

Report Concerning Space Data System Standards

|  |
| --- |
| Electronic Data Sheets and Common Dictionary of Terms for Onboard Devices and Components |

Draft Informational Report

CCSDS TBD.0-G-0

Draft Green Book

November 2014

AUTHORITY

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | Issue: | Draft Green Book, Draft A |  |
|  | Date: | November 2014 |  |
|  | Location: | Not Applicable |  |
|  | | | |

**(WHEN THIS INFORMATIONAL REPORT 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 reflects the consensus of technical working group experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in the *Procedures Manual for the Consultative Committee for Space Data Systems*.

This document is published and maintained by:

CCSDS Secretariat

Space Communications and Navigation Office, 7L70

Space Operations Mission Directorate

NASA Headquarters

Washington, DC 20546-0001, USA

FOREWORD

This document is a CCSDS Informational Report to assist readers in understanding the Spacecraft Onboard Interface Services (SOIS) documentation on Electronic Data Sheets (EDS) and the Common Dictionary of Terms (DoT) for Onboard Devices and Components. It has been prepared by the Consultative Committee for Space Data Systems (CCSDS). The concepts described herein are the baseline concepts for the CCSDS standardisation activities in respect of communication services and generic support services to be used in the flight segment of spacecraft systems.

This Report describes the requirements and use cases for electronic data sheets (EDS) providing an interchangeable description of the data interface of devices and software onboard spacecraft, including the provision of semantic information through a Common Dictionary of Terms (DoT). It also provides a worked example of an EDS. It is intended to serve as a reference for both EDS users and EDS tool chain implementers.

Through the process of normal evolution, it is expected that expansion, deletion or modification to this document may occur. This Report is therefore subject to CCSDS document management and change control procedures that are defined in reference [1]. 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 addressed to the CCSDS Secretariat at the 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.
* British National Space Centre (BNSC)/United Kingdom.
* Canadian Space Agency (CSA)/Canada.
* Centre National d’Etudes Spatiales (CNES)/France.
* Deutsches Zentrum für Luft- und Raumfahrt e.V. (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.

Observer Agencies

* Austrian Space Agency (ASA)/Austria.
* Belgian Federal Science Policy Office (BFSPO)/Belgium.
* Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
* Centro Tecnico Aeroespacial (CTA)/Brazil.
* 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.
* European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
* European Telecommunications Satellite Organization (EUTELSAT)/Europe.
* 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.
* MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
* Ministry of Communications (MOC)/Israel.
* National Institute of Information and Communications Technology (NICT)/Japan.
* National Oceanic and Atmospheric Administration (NOAA)/USA.
* National Space Organization (NSPO)/Taiwan.
* Naval Center for Space Technology (NCST)/USA.
* Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
* Swedish Space Corporation (SSC)/Sweden.
* United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

|  |  |  |  |
| --- | --- | --- | --- |
| **Document** | **Title** | **Date** | **Status** |
| CCSDS TBD.0-G-0 | Electronic Data Sheets and Common Dictionary of Terms for Onboard Devices and Components, Draft Informational Report,  Draft A | November 2014 | Current draft |

CONTENTS

Section Page

Executive Summary vii

1 Introduction 1-1

1.1 Purpose 1-1

1.2 Scope 1-1

1.3 Applicability 1-1

1.4 Rationale 1-2

1.5 Approach 1-2

1.6 Terms and Definitions 1-3

1.7 References 1-4

2 OBJECTIVES 2-5

3 EDS and Common Concepts and Architecture 3-6

3.1 Spacecraft Onboard Interface Services 3-6

3.2 Datasheets and Services 3-7

3.3 Device Components and Interfaces 3-10

3.4 Interfaces and Primitives 3-11

3.5 Data Types 3-12

3.6 Generic Types and Interface Usages 3-13

3.7 Component Implementation 3-14

4 Use Cases: End-to-End Mission Walkthroughs 4-15

4.1 Overall Data-Flow/Process 4-15

4.2 Mission Development Phases 4-17

5 Device Data Sheet Example 5-28

5.1 Star Tracker EDS 5-28

5.2 Calibrating Raw Parameters 5-46

5.3 Mapping Native interface to Functional Interface 5-49

5.4 Star Tracker Metadata 5-54

6 Available Reference Material 6-59

7 Security 7-60

ANNEX A ACRONYMS and Abbreviations A-1

CONTENTS (continued)

Figure Page

# Executive Summary

Electronic data sheets (EDSs) facilitate the integration of components to compose a vehicle that can perform a mission. By describing the interfaces of components in algorithmically accessible files, EDSs eliminate the tedium of composition that presently occupies much of the effort of engineers. Instead, that non-recurring engineering effort can be expedited by software tools that function throughout the lifetime of an assembly of components. EDSs introduce an economic ecosystem in which entrepreneurs can offer software tools that exploit the information in electronic data sheets to reduce the cost of design, of physical integration, and of operation.

* This document describes many of the functions that EDSs make possible, simply by describing the interfaces of components in a managed language. Anywhere a person would need to make a connection between components, EDSs can help. During the development of a device, an EDS may be used as a repository for initial requirements, interface control specifications, and validation data. Human-readable documentation can be generated from this information at any time.
* An EDS can describe the design model of a device, so during manufacture of a device it can provide parameters for verification of function of the device. An instance-specific EDS can be constructed by copying the design EDS, adding the serial number to the metadata in the copy, and updating the calibration constants in the copy to agree with measured performance of the instance.
* During design, engineers must select and adapt devices that function together in an assembly to accomplish a mission. EDSs enable construction and search of a database of candidate devices. The collection of EDSs of devices selected for a mission can be transformed to construct a system data base and system data repository. The EDSs of selected devices can be transformed and loaded into standard engineering tools in which engineers add configuration information to produce a system model.
* EDSs provide the device parameters for simulators that can detect some emergent aggregate phenomena before physical integration. Device parameters include nominal operating conditions and performance, which can be checked during simulation.
* During physical integration, engineers validate the design by assembling devices and testing the actual result of composition, which can affect data rates, power levels, thermal conditions, and other conditions that emerge from the aggregate, and are latent in the separate descriptions of devices. EDSs support automation of test procedures by providing nominal bounds for operating conditions and performance.
* EDSs enable algorithmic generation of the protocol stack for data communications between devices. The effect is to generate device drivers for on-board software.
* During operation, it is common in current space flight technology for there to be a virtual channel through telemetry and telecommands between devices of a vehicle and a mission control system. The collection of EDSs for devices selected for a mission can be transformed to construct a mission control system database. EDSs can facilitate automatic generation of graphical user interface layouts to assure the appropriate presentation of data for human operators.
* As space vehicles become more autonomous, the interface descriptions in EDSs can be used by software on board vehicles in detecting and adapting to faults.

The background for the EDS technology described here has developed from two communities of interest. One source was the United States Air Force Research Laboratory, where the focus of the community was to assemble space vehicles rapidly in response to tactically conceived missions. The current source is the Consultative Committee for Space Data Systems Spacecraft Onboard Interface Services working group, which produced this document, and in which the focus of the community is to enable algorithmic generation of the protocol stack for data communications between components. We expect additional contributions and improvements to come in the future from other communities.

# Introduction

## Purpose

The purpose of this document is to describe the concept and supporting rationale for the Electronic Data Sheets (EDS) and Common Dictionary of Terms (DoT) to describe in an interchangeable format the data interfaces of Onboard Devices and Components, complying to the Spacecraft Onboard Interface Services (SOIS), developed by the Consultative Committee for Space Data Systems (CCSDS). This document acts as a handbook for the EDS and Common DoT, providing a rationale, overview and worked examples for them.

This document is a CCSDS Informational Report and is therefore not to be taken as a CCSDS Recommended Standard.

## Scope

This document:

* provides an introduction and overview of the EDS and DoT concepts upon which the detailed CCSDS SOIS recommendations are based;
* provides an overview of the business case for the use of EDS and Common DoT;
* provides an end-to-end mission walkthrough illustrating the expected lifecycle of EDS and where they may be used;
* provides a fully worked example of an EDS;

The basic context of EDS is that, by providing a machine-readable definition of the data interface to a spacecraft device or software component with associated semantic meaning, this can be interchanged between different users of the information held in the EDS without mis-interpretation and allowing for automation of the use of the information, e.g. automatic generation of software device drivers implementing device- or software-specific access protocols.

## Applicability

This document is applicable for any user of SOIS EDS.

## Rationale

The Spacecraft Onboard Interface Services (SOIS) working group is examining the On-board communications architectures to determine how would it be possible to maximize reuse and minimize component integration efforts. Two key parts of this activity are the definition of the Command and Data Acquisition Services (CDAS), which correspond to level 7 in the OSI communication protocol model, and the Message Transfer Service, which occupies a similar position in the protocol stack. As those are above the layers where data can be simply treated as an abstract stream of bytes, implementing those services for a particular device requires knowledge of the layout, interpretation and sequencing of the underlying protocol data units to be transmitted to and from that device.

SOIS Electronic Data Sheets (SEDS) are the proposed standard by which to record such knowledge in a standardized form. It is important to note that the form is standardized here, but not the content of the interfaces. Standardization of interfaces is a separate enterprise that can be facilitated by EDSs, but which is outside the present scope.

## Approach

A SOIS Electronic Data Sheet is a set of related XML files corresponding to the schema defined in references [3] and [4].

Such a datasheet can contain definitions of:

* The **interfaces** that allow two-way data interchange between layers of the OSI protocol stack.
* The **commands** and **parameters** that make up such interfaces.
* The **components** that make up the services that implement a mapping between two sets of interfaces.
* The **state machines**, **variables** and **activities** that make up such components.
* The **types**, **ranges**, **encodings** and **semantic terms** referenced by any of the above.

The intent of the schema is to be usable as an *interchange format* between a variety of tools that need access to the information it contains. Consequently, the design of the schema has been performed according to the following principles:

* Avoid ambiguity or open areas in the specification.
* Limit supported features to those known to be used by one or more real devices.
* Limit the number of different ways it is possible to represent any given device feature.
* Minimise the amount of external validation needed to detect logically inconsistent specifications.

## Terms and Definitions

With respect to service and protocol definition, SOIS, in general, uses terms and definitions defined within the ISO Open Systems Interconnect model defined in reference [2]. The following definitions are provided:

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

**device abstraction control procedure, DACP**: The control procedure that provides the abstraction of a device-specific access protocol to a functional interface. This may involve e.g., the application of calibrations to raw values provided by the device or combination of multiple raw values to determine a derived value in SI units.

**device-specific access protocol, DAP**: The protocol that makes use of a subnetwork service to command and/or acquire data from a device. This is specific to each device as no standardisation of access protocols at the device level exists.

**dictionary of terms, DoT**: Ontology of terms used to describe data in interfaces in Electronic Data Sheets.

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

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

**packet**: Delimited octet-aligned data unit.

**protocol data unit; PDU**: A unit of data specified in a protocol and consisting of protocol control information and possibly user data.

**SOIS Electronic Data Sheet, SEDS**: Electronic description of a device’s metadata, device-specific functional and access interfaces, device-specific access protocol, and, optionally, device abstraction control procedure **Error! Reference source not found.**, compliant with SOIS standards. See EDS.**subnetwork**: An abstraction of a collection of equipment and physical media, such as a local area network or a data bus, which forms and autonomous whole and can be used to interconnect real systems for the purpose of data transfer.

**virtual device**: A virtual version of a single physical device, exposing an idealised interface with a structured syntax and a simplified semantics and thus hiding the operation of the real device.

## References

The following documents are referenced in the text of this Report. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Report are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Recommendations.

[1] *Organization and Processes for the Consultative Committee for Space Data Systems*. CCSDS A02.1-Y-3. Yellow Book. Issue 3. Washington, D.C.: CCSDS, July 2011.

[2] *Spacecraft Onboard Interface Services*. Informational Report, CCSDS 850.0-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, December 2013.

[3] *XML Specification for Electronic Data Sheets for Onboard Devices and Software Components*. Draft Recommendation for Space Data Standards (Blue Book), CCSDS 876.0-R-1. Red Book. Issue 1. Washington, D.C.: CCSDS, November 2014.

[4] *Common Dictionary of Terms for Onboard Devices and Software Components*. Draft Recommendation for Space Data Standards (Blue Book), CCSDS 876.1-R-1. Red Book. Issue 1. Washington, D.C.: CCSDS, November 2014.

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

# OBJECTIVES

Electronic Data Sheets (EDS) is a concept that has been proposed to allow the capture of the relevant information about a piece of equipment. This should capture the relevant aspects not just to enable an efficient exchange of information (easing its maintainability, enforcing consistency, etc.) but should also enable the development process to be partially automated.

Such automation can be expected to reduce not only development costs, but also revalidation costs associated with manual coding errors. Note that it is not expected or required to *eliminate* validation costs; the goal is a test campaign that passes on the first attempt, not a level of confidence that would justify skipping testing completely. In addition to direct cost savings, this approach should also naturally lead to reduced risk of schedule slips.

These savings and efficiencies can only be realised by the development of tooling that works across a range of devices; otherwise you have merely replaced the need to write code to support a specific device with a need to write a tool to support a specific device. Such device-independent tools have to be based on a standard for the representation of the information about the device that forms the input to the tool.

The SOIS EDS is such a *standardised interchange format*, developed for the specific needs of the space industry.

