

# (INFORMATIVE)

TBD

*This annex will include instructions on where to find the schema set referenced in this standard on the CCSDS Website.*

*Also provided for illustrative purposes will be a number of example instantiations of SEDS.*


CESG APPROVAL COPY - NOT FOR DISTRIBUTION