Mission Specification Schemas

Developer's Manual

Code : S2G-DME-TEC-SUM092
Issue : 1.D
Date : 19/08/2020

	Name	Function	Signature
Prepared by	Sérgio Saraiva Eduardo Lopes Hugo Rosado	Project Manager Project Engineer Project Engineer	
Reviewed by	Antonio Gutiérrez	Technical Consultant	
Approved by	Sérgio Saraiva	Project Manager	
Signatures and approvals on original			

DEIMOS Engenharia S.A.
Av. D. João II, Lote 1.17.01, Edifício Torre Zen, 10º
1998-023 Lisboa, PORTUGAL
Tel.: +351 21 893 3010 / Fax: +351 21 896 9099
E-mail: deimos@deimos.com.pt

This page intentionally left blank

Document Information

Contract Data	
Contract Number:	4000104594/11/NL/CT/ef
Contract Issuer:	ESA/ESTEC

Internal Distribution		
Name	Unit	Copies
Internal Confidentiality Level (DME-COV-POL05)		
Unclassified ☐	Restricted ☑	Confidential ☐

External Distribution		
Name	Organisation	Copies
Michele Zundo	ESA	1 (electronic)

Archiving	
Word Processor:	MS Word 2000
File Name:	S2G-DME-TEC-SUM092-1D.doc

Document Change Log

Issue	Change description	Date	Pages Affected
1.A	First issue of the document.	12/11/2015	All
1.B	Porting guide from v2.2.X to v2.3.X added.	06/06/2016	Annex II
1.C	Add the section 3.3	20/04/2020	§ 3.3
1.D	Add Annex 3, porting guide from 2.4.X to 2.5.X	19/08/2020	Annex III

Table of Contents

List of Tables

List of Figure

1. INTRODUCTION

The Space to Ground Data Viewer (S2G) [AD.1, AD.2, AD.3, AD.4, AD.5, AD.6, AD.7] is an extensible utility tool to support ground systems engineers during the test campaigns to inspect the contents of the communication channels between the signal-in-space and the ground systems apparatus. The Space to Ground testing comprise the analysis and visualisation of a variety of telemetry data files produced by satellites. These files can be formatted as CADUs, TFs or ISPs. The S2G Data Viewer has been implemented to support these activities.

The DFDL for Space (DFDL4S) is the underlying software library used by S2G. It comprises the capability to use DFDL schemas [RD.1] to read, parse, interpret and update CADU, TF or ISP data files.

1.1. Purpose

The objective of this manual is to provide detailed information on how to setup Mission Specification Schemas to configure S2G and DFDL4S behaviour for a given mission.

The intended readerships for this document are model developers and scientists that have the requirement to access telemetry data. This document is also useful to software engineers responsible of the testing stage.

1.2. Scope

The following sections of this document are organized as follows:

- Section 2 lists applicable and reference documents
- Section 3 provides a detailed description of the Mission Configuration Files

1.3. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

Acronym	Description
CADU	Channel Access Data Unit
DME	DEIMOS Engenharia
GUI	Graphical User Interface
ISP	Instrument Source Packet
S2G	Space to Ground
SoW	Statement of Work

This page intentionally left blank

2. RELATED DOCUMENTS

2.1. Applicable Documents

The following table specifies the applicable documents that shall be complied with during project development.

Table 1: Applicable documents

Reference	Code	Title	Issue
[AD.1]	S2G-DME-TEC-TNO005	S2G Data Viewer Technical Note: Technical Specification	1.A
[AD.2]	S2G-DME-RCR-ECP032	S2G Data Viewer: Proposal for CCN1 Activities	1.B
[AD.3]	S2G-DME-RCR-ECP056	S2G Data Viewer: Proposal for CCN2 Activities	1.C
[AD.4]	S2G-DME-RCR-ECP075	S2G Data Viewer: Proposal for CCN3 Activities	1.B
[AD.5]	S2G-DME-RCR-ECP094	S2G Data Viewer: Proposal for CCN5 Activities	1.B
[AD.6]	S2G-DME-RCR-ECP111	S2G Data Viewer: Proposal for CCN7 Activities	1.A
[AD.7]	S2G-DME-RCR-ECP117	S2G Data Viewer: Proposal for CCN8 Activities	1.A

2.2. Reference Documents

The following table specifies the reference documents that shall be taken into account during project development.

Table 2: Reference documents

Reference	Code	Title	Issue
[RD.1]	S2G-DME-TEC-TNO014	Technical Note: DFDL for S2G	1.E
[RD.2]	ECSS E-70-41	Ground systems and operations - Telemetry & telecommand packet utilisation	
[RD.3]	S2G-DME-TEC-SUM023	S2G Data Viewer User Manual	1.F
[RD.4]	GFD.207	Data Definition Language (DFDL) v.1.0 Specification	
[RD.5]	S2G-DME-TEC-SUM078	DFDL4S library Developer's Manual	1.E

This page intentionally left blank

3. MISSION SPECIFICATION SCHEMAS

3.1. Introduction

Satellite house-keeping telemetry or science instruments data is transmitted to the ground sensor stations in a packets hierarchy (see Figure 1) that is defined according to a standard format, e.g. [RD.2]. Based on that standard format, each mission customizes the packets hierarchy to according to its specific needs and instruments.