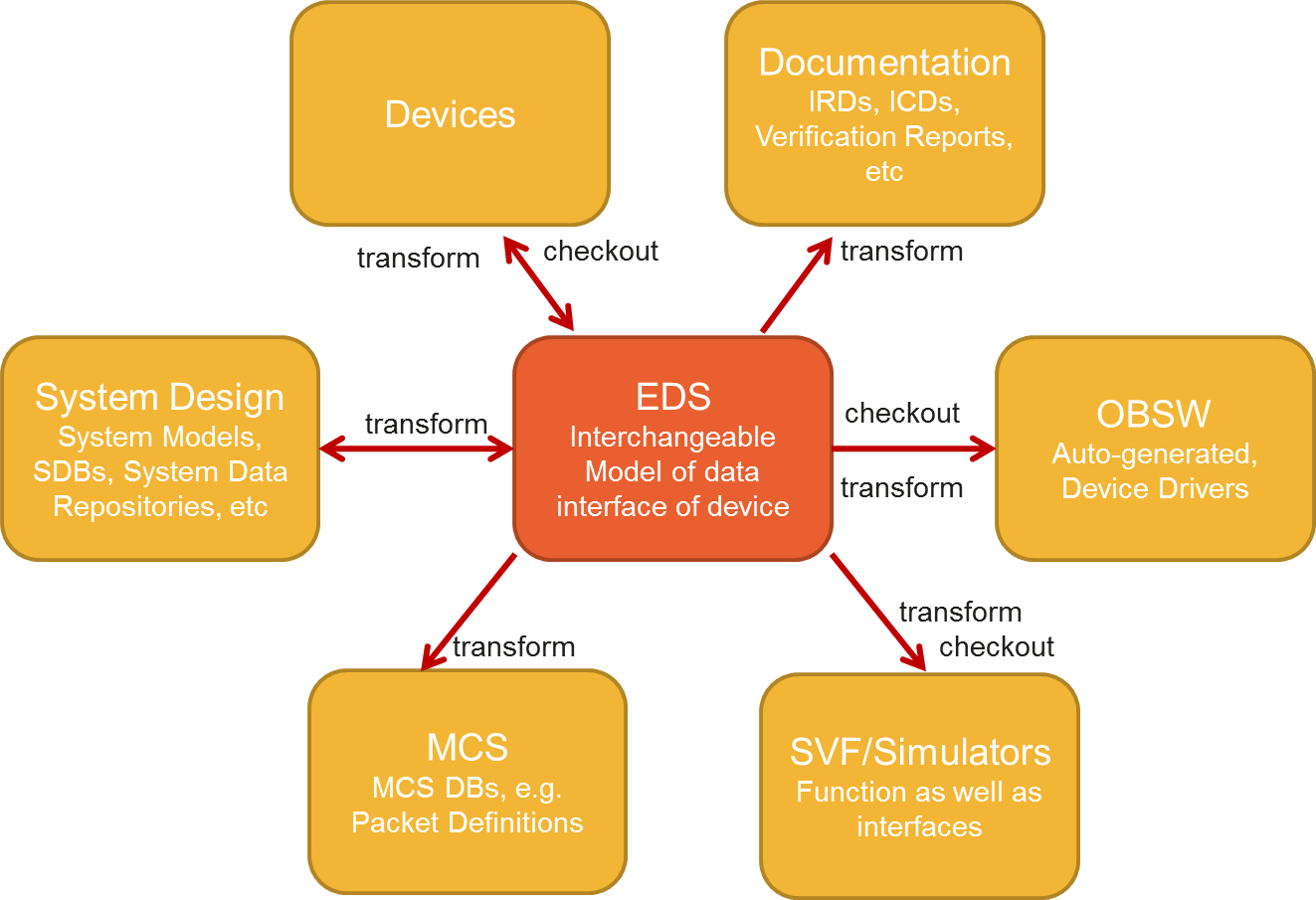


Figure 2‑1 SOIS EDS Scope

Due to the ubiquity of device data across the various activities of the mission lifecycle, the set of such possible tools is wide, as indicated in the diagram above. Consequently, if successful and widely adopted, the benefits will be seen across the mission lifecycle.

# EDS and Common Concepts and Architecture

## Spacecraft Onboard Interface Services



Figure 3‑1: SOIS Reference Communications Architecture

Figure 3‑1 illustrates the SOIS reference communication architecture, including the application support and subnetwork layer services. This is an architectural representation that must be mapped to physical communications. In essence, SOIS attempts to decompose the communications infrastructure inherent in all spacecraft into a defined set of services typically required by all missions. In general, SOIS does not define protocols; rather these are developed or adopted from commercial, CCSDS or ECSS developments.

This architecture conforms to the OSI protocol model, in that each service is at a specific layer and communicates only with those immediately above or below it.

Due to the variability of devices, while the services can be standardised at an abstract level, the implementation of, and communication patterns between, those services cannot be, in general.

The role of the datasheet within SOIS is to provide those device-specific details, allowing a complete and machine-readable specification of a service.

## Datasheets and Services

This section describes the role of electronic data sheets in describing a network (Figure 3‑2), as well as their function in a SOIS software architecture (Figure 3‑3).



Figure 3‑2 Electronic Data Sheets in a Network

The left side of Figure 3‑2 shows devices and the electronic data sheets that describe them.

A remote interface unit (RIU) also appears near the left side of the diagram, where it acts as a concentrator for the messages of some devices. The EDS for the RIU is a composite of the EDS’s for the devices whose messages it concentrates. The EDS for the RIU is more than the composite, because it also describes how the RIU communicates on the bus (1553 in the figure) to the onboard computer, and how the RIU transforms or passes the messages of devices.

To the right of the device layer is a layer for the SOIS protocol stack. It consists of subnet, DAS and DVS. The subnet maps to ISO layers 1 (physical) through 4 (transport). DAS and DVS correspond to layers 6 (presentation) and 7 (application). There is a set of EDS’s that describe the protocol stack, one for each device on the left side of the diagram, including one for each RIU. These EDS’s are the same as the device and RIU EDS’s in the leftmost layer.

Another feature in **Error! Reference source not found.** is an optional software bus, near the center of the diagram. The EDS’s labelled “SOIS Access” describe the adaptation of a device to communicate on the software bus. There is an adaptation EDS for each device that communicates on the software bus. One side of an adaptation EDS is the same as the device EDS; the other side describes a protocol stack with MTS at its top and the software bus at its bottom. There may also be an EDS for each software application that communicates on the software bus. The EDS’s for software applications communicating on the software bus are necessary. The adaptation EDS for devices may be implicit when the following conditions are true; otherwise, the adaptation EDS’s are necessary for portability.

* Tool chain algorithms generate the adaptation software in a way that exposes a specific subset of the interfaces defined in the device EDS’s. For example, all DVS interfaces may be omitted in a spacecraft that uses raw data in flight, rather than engineering units.
* The specific subset of interfaces exposed on the software bus is defined for device participants on the software bus within a single agency or within a project. Such definitions are not portable outside the agency or project, unless the subset consists of all interfaces.
* Only the protocol defined by SOIS MTS is needed to communicate with the device through the software bus.

A separate EDS for the software bus consolidates the EDS’s of the devices and applications on the software bus. The consolidated EDS acts as a catalog of data and services published on the software bus, which are accessible to applications through the MTS function of SOIS.

On the right of the software bus is a command and data handling function for the spacecraft. This function presents the set of EDS’s of the devices in the network, including any applications that communicate on the optional software bus. A mission control center can use that set of EDS’s to monitor and to control a vehicle. That set of EDS’s will be transformed by the tool chain into a language already standardized for description of communications between spacecraft and mission control centers, such as XTCE.



Figure 3‑3 SOIS EDS context

The above diagram shows the context of a SOIS EDS datasheet within a system following the SOIS reference architecture. The service interfaces of the layers above and below of relevance to datasheets are shown; the others are not device-specific, and so do not need datasheet support. In addition to the standardised SOIS services, place-holders are shown for mission or agency-specific services.

Datasheets are shown as UML packages; they are simply containers for the set of interface and component specifications. In this generic diagram, no connections are drawn; which connections actually exist will depend on the specifics of the mission and devices.

Services such as the Subnetwork Packet Service are implicitly global, in that there is a single interface definition which is used for all devices, with the device to use being indicated by an argument to each service request. Whereas a given datasheet is specific to a particular device type, of which there will be many in an overall system.

## Device Components and Interfaces

A SOIS EDS Datasheet is a package that can contain definitions of:

* **Interfaces** that allow two-way data interchange within the scope of a single device.
* **Components** that map a set of *provided* interfaces to a set of *required* interfaces.



Figure 3‑4 SOIS EDS Device Datasheet Contents

A complete datasheet for a device should contain at least two interfaces:

* The **Device-Specific Access Interface**; the lowest-level access to all raw decoded data transmitted to and from the device.
* The **Device-Specific Functional Interface**; higher-level access to calibrated or derived data.

Both of these interfaces are device-specific because different devices support different sets of data. These are split to allow missions the option of supporting only one or the other.

In the typical case, there will be a single component providing each interface, and the component implementing the higher-level interface will be defined in terms of the lower-level one. The lowest-level component will require one or more subnetwork-level interfaces.

## Interfaces and Primitives

Rather than supporting a fixed set of known interfaces, the SOIS EDS schema has an explicit construct for specifying arbitrary interfaces which can be used for both standard and device-specific interfaces.

Within a SOIS EDS datasheet, an **interface** is defined in terms of the **commands** and **parameters** that make it up. Commands have **arguments**, which can be either input, output or both. A read-only parameter can be considered to be a shorthand for a command with a single output argument, avoiding the need to provide two names (e.g. Value and getValue). A non-read-only parameter is equivalent to a pair of single-argument get and set commands.

The data transfer associated with an interface parameter, or the invocation of a command, is modeled using the basic OSI service **primitives** which are used throughout the SOIS standards. For example, the acquisition (reading) of the value of a parameter on an interface provided by a component is modeled 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, one received, forms a **transaction**.

Table 1 Mapping between parameters, commands and primitives

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

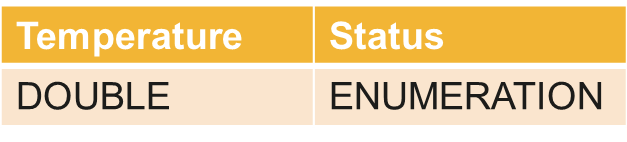
The above table shows the mapping between the interface definition and the request and indication primitives that pass across the interface. Where both a request and indication exist, that parameter or command is counted as transactional, in that the underlying service layers are responsible for associating the two together

Note that there are not currently any explicit constructs for describing more complicated patterns of interaction across an interface, such as multi-stage progress reports.

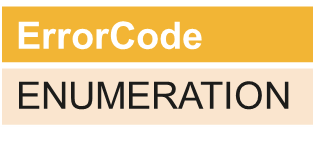
## Data Types

The parameters and command arguments of an interface are defined in terms of named **types**, which can have a **range**, **encoding** and **semantics** (see XREF).

These types can be simple (Boolean, integer, float, enumerated, etc.) or aggregate (fixed and variable-length arrays, containers). Containers can be **abstract**, in which case they are placeholders for any of a known set of concrete types that share a common header that can be used to classify them.



Packet ID = 3



Packet ID = 4

Figure 3‑5 Using values specified by an abstract container to determine the concrete type

Numeric types can have numeric ranges; enumeration types can have ranges that are arbitrary subsets of the full set of values.

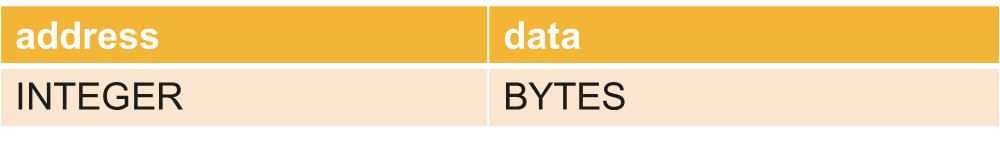
Specification of encoding is only necessary for primitives that will be converted to or from byte arrays.

Data types are also used to define variables within a container and arguments to activities; see later sections.

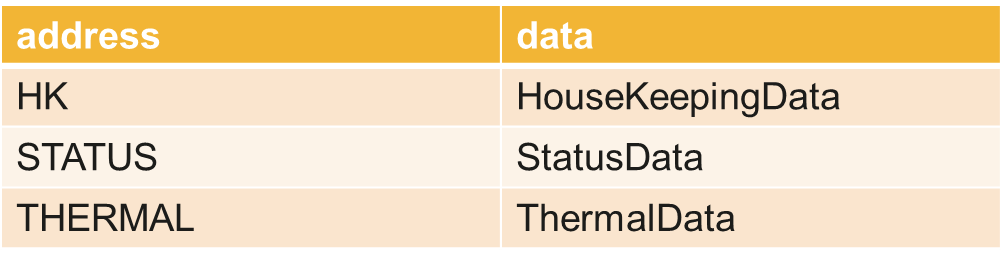
## Generic Types and Interface Usages

When a datasheet references a standardised or otherwise externally-specified interface definition, it is common for the types used to *define* that interface to not be the most appropriate ones to *describe its usage*. For example, a generic interface may allow arbitrary integers, whereas within the context of a specific device, only a fixed set of enumeration values are used. Similarly, a generic interface such as the Subnetwork Packet Service may be defined in terms of arbitrary byte arrays, but a particular device will only support a fixed set of packets, i.e. an abstract container as above.

In such cases, the **usage** of the interface specification by the datasheet component can be accompanied by a **generic type mapping**.



Example command on generic interface



Same command on interface usage within datasheet

Figure 3‑6 Using a type mapping to capture the relation between command arguments

In the above diagram, the first command expresses the idea that ‘if you read from an address, you will get some data back’. The second specifies the same abstract interface in terms of a its support by a specific device, i.e. ‘if you read from one of these 3 supported addresses, you will get the corresponding known set of data back’.

This is possible using generic type mappings to replace the arbitrary address with a specific enumeration type, and then setting up a set of **alternate mappings** for the data type corresponding to each legal value for the address.

Note that the data type mapped to can itself be abstract, representing variably-structured data that will still need to be classified in order to be fully processed.

## Component Implementation

The implementation of a component can be specified within the datasheet in terms of:

* A set of **state machines** where the triggering events are incoming “sink” primitives on the component’s provided and required interfaces.
* A set of **variables** that can hold data values over time.
* A set of **activities** that can be referenced by state machine transitions, perform basic calculations (including **calibration**) and initiate outgoing “source” primitives on the component’s provided and required interfaces



Figure 3‑7 Sample component implemented by state machines

The state machine concept used is taken directly from UML; a [directed](http://en.wikipedia.org/wiki/State_diagram" \l "Directed_graph" \o "State diagram) graph in which nodes denote **states** and connectors denote state **transitions** protected by **guards**.

# Use Cases: End-to-End Mission Walkthroughs

This section provides an end-to-end walkthrough of the development of a Mission illustrating a variety of use cases where EDS and Common DoT may be used.

## Overall Data-Flow/Process

A summary of some representative use cases for electronic data sheets appears in Figure 4‑1. More use cases could be added, but this set is representative of the diverse opportunities to reduce costs when transferring descriptions of components through a mission development process.

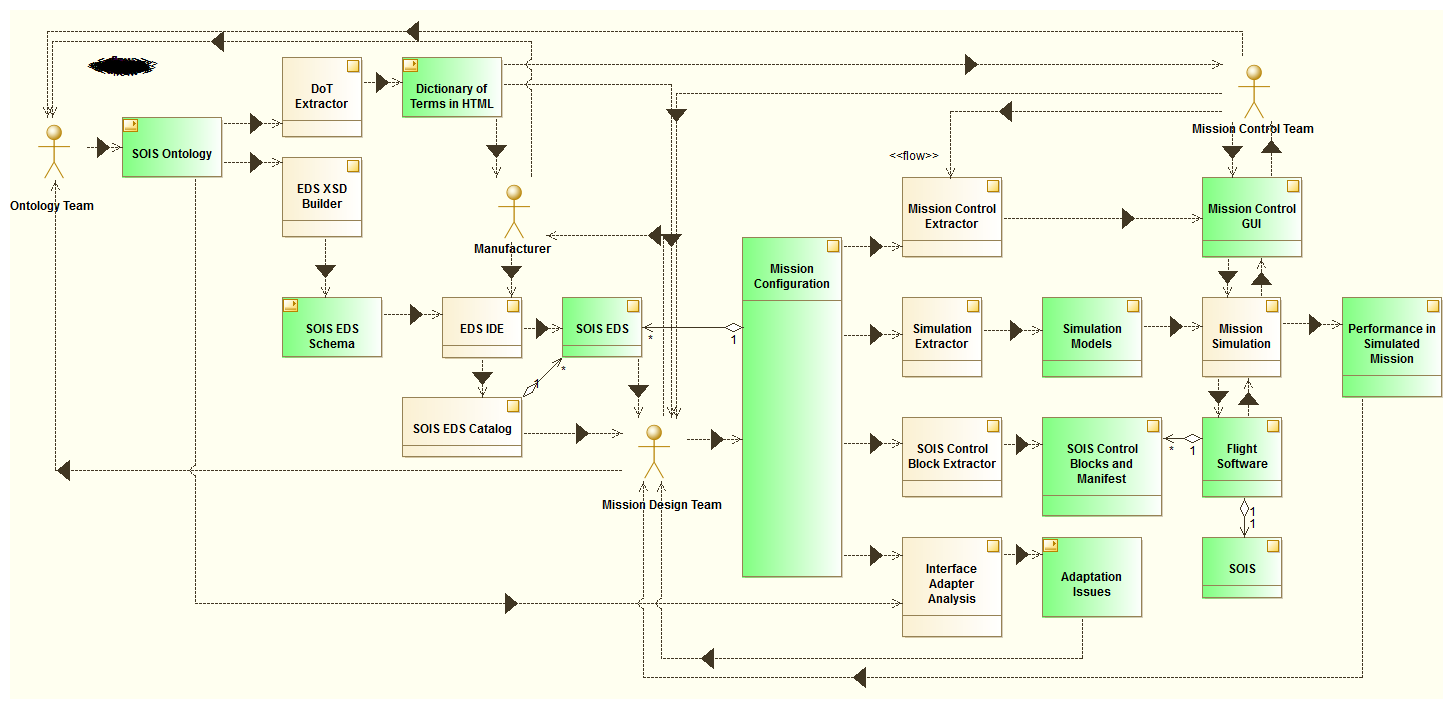


Figure 4‑1 Some Use Cases for Electronic Data Sheets. The processes in Figure 4‑1 are rendered as beige blocks, while the data artifacts are green blocks.

Four teams of people appear in Figure 4‑1; three of those teams would appear in a traditional mission development process.

* Manufacturer: This team represents the people who provide parts for the mission vehicles. This may include both hardware and software manufacturers. The concept can include in-house instrument builders and software engineers.
* Mission Designers: This team consists of people who decide how to compose parts into the assemblies needed for a mission. Starting with mission objectives, the designers may invent one or more vehicles or networks to accomplish the mission.
* Mission Control: This team includes the people who operate the mission vehicles. These people must track and understand the status of parts of the vehicles, and they must command or guide the vehicles to conduct the mission.
* Ontology Team: This team stays in the background, coming out when necessary to facilitate the use of terms in electronic data sheets. The team can serve the needs of multiple projects by publishing terms in a global location that enables a part to be used in any project, with the same interpretation for its electronic data sheet in each project. Some large projects, or a project with a short time line, may have a local ontologist in order to provide locally-defined terms that drive the project tool chain for parts that do not need to be portable across projects.

The diagram in Figure 4‑1 will be treated piecemeal in section 4.2. The software tools described in that section are expected to develop in response to the availability of consistent descriptions of parts in electronic data sheets. The tools are described conceptually, and it is left to the ingenuity and economics of the industry to implement them.

## Mission Development Phases

This section breaks a mission down into development phases and discusses topics on the use of EDS and Common DoT, including tooling support.

### Selecting Parts

The first use case appears where a group of engineers are figuring how to implement a mission. The engineers have gotten to the point of selecting the parts to build a mission vehicle, given some practical constraints that they have derived from mission requirements. In a traditional process, there would be a search of possible suppliers, using various media, including telephones, internet, advertisements received in mail, and memory of prior projects. The sense that no stone has been left unturned is seldom achieved, because this search occupies valuable time and talent and so must often be curtailed.

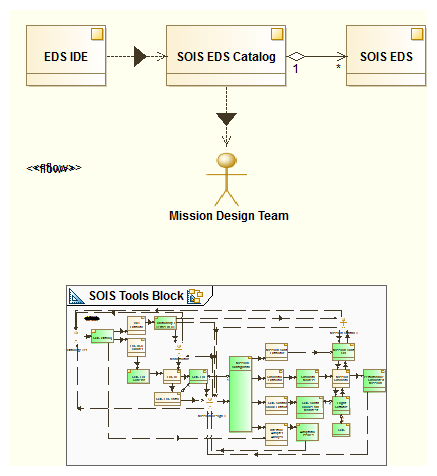


Figure 4‑2. Rapid Search for Candidate Parts

Electronic data sheets can facilitate this search by making each manufacturer’s wares accessible to algorithmic search engines. The manufacturers have the privilege of keeping their EDS’s confidential while presenting their EDS’s to the search engines for indexing. The mission designers can rapidly obtain a list of candidate parts by using the practical constraints that they derived from mission requirements as search arguments.

The kind of search engine that uses constraints for search arguments can provide more precise results than the generic search engines that operate on unstructured text arguments to find web pages. The precision is possible due to the fact that electronic data sheets use simple syntax and terms defined in the global ontology.

Some working prototypes of this kind of search engine have been developed, using various techniques to optimize an engineer’s time. Standard constraint solver technology can whittle away at physical resource allocations and reduce other features of the design space as engineers make choices of parts. Randomized search can explore a design space with various combinations of parts until a viable combination is found.

At the time of this writing, the development of electronic data sheets has concentrated on the description of data interfaces. However, prototypical research has been conducted on the description of physical interfaces, including mounting planes, mass properties, power utilization, thermal properties, and radiation requirements. The electronic data sheet syntax allows for a “Metadata” element that can be used by defining terms in the ontology to describe physical interfaces of devices. The physical interface description is important to the process of selecting components, and later in flight simulation.

### Validating a Mission Configuration

When the mission designers have a short list of candidate parts, they can work out the connections and relations among the parts, building a mission configuration file. Standard modelling tools, such as MOF, UML, and SysML, can be used to describe the mission configuration in an algorithmically accessible manner. This brings up another activity that occupies the time of engineers on the design team. They must tediously check the mission configuration to be sure that every part has the resources that it needs.

There are at least two classes of questions about resources. Here are representatives of those classes. Is there a publisher of attitude data on board that the attitude control application can use? Can the solar panels keep the battery charged, given the orbit and the mission operations? The first class of questions can be answered in a straight-forward manner as described in Figure 4‑3. The second class of questions requires a system-wide view of the mission, which will be explained in section 4.2.5.

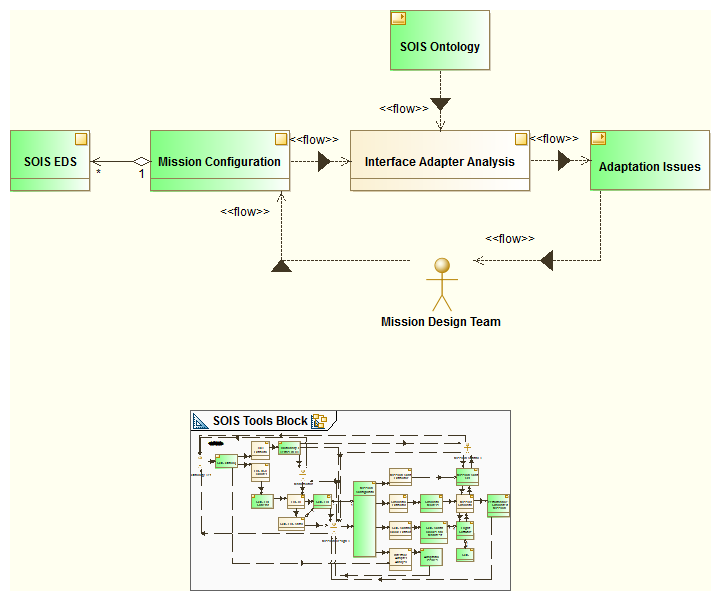


Figure 4‑3 First-Order Validation of Design

The interface adapter analysis tool aids mission designers in validating a design, using the set of electronic data sheets referenced by the mission configuration and their relationships in the configuration. In many cases, the validation of resources for a part is simply done by checking that the terms describing required interfaces in the part’s EDS match the terms describing provided interfaces in another part’s EDS.

Sometimes, validation is complicated by partial or cryptic agreement between interfaces. The tool uses the ontology to make the more complicated comparisons. Here is an example. A designer may have chosen an inertial rate unit that presents its data as raw counts from an analog to digital converter, while the attitude control software application requires an interface that provides rates in radians per second. The quantity kind (rotation rate) of the data match, but the units of measure (counts versus radians per second) do not. The electronic data sheet for the inertial rate unit describes a calibrator that transforms the counts to engineering units in the device abstraction control procedure. In case the calibrator produces degrees per second, the ontology describes conversions between engineering units, so the validation algorithm can recognize that a conversion between radians and degrees is needed to make an exact match. The importance of the calibrator is to provide a function that converts raw counts to some kind of engineering units. The calibrator could be adjusted when generating the driver for the inertial rate unit in section 4.2.3, but for the purpose of validating the mission configuration, it is only necessary to indicate the nature of mismatch.

The interface adapter analysis produces a list of interfaces that are required but not provided in the mission configuration, as well as a list of adaptations that are needed to satisfy weakly matching interfaces. The mission design team treats the report of the interface adapter analysis as a list of unfinished work.

### Generating Flight Software

An electronic data sheet for a device describes how to transform between messages in sub-network layers of a protocol stack and data presented at a functional interface for use by applications. The description is not strongly procedural, so it accommodates a variety of implementations. See section 3.7.

It is important to state that the description of protocol layers in an EDS is not intended to be a description of a single implementation. Rather, the purpose of the seemingly procedural description is to provide a concise package that contains the constraints on using a device. Here are some of those constraints, and a suggestion of how loosely they are intended to be interpreted.

* A calibrator describes a conversion between raw counts and engineering units. The fact that a calibrator is present in an EDS does not mean that the calibrator must be on board a vehicle along with the device described by the EDS. A mission designer could choose to build a control system that operates on the raw counts, never converting to engineering units on board the vehicle. Given such a design decision, the calibrator in the EDS could provide the device-dependent constants needed by ground software, so the mission control team can display telemetry in engineering units. The EDS would participate in both the development of flight software and in the configuration of ground software.
* A state machine describes the sequence of messages necessary to operate a pyro device. The machine cannot reach the “fire” state without passing through the “arm” state. This constraint describes the flow of messages without limiting the software paradigm used to implement that flow. A software designer could use this description as if it were a specification in UML.
* See section 5.1.9 for another example, dealing with features that might or might not be implemented in the subnetwork layer.

Figure 4‑4 shows a software tool that extracts SOIS control blocks from the mission configuration and the EDSs that it references.

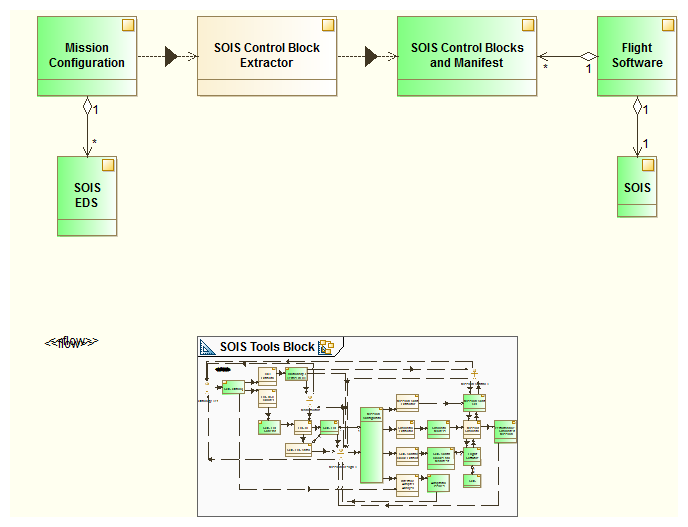


Figure 4‑4 Extracting Flight Software Tables from Mission Configuration

Prototypes for generating device drivers from EDSs have been built for multiple platforms, including a SOIS reference implementation and a flight executive organized around a software bus.

* + Device Driver development – this can be filled in from ESA’s “Adoption of EDS and Device Virtualisation” project outputs
  + Software component development – this can be filled in by NASA/GSFC from their adoption of MTS EDS for cFE

### Mission Control Software

Electronic data sheets provide an efficient source for loading mission control databases and for configuring displays. Using the electronic data sheets referenced by a mission configuration, a software tool builds a model of the data that will be visible to mission controller personnel in a set of screens. The mission control team guides the tool in selecting the data to be displayed in each screen. The software tool can produce graphical user interface layout files, or equivalent tables. It also produces the database behind the display, which constitutes the definition of the mission control view for operating the mission. See Figure 4‑5

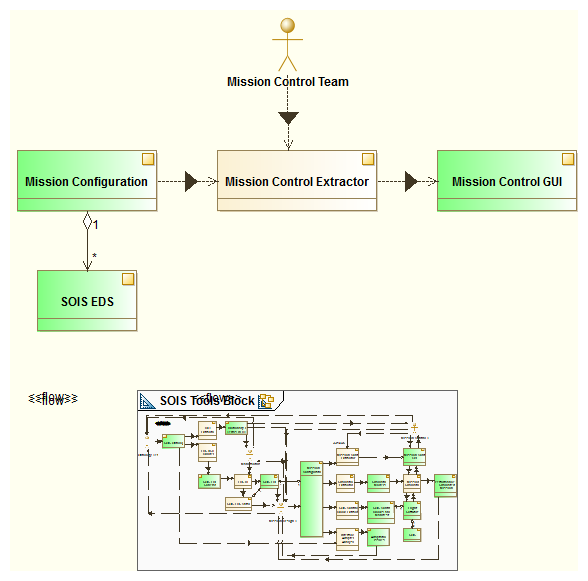


Figure 4‑5 Extracting a Mission Control Database

### Simulation

When the design for a mission vehicle has gathered enough detail to be testable, flight simulation can suggest whether the design can be viable in operation. Figure 4‑6 shows a software tool that extracts a simulation model from a mission configuration, including the EDSs referenced by the design.

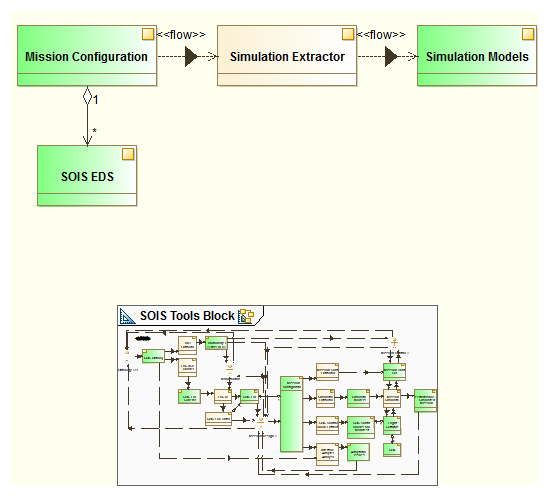


Figure 4‑6 Extracting a Simulation Model from a Mission Configuration

There are at least two levels of simulation models that can exploit EDSs to simulate a day in the life of a mission vehicle. The mission configuration provides the highest level model of a vehicle; it can be converted into a simulation model that represents the parts of a vehicle and the relations among the parts. The EDSs referenced by the mission configuration provide parameters for simulation models of parts of the vehicle, which constitute a more detailed level of simulation.

Prototypical simulators have been constructed that support the two-level vehicle model described in the previous paragraph.

In addition to the day-in-the-life simulation described above, it is also possible to use EDS’s for scenarios that only affect a part of the mission configuration. Partial simulations can be achieved by the use of EDSs for the incremental validation of the FSW by simulating and stubbing the FSW and the simulated equipment counterparts at the level of the interfaces captured by the EDS.

### Reality In the Loop

Given the consistent description of mission vehicle parts that EDSs provide, it is easy to simulate the mission with real hardware and software parts, and with real people participating as part of the simulation, as shown in Figure 4‑7.

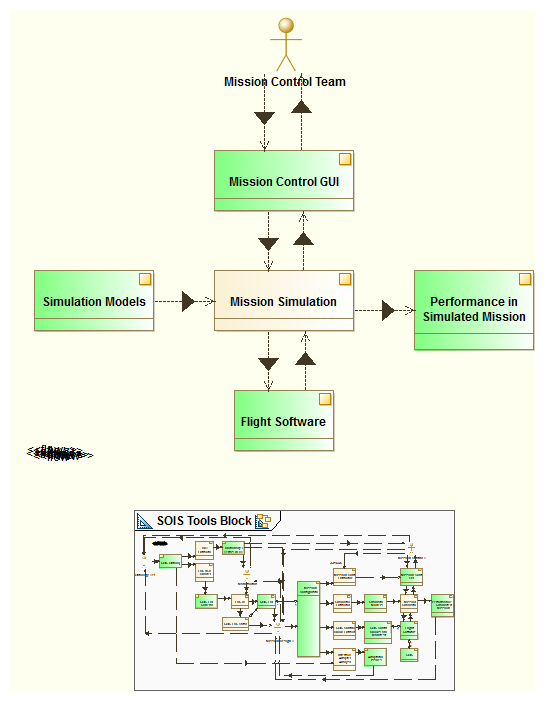


Figure 4‑7 Simulating a Mission with Reality in the Loop

The simulator prototypes described in section 4.2.5 allowed for flight software parts or actual device parts to participate actively in simulation. These additional configuration options for simulation are called “flight software in the loop” and “hardware in the loop”. The prototypes also included the mission control center in simulations. These options allow for system-testing of parts before flight, to identify difficulties that could arise during real operation.

Another topic in section 4.2.2 is the interchange of information between model-based system engineering tools and EDS's, for which some initial R&D has been done at NASA's Johnson Space Center

TBD AIT, simulators, test equipment, MCS databases –some of the stuff Stuart Fowell and Stewart Hall have put together for the EDS business development slides could be turned into text.

### Standard Interfaces

Given a standard syntax and semantics for describing interfaces of parts, it is natural to want to extend the benefits of standardization to the interfaces themselves. Such an extension would greatly simplify the task of design, because the question of matching interfaces would be reduced to matching the names of standard interfaces. EDS’s are agnostic to the concept of standard interfaces.Device Development

* Device development – this can be filled in from ESA’s “Deploying Plug-and-Play Avionics” project or at least the proposal

### Maintenance of Terminology

In addition to the standard machine-readable and human-readable syntax of EDSs, the consistent use of terminology across all EDS instances is essential for the algorithmic usage of EDSs. Terminology is a moving target, and the DoT relies upon feedback from the users of terms to keep a useful set of terms across time and space, as shown in Figure 4‑8.

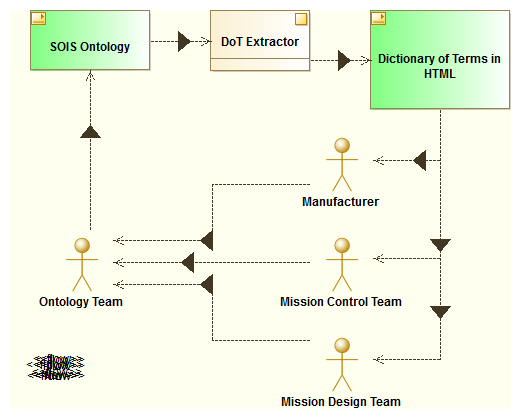


Figure 4‑8 Ontology Feedback Loop

The ontology team must listen, understand, and reconcile a variety of vocabularies and views of the subject matter, rather than to dictate a particular view of the subject matter. The manufacturer may report problems in expressions to the ontology team. The mission controllers may report ambiguities and errors to the ontology team. The designers report issues of expression and ambiguity to the ontology team.

Terms that represent synonyms or alternative factorizations may be necessary:

* e.g., “gyrometer” and “IRU”
* e.g., layered single-processor architecture and distributed message-bus architecture
* e.g., procedural specification and specification of mission objectives

Where synonyms and alternative factorizations become a part of the DoT, it becomes necessary that software tools use the ontology form of the DoT in order to have a clear representation of these relations among terms when interpreting EDSs.

# Device Data Sheet Example

This section provides some example data sheets for devices and software components in the form of t

opical illustrations how different EDS elements and attributes are used together to define a feature of a data interface of a device or software component. This section should be read in conjunction with the EDS and Common DoT standards [3, 4].

The examples in this section begin with the problem of describing a functional interface. Subsequently, the remainder of this section provides examples of implementations that can move data between a functional interface and a native interface.

Open Point: suggested features include:

* Mapping to a Raw 1553 Device
* Mapping to an ECSS 1553 Device
* Mapping to a SpaceWire device
* Mapping to MTS from both DVS and DAS

## Star Tracker EDS

This example describes an EDS for a star tracker. A summary of this star tracker EDS appears in Figure 5‑1.

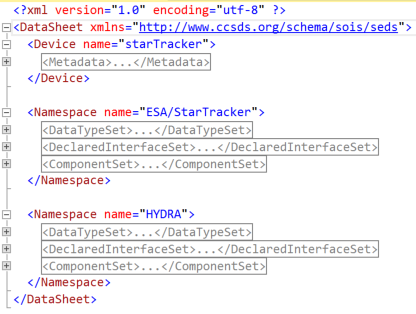


Figure 5‑1 Star Tracker EDS Summary

### Identifying the Schemas that Validate an EDS

This EDS consists of an outer element named <DataSheet>, which contains references to the schemas that validate its syntax. Only a reference to the SOIS EDS schema is needed. This is a standard facility of xml, not specific to EDS’s.[[1]](#footnote-1) The SOIS EDS schema provides the default name space for the EDS, so the element and attribute names can be written without xml namespace qualifiers.

### Reusable EDS Content

There are many definitions of data types and interfaces that appear in multiple EDS’s. To avoid repeating those definitions, the EDS namespace concept makes it possible to refer to them without copying them from the external file where they reside. References to EDS namespaces may appear in an EDS. To resolve those references, it is necessary that the toolchain software have a copy of an EDS containing the referenced EDS namespace.

The EDS in Figure 5‑1 contains references to an EDS namespace “foundation”, as can be seen in Figure 5‑5. The file named “foundation.eds.xml” defines the content of that namespace, and an EDS like it is available with the EDS schema at the SANA website. A summary of the referenced EDS appears in Figure 5‑2.



Figure 5‑2 Foundation EDS Summary

### Describing a Device in an EDS

The <Device> element in Figure 5‑1 describes the device or software component represented by the EDS. The entire <Device> element may be omitted, as in Figure 5‑2. The foundation EDS is an example of an EDS that describes only an EDS namespace.

A <Device> element includes a <Metadata> element, described elsewhere in this document, which provides information about the device that is outside the protocol for communication. The <Metadata> element can contain information about the hardware part of a device.

The content of the metadata is described in Section 5.4.

### Describing Protocol Stack

The <Namespace> elements in Figure 5‑1 contain <Component> elements that can describe protocol layers for communicating with a device. Examples of such protocol layers in SOIS are DVS, DAS, and Subnetwork. Any or all of the SOIS layers may be omitted. The components are related by matching a provided interface of one component with a required interface of another. This method of forming relationships between components is capable of defining structures other than layers; however, layers are often an appropriate and simple architecture.

Consider that many devices, such as a star tracker, are actually software objects that communicate through a protocol stack. Whether these software objects have associated hardware sensors or actuators to connect them to the real world, as does a star tracker, is optional. What matters for an EDS is that communication occurs through a protocol stack.

The content of components is described in Section 5.3.

### Describing an Interface

The description of the “ESA/StarTracker/functional” interface appears in Figure 5‑3, which expands the <DeclaredInterfaceSet> that is collapsed in Figure 5‑1.

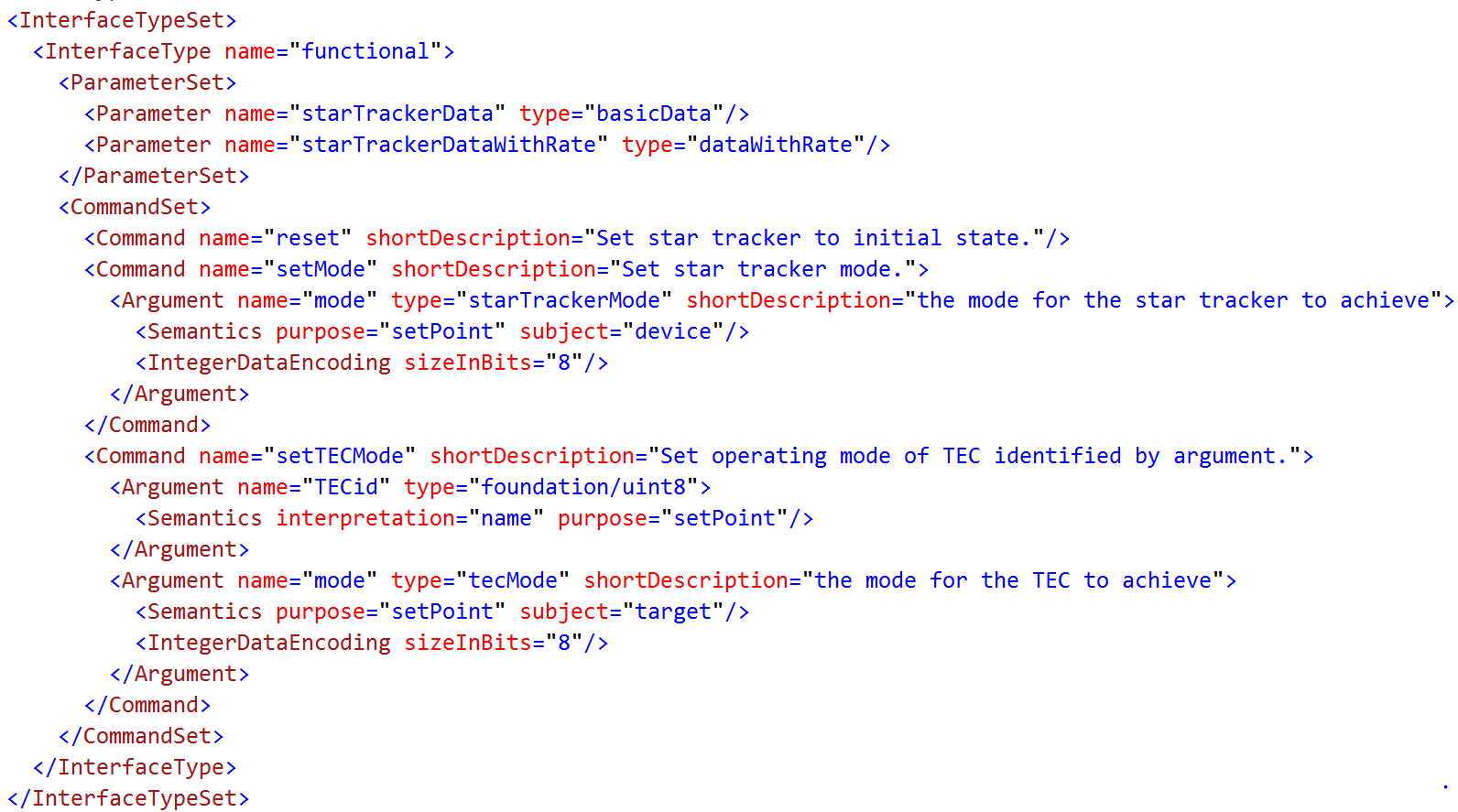


Figure 5‑3 ESA/StarTracker/functional Interface

The interface consists of a set of parameters and a set of commands. The parameters are defined by name and by reference to data types that appear elsewhere in the same EDS or in referenced EDSs. The commands are defined by name and by arguments. The arguments are defined by name and by reference to data types that appear elsewhere in the same EDS or in referenced EDSs.

### Semantics of Arguments

Before looking at those data types for parameters and arguments, it is interesting to note that the arguments may have semantic interpretations that are unique to their positional context within a command. Later, when examining the data types, we will find semantic interpretations that are associated with the data types.

For example, the “setMode” command has an argument that specifies the desired operating mode for the device. The <Semantic> element in that argument contains a “purpose” attribute with value “setPoint” to indicate that the intent of the parameter is that the mode should become the specified value. The <Semantic> element also contains a “subject” attribute with value “device” to indicate that the mode is the mode of the device receiving the command.

The “setTECMode” command is a little more complicated than the “setMode” command, because there are multiple heater controllers on the star tracker device hardware. The command has an argument with name “TECid”, which (according to its <Semantics>) should be interpreted as a name and which should set the target of the command. The other argument is “mode”, which has a <Semantic> element that says that it provides the mode setting for the target heater.

### Late Binding of Syntax

One more topic before looking at the data types for parameters and arguments: The data types of arguments may describe information without specifying a concrete syntax in which to store that data. When (or before) the data types are used as arguments in a concrete interface, they must bind to a specific syntax for representation.

The definition of “tecMode” in Figure 5‑4 says nothing about the number of bits to be used to represent the number in the enumeration. In Figure 5‑3, the number of bits for the enumerated modes is specified in the definition of the arguments. The “mode” argument refers to “tecMode”, and obtains its syntax by reference to a type named “uint8” defined in the foundation EDS. A “foundation/uint8” is an 8-bit unsigned binary integer.

### Defining an Enumeration Data Type

Discrete variables, like the operating modes in Figure 5‑3, must be defined in a way that relates their values to concepts in some domain of knowledge. An example appears in Figure 5‑4, which expands part of the <DataTypeSet> in the star tracker EDS.

The <Semantics> element of the <EnumeratedDataType> element relates the enumeration to the domain of knowledge that is encompassed by the SOIS Dictionary of Terms (DoT). It does so by setting the “enumeration” attribute to “tecOperatingMode”. The labels for the “tecOperatingMode” enumeration are defined in the DoT as “notOperational”, “controlling”, and “inBand”, with the meanings “turned off”, “seeking a set point”, and “maintaining a set point”. The enumeration for the star tracker uses only two of these enumeration labels, because it is only necessary to tell whether the heater controller is on or off. In the “on” state, the controller is either seeking or maintaining its set point. However, the “controlling” state is implicit because the “inBand” value is delivered to the star tracker through the setTECMode command in the form of a set point, effectively telling the controller to seek the “inBand” mode of operation, which may include some time in the “controlling” state. In other words, the semantics in the DoT are mapped into a simpler “off” and “on” semantics, with a clear meaning for those two states.



Figure 5‑4 Definition of TEC Operating Mode

The values of the enumeration for the “off” and “on” states are assigned in the star tracker EDS. The DoT has no information about the values that represent enumeration labels; it only defines the labels. The star tracker EDS defines both the values and the strings to represent the labels. The latter strings may appear on a mission control center console when commanding the star tracker, or when displaying its state.

The “tecMode” enumerated data type specifies that the enumerated values should be represented as an unsigned binary integer. It defers the specification of the number of bits until the data type is used as a concrete argument in a command, for example. An EDS validation tool can be written to assure that the number of possible values that can be expressed in an enumerated data type is greater than or equal to the number of labels defined in the EDS.

### Defining a Message

The parameters of the star tracker functional interface describe the messages that the star tracker can publish. The “starTrackerData” parameter in Figure 5‑3 refers to a <ContainerDataType> named “basicData”, which appears in Figure 5‑5.



Figure 5‑5 Basic Message Published by Star Tracker

The container data type describes a list of data items that are related by being in the same message together. Each item of data is represented by an <Entry> element, which names the entry and assigns a type of data to the entry.

It may translate to a structure in a programming language, but the purpose of a container data type is to describe the content of a message that passes through the protocol layers defined in an EDS. For this reason, the tools of expression for the content of a <ContainerDataType> manage the content of the container as it appears in a service data unit at that layer. The service data unit in this case is also a protocol data unit, that is a message through which the device communicates.

The specification in the preceding paragraph is not yet sufficient to write and to interpret an EDS container exactly. For example, segmentation is often handled in the OSI transport layer, which can be described in an EDS. Although Figure 3‑1 shows the transport layer outside the subnetwork layer, an EDS treats it and the network layer as part of the subnetwork layer.

* This could mean that a container in a subnetwork interface should not show segmentation data, but that is not true of all devices.
* When the segmentation mechanism is standard, and can be specified by name in the subnetwork, then there is no need for a container in a subnetwork interface to show segmentation data. If a designer wishes not to implement segmentation in the transport layer, then they will be able to consult the standard to determine the appearance of service data units with standard segmentation. Otherwise, the specification of the name of the standard with the device enables the designers to assure that their transport layer has the proper implementation of segmentation.
* When the segmentation mechanism is not standard, then there is value in describing any segmentation data in the service data units of the subnetwork, as if segmentation were not implemented in the transport layer. A local ontology will be needed to describe the segmentation mechanism. The designer using the EDS will still have the option of implementing segmentation in the transport layer, or of not doing so.

When describing interfaces of protocol layers above the physical layer, an author of an EDS may be tempted to define the content as it would appear in a familiar processor, aligned on the natural word boundaries of that processor; however, it is best that an EDS not describe that alignment, because it could vary from processor to processor. Instead, an EDS should describe the content of protocol data units. Additional data that is part of a service data unit may appear as arguments of a command.

The best practice for managing this issue is to state clearly whether

* a container describes a protocol data unit, which requires explicit description of the bit location and encoding of each item of data, or
* a container describes a data object that passes across an application programming interface within an undetermined processor, which allows abstract description of the type of each item of data.

The location of a definition of a container in an EDS does not determine whether it represents a protocol data unit, or it represents part of a service data unit. It is necessary to use an explicit attribute in the command argument definition to indicate that the container used in that argument is a protocol data unit. The attribute is “dataUnit”, and sets a Boolean value, defaulting to “false”, which is appropriate for parts of a service data unit. The attribute must be set explicitly to “true” for protocol data units.

The alignment of entries of an abstract container in a processor should remain abstract in an EDS. It is the function of a tool chain that generates software to determine alignments in a target processor for a protocol layer described in an EDS. An EDS could say nothing about this aspect of implementation for an argument marked dataUnit=”false”. An EDS should say everything about this aspect of implementation for containers marked dataUnit=”true”, which describe protocol data units.

### Applying Standard Data Types

The entries in the “basicData” container include one named “attitude”, which has a type defined in the foundation EDS, “foundation/QuaternionRxyz. The foundation definition of this kind of quaternion is shown in Figure 5‑6.

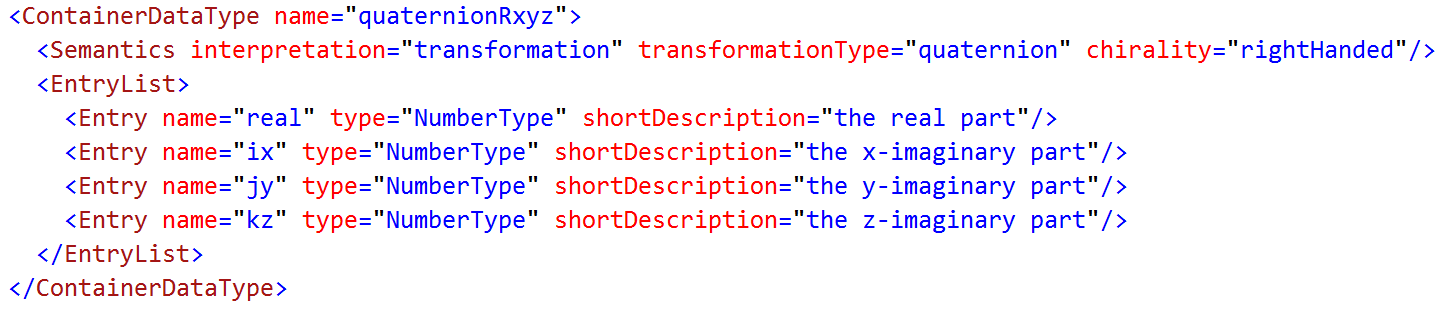


Figure 5‑6 Description of a Quaternion

The foundation EDS provides some conventions for the representation of a quaternion, in this case choosing to place the real part first in the array of four numbers. It also attaches some semantics. The <Semantic> element states that the array is interpreted as a transformation, and the algorithm to be used for the transformation is quaternion conjugation, which implies a Hamilton product. It also states that it applies to right-handed 3-D coordinates. This definition of a quaternion omits information such as the syntax of representation of the elements of the array.

The use of the quaternion in the star tracker EDS adds information that is appropriate in the context of the star tracker. One simple choice is binding the syntax of the elements of the array as double precision IEEE 754 2008 floating point numbers; that is done by the <FloatDataEncoding> element in the attitude <Entry> in Figure 5‑5. Another important bit of semantic information is the statement that the quaternion is a measurement, so it will not be accidentally used as a set point or nominal value.

The semantics in the attitude entry state that the rotation provided by the quaternion is from J2000 ECI coordinates to the device coordinate system. That information enables flight software writers to know that they can convert the attitude using the mounting rotation of the star tracker to obtain the attitude in vehicle coordinates. The mounting rotation would not appear in the star tracker EDS; instead, it would be a property of a particular star tracker in a manifest file that describes the structure of a vehicle, and is outside the scope of this discussion.

### A Message as a Relation

The data items thrown together in a message probably bear some mutual relation, just by virtue of their near-simultaneous determination, like a row in a relational table. However, there is no formal guarantee of the actual existence or meaning of such a relation. It is necessary to use <Semantic> elements to describe relationships between entries in a container, when these relationships are guaranteed by the design of the device that publishes the messages. The “timeOfAttitude” and “quaternionQualityIndex” entries both contain the semantic attribute ‘subject=”attitude”’, to show that their information concerns the “attitude” entry.

#### The Complete Star Tracker Example

The complete text of the star tracker EDS that has served as an example in the preceding discussion appears below.

<?xml version="1.0" encoding="utf-8" ?>

<DataSheet xmlns="http://www.ccsds.org/schema/sois/seds">

<Device name="starTracker">

<Metadata>

<Category name="manufacture">

<Semantics memberOf="Manufacture"/>

<MetadataValueSet>

<StringValue name="manufacturer" value="Sodern"/>

<StringValue name="modelNumber" value="Hydra"/>

</MetadataValueSet>

</Category>

<Category name="CM">

<Semantics memberOf="ConfigurationManagement"/>

<MetadataValueSet>

<StringValue name="file" value="starTracker.hydra.m.eds.xml"/>

<StringValue name="version" value="0.1.0"/>

</MetadataValueSet>

</Category>

<Category name="modops">

<Semantics memberOf="ModelOfOperation"/>

<Category name="device">

<MetadataValueSet>

<StringValue name="opticalAxis" value="hasA"/>

<StringValue name="focalPlane" value="hasA"/>

<StringValue name="mountingBase" value="hasA"/>

<StringValue name="electronicsBox" value="hasA"/>

<StringValue name="coordinateSystem" value="cartesian">

<Semantics referenceFrame="device" chirality="rightHanded"/>

</StringValue>

</MetadataValueSet>

</Category>

<Category name="opticalAxis">

<MetadataValueSet>

<FloatValue name="originZ" value="0.02">

<Semantics unit="metre" quantityKind="lengthQK"/>

</FloatValue>

<FloatValue name="alignmentZ" value="1.0">

<Semantics quantityKind="lengthQK"/>

</FloatValue>

</MetadataValueSet>

</Category>

<Category name="focalPlane">

<MetadataValueSet>

<FloatValue name="originZ" value="0.02">

<Semantics unit="metre" quantityKind="lengthQK"/>

</FloatValue>

<FloatValue name="normalZ" value="1.0">

<Semantics quantityKind="lengthQK"/>

</FloatValue>

<FloatValue name="planeRotation" value="0.0">

<Semantics quantityKind="angleQK" unit="radian"/>

</FloatValue>

</MetadataValueSet>

</Category>

<Category name="mountingBase">

<MetadataValueSet>

<FloatValue name="originZ" value="0.0">

<Semantics unit="metre" quantityKind="lengthQK"/>

</FloatValue>

<FloatValue name="normalZ" value="1.0">

<Semantics quantityKind="lengthQK"/>

</FloatValue>

<FloatValue name="planeRotation" value="0.0">

<Semantics quantityKind="angleQK" unit="radian"/>

</FloatValue>

<StringValue name="fastenerDiagram" value="http://www.sodern.com/..."/>

</MetadataValueSet>

</Category>

<Category name="electronicsBox">

<MetadataValueSet>

<FloatValue name="cableLength" value="3.0">

<Semantics quantityKind="lengthQK" unit="metre"/>

</FloatValue>

</MetadataValueSet>

</Category>

</Category>

</Metadata>

</Device>

<Namespace name="ESA/StarTracker">

<DataTypeSet>

<EnumeratedDataType name="starTrackerMode">

<Semantics interpretation="mode" enumeration="starTrackerOperatingMode" subject="device"/>

<EnumerationList>

<Enumeration label="notOperating" value="0">

<Semantics starTrackerOperatingMode="notOperational"/>

</Enumeration>

<Enumeration label="acquiring" value="1">

<Semantics starTrackerOperatingMode="starTrackerAcquiring"/>

</Enumeration>

<Enumeration label="tracking" value="2">

<Semantics starTrackerOperatingMode="starTrackerTracking"/>

</Enumeration>

<Enumeration label="rateMode" value="4">

<Semantics starTrackerOperatingMode="starTrackerReportingRates"/>

</Enumeration>

</EnumerationList>

</EnumeratedDataType>

<EnumeratedDataType name="tecMode">

<Semantics interpretation="mode" enumeration="tecOperatingMode"/>

<EnumerationList>

<Enumeration label="off" value="0">

<Semantics tecOperatingMode="notOperational"/>

</Enumeration>

<Enumeration label="on" value="1">

<Semantics tecOperatingMode="inBand"/>

</Enumeration>

</EnumerationList>

</EnumeratedDataType>

<ContainerDataType name="basicData">

<EntryList>

<Entry name="attitude" type="foundation/QuaternionRxyz" shortDescription="Final Attitude Quaternion">

<Semantics referenceFrame="ECI" coordinateType="J2000" toFrame="device" purpose="measurement"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Entry>

<Entry name="timeOfAttitude" type="foundation/float64"

shortDescription="time elapsed between the last synchronisation signal and attitude measurement">

<Semantics subject="attitude" interpretation="difference" purpose="measurement"

differenceConvention="sinceSync" quantityKind="timeQK" unit="millisecond"/>

</Entry>

<Entry name="mode" type="starTrackerMode" shortDescription="the mode of operation of the star tracker">

<Semantics purpose="measurement"/>

<IntegerDataEncoding encoding="unsigned" sizeInBits="8"/>

</Entry>

<Entry name="quaternionQualityIndex" type="foundation/float32">

<Semantics subject="attitude" interpretation="status" statusConvention="qualityFraction"

purpose="measurement"/>

<ValidRange>

<MinMaxRange min="0" max="1" rangeType="inclusiveMinInclusiveMax"/>

</ValidRange>

</Entry>

<Entry name="deviceHealth" type="foundation/HealthStatus" shortDescription="health of star tracker">

<Semantics purpose="measurement" subject="device"/>

</Entry>

</EntryList>

</ContainerDataType>

<ContainerDataType name="dataWithRate" baseType="basicData">

<EntryList>

<Entry name="angularRate" type="foundation/AngularRate" shortDescription="Final Angular Rate vector">

<Semantics referenceFrame="ECI" toFrame="device" purpose="measurement" quantityKind="angularRateQK" unit="radianPerSecond"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Entry>

<Entry name="timeOfRate" type="foundation/float64" shortDescription="Time elapsed between the last synchronisation signal and rate measurement">

<Semantics subject="angularRate" interpretation="difference" purpose="measurement" differenceConvention="sinceSync" quantityKind="timeQK" unit="millisecond"/>

</Entry>

</EntryList>

</ContainerDataType>

</DataTypeSet>

<DeclaredInterfaceSet>

<Interface name="functional">

<ParameterSet>

<Parameter name="starTrackerData" type="basicData"/>

<Parameter name="starTrackerDataWithRate" type="dataWithRate"/>

</ParameterSet>

<CommandSet>

<Command name="reset" shortDescription="Set star tracker to initial state."/>

<Command name="setMode" shortDescription="Set star tracker mode.">

<Argument name="mode" type="starTrackerMode" shortDescription="the mode for the star tracker to achieve">

<Semantics purpose="setPoint" subject="device"/>

<IntegerDataEncoding encoding="unsigned" sizeInBits="8"/>

</Argument>

</Command>

<Command name="setTECMode" shortDescription="Set operating mode of TEC identified by argument.">

<Argument name="TECid" type="foundation/uint8">

<Semantics interpretation="name" purpose="setPoint"/>

</Argument>

<Argument name="mode" type="tecMode" shortDescription="the mode for the TEC to achieve">

<Semantics purpose="setPoint" subject="target"/>

<IntegerDataEncoding encoding="unsigned" sizeInBits="8"/>

</Argument>

</Command>

</CommandSet>

</Interface>

</DeclaredInterfaceSet>

<ComponentSet>

<Component name="DVS">

<ProvidedInterfaceSet>

<Interface name="functional" type="functional"/>

</ProvidedInterfaceSet>

<RequiredInterfaceSet>

<Interface name="native" type="HYDRA/HYDRA\_DAS"/>

</RequiredInterfaceSet>

<Implementation>

<VariableSet>

<Variable name="working\_starTrackerData" type="ESA/StarTracker/basicData"/>

</VariableSet>

<ParameterMapSet>

<ParameterMap interface="functional" parameter="starTrackerData" variableRef="working\_starTrackerData"/>

<ParameterMap interface="native" parameter="attitude" variableRef="working\_starTrackerData.attitude"/>

</ParameterMapSet>

<ActivitySet>

<Activity name="read\_attitude">

<Body>

<SendParameterPrimitive interface="native" parameter="attitude" operation="get" transaction="t0"/>

</Body>

</Activity>

</ActivitySet>

<StateMachineSet>

<StateMachine name="read\_attitude" defaultEntryState="idle">

<State name="idle"/>

<State name="active">

<OnEntry activity="read\_attitude"/>

</State>

<Transition fromState="active" toState="active" name="receive">

<OnParameterPrimitive interface="native" parameter="attitude" operation="get" transaction="t0"/>

</Transition>

<Transition fromState="idle" toState="active" name="request">

<OnParameterPrimitive interface="functional" parameter="starTrackerData" operation="get" transaction="t0"/>

</Transition>

</StateMachine>

</StateMachineSet>

</Implementation>

</Component>

</ComponentSet>

</Namespace>

<Namespace name="HYDRA">

<DataTypeSet>

<FloatDataType name="rawtemperature" shortDescription="a temperature reading from a thermistor">

<Semantics purpose="measurement" quantityKind="thermodynamicTemperatureQK" unit="count"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_single"/>

<Range>

<PrecisionRange>single</PrecisionRange>

</Range>

</FloatDataType>

<ContainerDataType name="hydraHousekeeping">

<EntryList>

<Entry name="eTemp" type="rawTemperature">

<Semantics subject="device.hasA.electronicsBox"/>

</Entry>

<Entry name="fTemp" type="rawTemperature">

<Semantics subject="device.hasA.focalPlane"/>

</Entry>

</EntryList>

</ContainerDataType>

</DataTypeSet>

<DeclaredInterfaceSet>

<Interface name="HYDRA\_DAS">

<ParameterSet>

<Parameter name="attitude" type="foundation/QuaternionRxyz" shortDescription="Final Attitude Quaternion">

<Semantics referenceFrame="ECI" coordinateType="J2000" toFrame="device" purpose="measurement"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Parameter>

<Parameter name="housekeeping" type="hydraHousekeeping"/>

</ParameterSet>

</Interface>

<Interface name="HYDRA\_Subnet">

<SubnetInterface>

<SpaceWireSubnet spaceWireLinkState="SpaceWireAlwaysEnabled">

<ReceiveRate minInclusive="10" maxInclusive="10"/>

<SpaceWireRMAP>

<SpaceWireLogicalAddressValidRange minInclusive="128" maxInclusive="128"/>

<SpaceWireRMAPAddressValidRange minInclusive="2000" maxInclusive="3024"/>

</SpaceWireRMAP>

<SpaceWireCustomProtocol protocolId="254">

<SpaceWireLogicalAddressValidRange minInclusive="132" maxInclusive="132"/>

</SpaceWireCustomProtocol>

<TransmitRate minInclusive="10" maxInclusive="10"/>

</SpaceWireSubnet>

</SubnetInterface>

</Interface>

</DeclaredInterfaceSet>

<ComponentSet>

<Component name="DAS">

<ProvidedInterfaceSet>

<Interface type="HYDRA\_DAS" name="HYDRA\_DAS"/>

</ProvidedInterfaceSet>

<RequiredInterfaceSet>

<Interface type="CCSDS/SOIS/Subnetwork/MASInterfaceType" name="SubnetworkMAS"/>

</RequiredInterfaceSet>

<Implementation>

<VariableSet>

<Variable name="working\_attitude" type="foundation/QuaternionRxyz" shortDescription="Final Attitude Quaternion">

<Semantics referenceFrame="ECI" coordinateType="J2000" toFrame="device" purpose="measurement"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Variable>

</VariableSet>

<ParameterMapSet>

<ParameterMap interface="HYDRA\_DAS" parameter="attitude" variableRef="working\_attitude"/>

</ParameterMapSet>

<ActivitySet>

<Activity name="read\_attitude">

<Body>

<SendCommandPrimitive interface="SubnetworkMAS" command="read" transaction="t24">

<ArgumentValue name="MemoryID">

<Value value="24"/>

</ArgumentValue>

<ArgumentValue name="MemoryAddress">

<Value value="0"/>

</ArgumentValue>

<ArgumentValue name="data">

<VariableRef variableRef="working\_attitude"/>

</ArgumentValue>

</SendCommandPrimitive>

</Body>

</Activity>

</ActivitySet>

<StateMachineSet>

<StateMachine name="read\_attitude" defaultEntryState="idle">

<State name="idle"/>

<State name="busy"/>

<Transition fromState="idle" toState="busy" name="request">

<OnParameterPrimitive interface="HYDRA\_DAS" parameter="attitude" operation="get" transaction="t24"/>

<Do activity="read\_attitude"/>

</Transition>

<Transition fromState="busy" toState="idle" name="receive">

<OnCommandPrimitive interface="SubnetworkMAS" command="read" transaction="t24"/>

</Transition>

</StateMachine>

</StateMachineSet>

</Implementation>

</Component>

</ComponentSet>

</Namespace>

</DataSheet>

The text of the foundation EDS appears below.

<?xml version="1.0" encoding="utf-8" ?>

<DataSheet xmlns="http://www.ccsds.org/schema/sois/seds">

<Namespace name="foundation">

<DataTypeSet>

<ContainerDataType name="angularRate">

<Semantics interpretation="transformation" transformationType="rotationRate" chirality="rightHanded"/>

<EntryList>

<Entry name="rx" type="NumberType" shortDescription="rotation rate around x axis"/>

<Entry name="ry" type="NumberType" shortDescription="rotation rate around y axis"/>

<Entry name="rz" type="NumberType" shortDescription="rotation rate around z axis"/>

</EntryList>

</ContainerDataType>

<ContainerDataType name="quaternionRxyz">

<Semantics interpretation="transformation" transformationType="quaternion" chirality="rightHanded"/>

<EntryList>

<Entry name="real" type="NumberType" shortDescription="the real part">

<Semantics subject="quaternion.hasA.realPart"/>

</Entry>

<Entry name="ix" type="NumberType" shortDescription="the x-imaginary part">

<Semantics subject="quaternion.hasA.xPart"/>

</Entry>

<Entry name="jy" type="NumberType" shortDescription="the y-imaginary part">

<Semantics subject="quaternion.hasA.yPart"/>

</Entry>

<Entry name="kz" type="NumberType" shortDescription="the z-imaginary part">

<Semantics subject="quaternion.hasA.zPart"/>

</Entry>

</EntryList>

</ContainerDataType>

<EnumeratedDataType name="HealthStatus">

<Semantics interpretation="status" statusConvention="ternaryStatus" enumeration="healthStatus"/>

<IntegerDataEncoding encoding="unsigned"/>

<EnumerationList>

<Enumeration label="OK" value="1">

<Semantics healthStatus="healthGood"/>

</Enumeration>

<Enumeration label="warning" value="2">

<Semantics healthStatus="healthWarning"/>

</Enumeration>

<Enumeration label="severeWarning" value="3">

<Semantics healthStatus="healthSevereWarning"/>

</Enumeration>

</EnumerationList>

</EnumeratedDataType>

<IntegerDataType name="uint8">

<IntegerDataEncoding encoding="unsigned" sizeInBits="8"/>

<Range>

<MinMaxRange rangeType="inclusiveMinInclusiveMax" min="0" max="255"/>

</Range>

</IntegerDataType>

<FloatDataType name="float32">

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_single"/>

<Range>

<PrecisionRange>single</PrecisionRange>

</Range>

</FloatDataType>

<FloatDataType name="float64">

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

<Range>

<PrecisionRange>double</PrecisionRange>

</Range>

</FloatDataType>

</DataTypeSet>

</Namespace>

</DataSheet>

## Calibrating Raw Parameters

This section describes the top two layers of the protocol stack for an inertial rate unit. In this example, the communication between the top protocol layer (DVS) and the penultimate layer (DAS) has been simplified to a single calibrator, in order to provide a minimalist example for linking the top two layers of the protocol stack.

A calibration function maps values from the native measurement of a device to engineering units. In this example, the engineering units appear in the functional interface of the EDS.

A calibration function can be invoked by the following means.

* As the content of a conditional element in an activity
* As the content of an iteration element in an activity
* As an activity
* As the content of a mapping element
* Implicitly when the raw value changes (There seems to be no explicit way to specify this; perhaps it occurs only when there is not a finite state machine nor a mapping to determine when it should occur.)

The description of a calibrator in an EDS specifies that the DVS interface should present engineering units, rather than raw counts. However, the designer of a vehicle may use their discretion in the question of whether to implement the DVS layer on board the vehicle or elsewhere. Some space agencies may choose to implement the DVS layer on the ground, for the benefit of the ground controllers, and to train the flight software to use raw counts, in order to minimize the risk of implementing a calibrator in flight software.

The following example converts a rate measurement of an inertial rate unit to engineering units, starting with the DataSheet element.

<DataSheet xmlns:seds="http://www.ccsds.org/schema/sois/seds">

The angularRate container data type that follows defines the general form of an angular rate vector, without binding it to a particular syntax of number, nor to a particular unit of measure. The quantity kind says that it represents an angular rate.

<ContainerDataType name="angularRate">

<Semantics interpretation="transformation" quantityKind="angularRateQK"

transformationType="rotationRate" chirality="rightHanded"/>

<EntryList>

<Entry name="rx" type="NumberType"

shortDescription="rotation rate around x axis"/>

<Entry name="ry" type="NumberType"

shortDescription="rotation rate around y axis"/>

<Entry name="rz" type="NumberType"

shortDescription="rotation rate around z axis"/>

</EntryList>

</ContainerDataType>

…

</DataTypeSet>

<DeclaredInterfaceSet>

<InterfaceType name="rawRateIType">

<ParameterSet>

The rate parameter that follows is an angular rate, as defined above. This parameter has a unit of measure that is the counts from an analog-to-digital converter.

<Parameter name="rate" type="angularRate"

shortDescription="rate vector">

<Semantics purpose="measurement" coordinateType="J2000"

referenceFrame="device" relationToTimestamp="generation"

unit="aToDCount"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Parameter>

…

</ParameterSet>

</InterfaceType>

<InterfaceType name="rateIType">

<ParameterSet>

The rate parameter that follows is another angular rate, as defined above. This parameter has a unit of measure that is radians per second.

<Parameter name="rate" type="angularRate"

shortDescription="rate vector ">

<Semantics purpose="measurement" coordinateType="J2000"

referenceFrame="device" relationToTimestamp="generation"

unit="radianPerSecond"/>

<FloatDataEncoding encodingAndPrecision="IEEE754\_2008\_double"/>

</Parameter>

…

</ParameterSet>

…

</InterfaceType>

…

</DeclaredInterfaceSet>

<ComponentTypeSet>

The component type that follows defines the native data interface of the inertial rate unit.

<ComponentType name="IRUNative">

<ProvidedInterfaceSet>

<Interface name="rawRate" type="rawRateIType"/>

…

</ProvidedInterfaceSet>

…

</ComponentType>

…

The component type that follows defines the functional interface of the inertial rate unit.

<ComponentType name="inertialRateUnit">

<ProvidedInterfaceSet>

<Interface name="rate" type="rateIType"/>

…

</ProvidedInterfaceSet>

…

<Implementation>

<ActivitySet>

<Activity name="rateConversion">

<Implementation>

The calibrator that follows defines how to convert numbers in the native data interface into numbers in the functional interface, as specified by the input variable reference and the output variable reference. The variables referenced are vectors, not numbers, so the effect of the calibrator distributes over the numbers in the vector. (This distribution of calibrator is not defined in the EDS red book.)

<Calibration inputVariableRef="IRUNative.rawRate.rate"

outputVariableRef="rate.rate">

<PolynomialCalibrator>

<Term coefficient="2.71828" exponent="0"/>

<Term coefficient="3.14159" exponent="1"/>

</PolynomialCalibrator>

</Calibration>

</Implementation>

</Activity>

</ActivitySet>

</Implementation>

</ComponentType>

…

</ComponentTypeSet>

</Namespace>

</DataSheet>

The calibrator in the example above is invoked implicitly when the input variable changes value, because there is an activity map to determine when to invoke the calibrator.

## Mapping Native interface to Functional Interface

The <Component> element named “DVS” in the “ESA/StarTracker” namespace in Figure 5‑1 is expanded in Figure 5‑7.



Figure 5‑7 A Component Providing a Functional Interface

A component describes the data interfaces provided and required by the device. In this example, only a provided interface appears, to keep the discussion simple. The interface provided at this level in the protocol stack would be a functional interface that would be used by application software. The <Interface> element refers to an <InterfaceType> element, and associates the name “functional” with that interface. The interface type is defined in the same EDS name space, so the simple name of the interface type, “functional”, is sufficient to refer to the interface type whose fully qualified name would be “ESA/StarTracker/functional”.

This section returns to the star tracker functional interface EDS of section 5.1, and adds a native interface to produce a simple EDS that describes a particular star tracker. The particular star tracker is the namespace named Hydra in Figure 5‑1. The native interface is incomplete here, but there is sufficient material present to demonstrate the function of a state machine implementation that moves data between the native interface and the functional interface.

### Content of DVS Component

The relevant content of the DVS Component appears in summary in Figure 5‑8. The RequiredInterfaceSet element contains a required interface, which is the top interface of the DAS protocol layer. That interface is called “HYDRA/HYDRA\_DAS”; the name consists of the name space name “HYDRA” and the interface name within that name space “HYDRA\_DAS”. This statement says that the DVS component will communicate with the device specific DAS component. The communication between protocol layers can be described by means of service primitives in the ISO protocol model. Those primitives are called “ParameterPrimitiveSinkType”, “CommandPrimitiveSinkType”, “ParameterPrimitiveSourceType”, and “CommandPrimitiveSourceType” in an EDS.

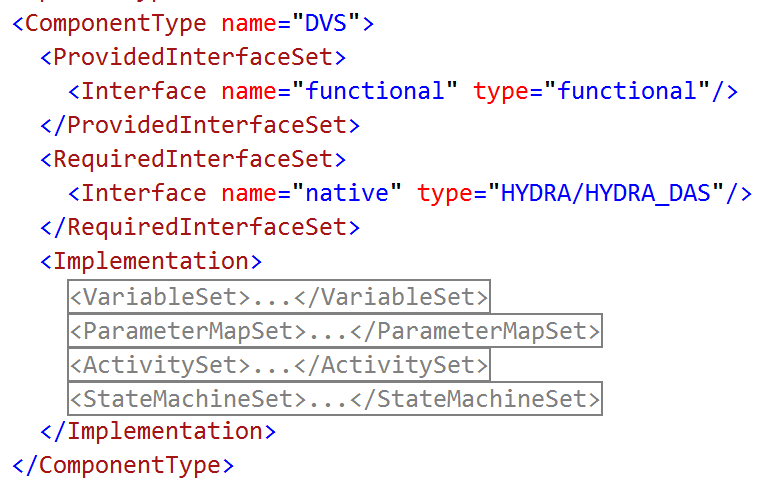


Figure 5‑8 Summary of Content of DVS Component

The Implementation element in Figure 5‑8 contains a variable set, a parameter map set, an activity set, and a state machine set. The variable set contains the working memory of the DVS protocol layer. The parameter map set associates items of data in an interface with variables in the variable set, with the understanding that a change in one implies that the other changes to the same value. The activity set contains procedures that can be executed to implement the functions of DVS; the timing of execution of those procedures is determined by the state machine set. The multi-threaded asynchronous processes in the DVS are described by state machines.

In general, the purpose of the content of the Implementation element is to describe procedures without implying or favouring any particular programming language, computer processor architecture, or memory word size.

### Device Abstraction Control Procedure

An expansion of the variable set and the parameter map set appears in Figure 5‑9.

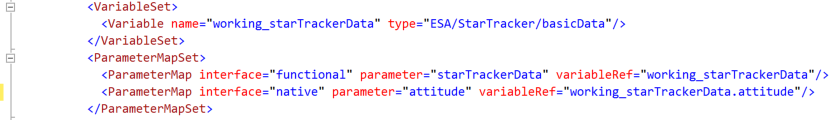


Figure 5‑9 Variable Map for Star Tracker DVS

One variable is defined and mapped to data in two interfaces. The variable “working\_starTrackerData” holds a message whose “attitude” entry maps to the “attitude” parameter in the DAS interface. The entire message maps to the “starTrackerData” in the functional interface in the DVS.

The procedural part of the DVS protocol layer appears in Figure 5‑10.

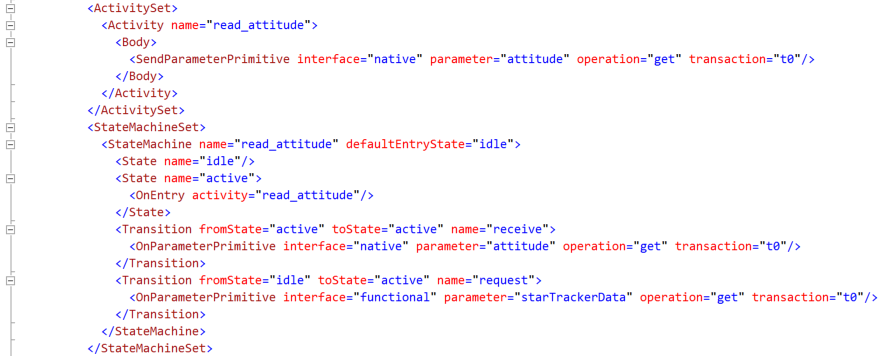


Figure 5‑10 Procedure for Star Tracker DVS

The state machine responds to events received from the DAS protocol layer, and from the application that uses the DVS protocol layer. These events appear as OnParameterPrimitive elements that cause the state machine to change from one state to another. The state machine has two states, “idle” and “active”. It changes from idle to active when the application first receives a starTrackerData message through the functional interface of DVS. Afterward, it remains in the active state until the software is reloaded, if ever. The state machine transitions from active to active when there is an attempt to read the attitude data item in the HYDRA\_DAS interface. The attempt to read that attitude data item is represented by the Parameter element in the activity “read\_attitude”, which is designated to executed whenever the state machine enters the “active” state. The OnEntry element in the state named “active” specifies to execute the “read\_attitude” activity whenever the state machine enters that state. The result is that the state machine is continually attempting to read the attitude data item in the HYDRA\_DAS interface. (This description of the implementation should be more interrupt driven, instead of just spinning in the active state. The reading of the starTrackerData item in the functional interface or a clock should cause the read\_attitude activity to execute, or the operation could be “set” to cause the read\_attitude activity to execute.)

### Device Access Service for Star Tracker

The interface between DAS and DVS appears in Figure 5‑11.



Figure 5‑11 Interface between DAS and DVS

The interface contains an item of data named “attitude”, which is a quaternion computed by the star tracker. The actual star tracker produces much more data in this interface, but for this presentation, the interface is simplified.

The component type in Figure 5‑11 provides the HYDRA\_DAS interface that is required by the DVS protocol layer. This DAS layer requires the subnetwork interface of the star tracker, which is a form of memory access service. The subnetwork protocol layer will be discussed later.

### Device-Specific Access Procedure

The variable map for the DAS protocol layer appears in Figure 5‑12.

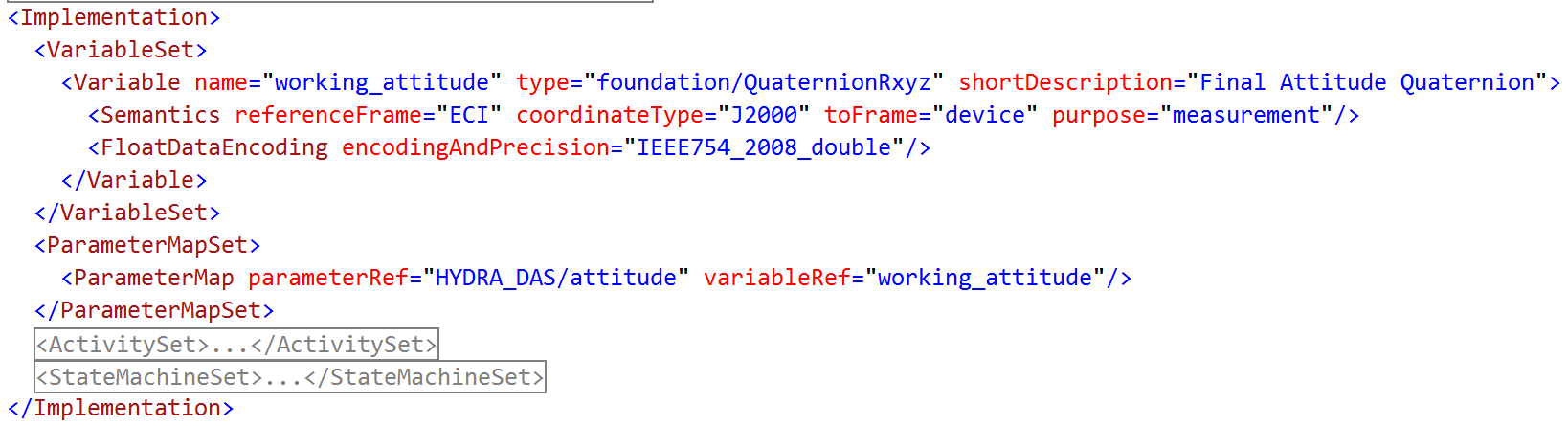


Figure 5‑12 Variable Map for Star Tracker DAS

There is just one variable for DAS, given its simplified interface, and that variable maps directly to the interface.

The procedure for DAS appears in Figure 5‑13.



Figure 5‑13 Procedure for Star Tracker DAS

The procedure consists of a simple state machine that alternates between idle and busy states, starting idle. When DVS attempts to read the attitude data item in the HYDRA\_DAS interface, it is an event that causes the state machine to change to the “busy” state, and to execute the activity “read\_attitude”. The activity is a memory access service read that stores its result in the variable named “working\_attitude”. When the confirmation for that request comes back to DAS, it causes the state machine to switch back to the idle state.

## Star Tracker Metadata

This section coonsiders metadata in the star tracker example. A summary of the star tracker EDS with metadata appeared in Figure 5‑1

### Metadata Sections

Three purposes have been identified for metadata, but the syntax for metadata remains sufficiently flexible to accommodate additional purposes. Each purpose occupies a category element in the metadata, as shown in Figure 5‑14.

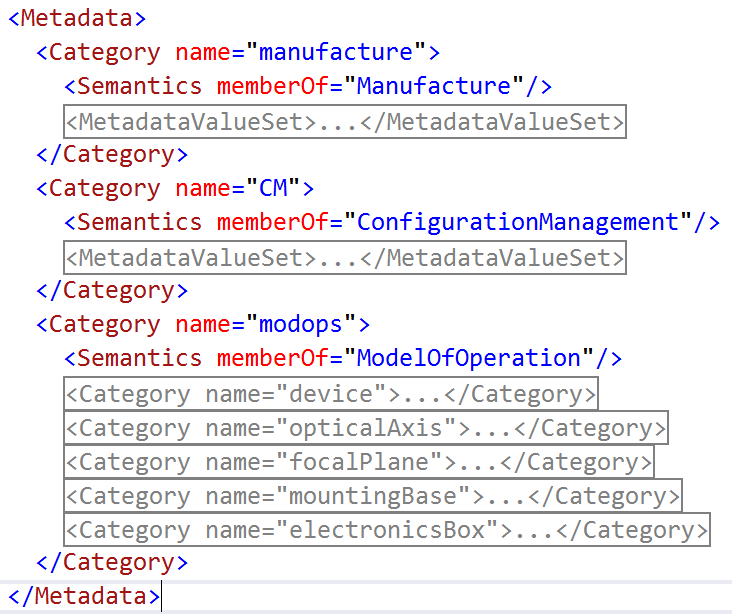


Figure 5‑14 Metadata Categories

The categories are explained in greater detail below. The Semantics attribute “memberOf” connects each category with its purpose, as described in the dictionary of terms.

The information about the manufacture of the device described by the EDS appears in the category that is a member of the “Manufacture” metadata. An example of the content of that category appears in Figure 5‑15.

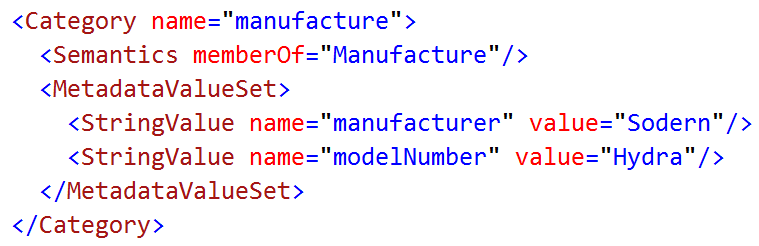


Figure 5‑15 Manufacturer Metadata

The basic information about the manufacture of a device is the name of the manufacturer and that manufacturer’s model number for the device. (Additional information, such as contacting the manufacturer, is to be determined.)

A category of metadata is reserved for the purpose of imbedding configuration management information in an EDS. See Figure 5‑16.

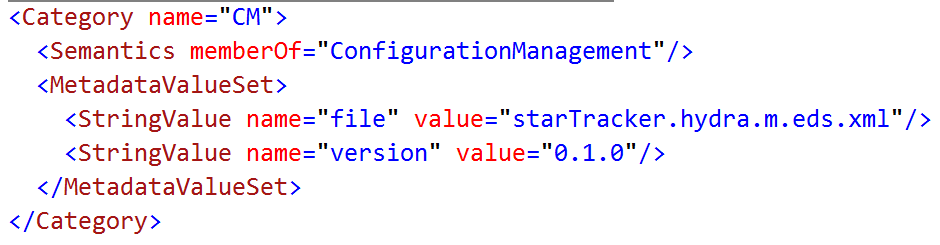


Figure 5‑16 Configuration Management Metadata

The content of configuration management metadata depends upon the software tool used for configuration management. A file name and a version number appears in the example.

The metadata that is important for interpretation of some of the data in the interfaces described in the EDS is the model of operations. An example of this metadata for a star tracker is shown in Figure 5‑17.

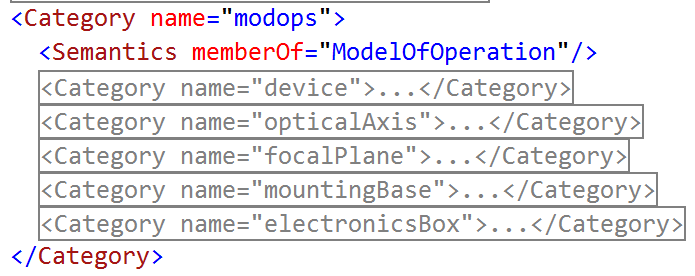


Figure 5‑17 A Model of Operations

The elements of the model of operations identify concepts that are important for description of the device. Standard versions of these models will be defined in the dictionary of terms, but they may also be defined in an EDS before the standard versions are available or for nonstandard models. The example in Figure 5‑17 is one that is defined in the EDS. The basic concept is the category named “device”, which is expanded in Figure 5‑18.

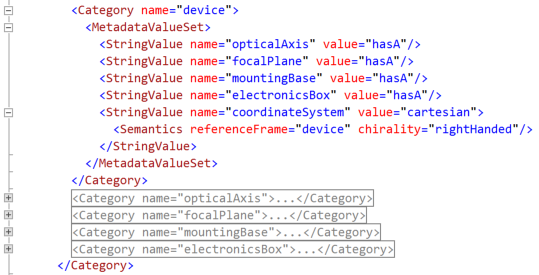


Figure 5‑18 The Device in the Model of Operations

A set of string values with the value “hasA” relate the device concept to the other concepts in the model of operations. A string value named “coordinateSystem” defines the coordinate system of the device, using semantic attributes.

The focal plane concept is expanded in Figure 5‑19.

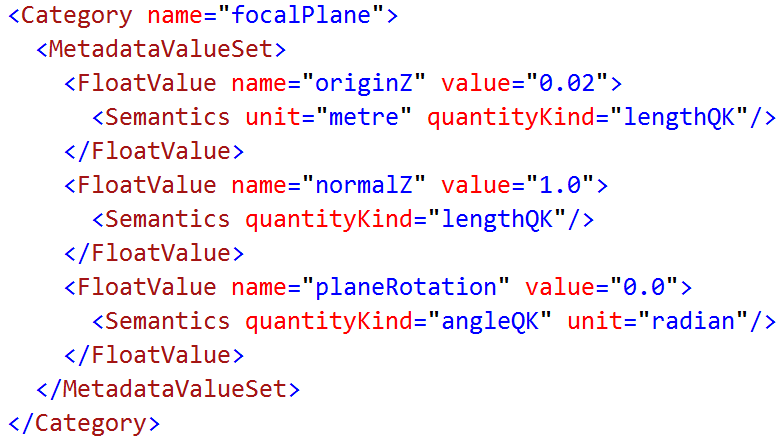


Figure 5‑19 The Focal Plane in the Model of Operations

The metadata value set for the focal plane positions its origin two centimetres from the origin of the device coordinate system along the Z axis, orients its normal in the positive Z direction in the device coordinate system, and indicates that there is no rotation from the device X and Y coordinates to those coordinates of the focal plane. This information would enable calculation of attitude in device coordinates, if the device were operating as a star camera instead of a star tracker.

The other concepts in the model of operations are similar. Only geometric metadata appears in these examples, but other physical features could be described. For example, the mounting plane could have a metadata value that indicates that the manufacturer intends that the mounting plane should function to conduct excess heat from the device.

### Usage of Metadata

Semantic tags in the namespaces in an EDS can make reference to the model of operations in the metadata. To provide an example of this usage, we define a small amount of housekeeping data that the star tracker could report, in Figure 5‑20.

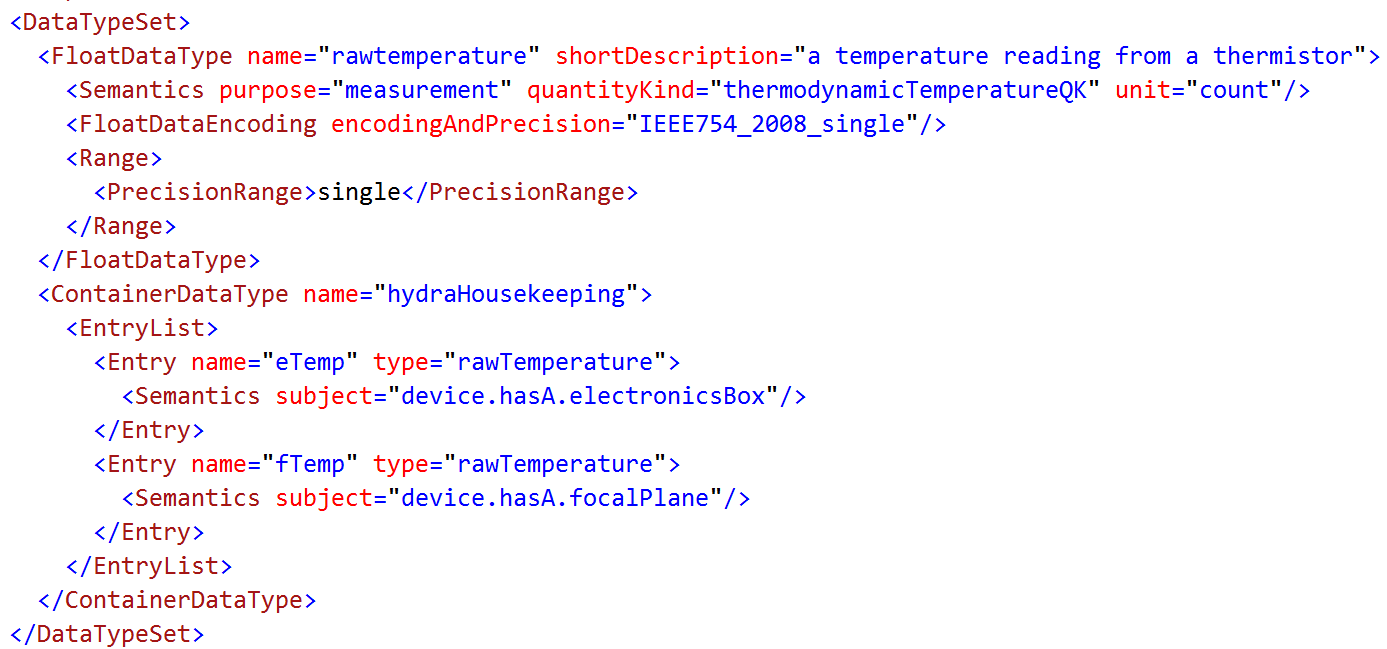


Figure 5‑20 Housekeeping Data for the Star Tracker

The housekeeping data will be reported through the container named “hydraHousekeeping”. It contains two items that represent raw temperature counts from analog-to-digital converters that digitize the resistance of thermistors.

Since this housekeeping data is not in engineering units, it would be expected that there would be calibrators that convert the counts to degrees centigrade in the DVS implementation. The calibration function would be approximate in an EDS that applies to all devices in a manufacturer’s model number. That kind of EDS would be used during engineering. At the time of integration, each real device would arrive with its own calibration numbers.

Without the model of operations, it would be impossible to say what was measured by each thermistor. However, the model of operations makes it possible to trace a path from the device to another concept in the model by following a relationship between the concepts. In this case, the relationship is the “hasA” relation that connects the device to its parts. Given that information, it is possible to say that the “eTemp” number refers to the electronics box, and the “fTemp” number refers to the focal plane.

# Available Reference Material

This section provides a source point to the reference material available on EDS and Common DoT, to be used as a starting point for using EDS and implementing tools supporting the use of EDS.

TBD including:

* Training material, i.e. presentation;
* XML files for the example data sheets in section 5;
* Open source EDS tools and libraries, including:
  + EDS import/export library;
  + EDS validator – reports if an input EDS is correct (more than just compliant to the EDS XML schema);
  + EDS creation, editing;
  + Document generation – generate HMTL view of the EDS;
  + EDS device driver generator.

# Security

One of the questions that arises when considering the security of a system whose interfaces are defined by EDSs, is whether the reduction in difficulty of connecting parts makes it easier for an intruder to gain access to the system. Clearly, the use of published interfaces eliminates one of the difficulties that an intruder would face. Difficulty is the general mechanism used to secure a data processing system, so it would seem that any facilitation of connecting parts would decrease the security of an assembly of parts. In other words, any attempt to reduce the cost of engineering an assembly of parts would decrease the security of the assembly. According to this line of reasoning, a secure design would have to be hand-crafted without any published interfaces. The consideration that is missed by strictly following this line of reasoning is the degree of difficulty faced by an intruder. Other mechanisms of security, such as passwords and encryption are significantly more difficult to defeat than a secret interface. The cost benefit of using a published interface can outweigh the risk of intrusion in many cases, where other stronger security mechanisms are in place.

More, TBD

1. ACRONYMS and Abbreviations

This annex identifies and defines the acronyms and abbreviations used in this Report.

API Application Programming Interface

CCSDS Consultative Committee for Space Data Systems

CDAS Command and Data Acquisition Services

DACP Device Abstraction Control Procedure

DAP Device-specific Access Protocol

DAS Device Access Service

DoT Dictionary of Terms

DVS Device Virtualisation Service

EDS Electronic Data Sheet

ISO International Standards Organisation

MIB Management Information Base

MTS Message Transfer Service

OSI Open Systems Interconnection

PDU Protocol Data Unit

SOIS Spacecraft Onboard Interface Services

XML eXtensible Markup Language

1. The Namespace elements in Figure 18 shows that an EDS can define something called a name space; an xml name space is a similar concept, but not the same as an EDS name space. An EDS name space is a set of names defined by an EDS to describe a device; an xml name space is a set of names defined in a schema to describe the syntax of an xml file. [↑](#footnote-ref-1)