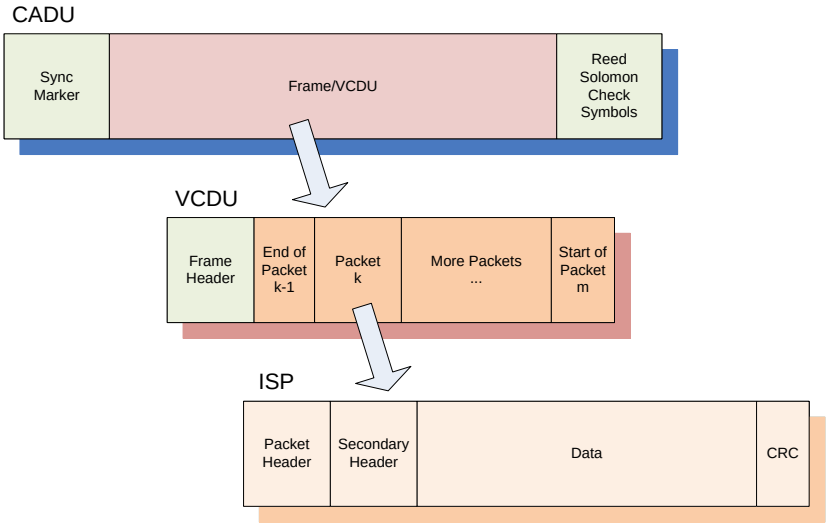


Figure 1: Hierarchy of Data received by the Ground Sensor Stations

The Space to Ground Data Viewer (S2G) displays the contents of the communication channels between the signal-in-space and the ground systems apparatus. Data files parsing is done thru DFDL4S (DFDL for Space) library. This library interprets files containing concatenated CADUs, TFs or ISPs, and lists of available data units and displays the fields and associated values inside each data unit.

3.2. Mission Configuration

S2G takes a set of mission configurations as inputs, composed of several separate files. The tool provides a set of default mission configurations that the user can extend or modify by following the procedure described in section 3.3.

The 'jar' files composing the Mission Configuration (Figure 2) provide a wide range of configuration parameters used by the application, and can be divided in two groups:

- the Mission Definition file is an xml file that contains the mission definition parameters used by the GUI (such as mission name, the list of searchable or hidden fields); this file also contains the reference to the schemas defining the structure of the binary data.
- the Mission Data Definition schema files (*.xsd) are a set of schemas that define the binary contents of the several levels of packages (CADU, VCDU and ISP) based on the DFDL [RD.1].

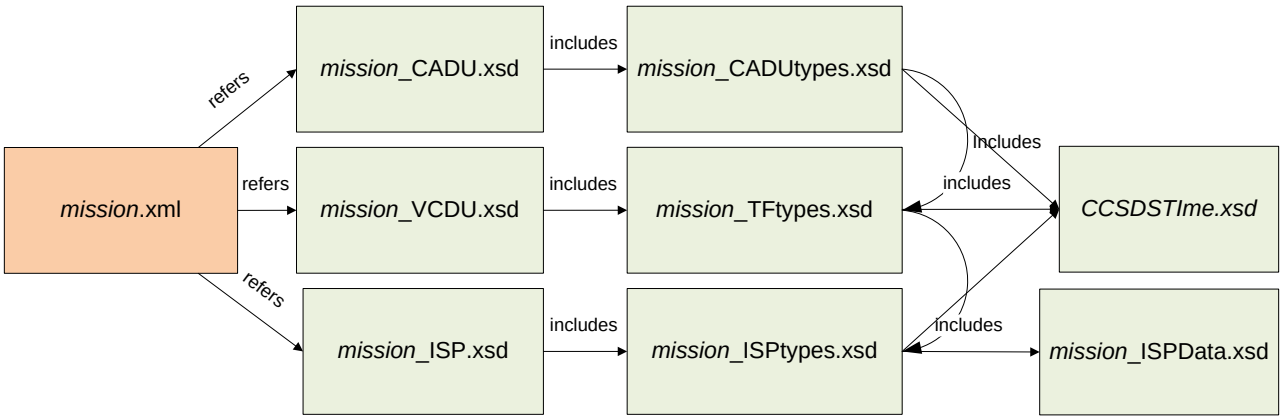


Figure 2: Mission Configuration files structure

The following sections present details of the two types of configuration files.

3.2.1. Mission Definition File

The Mission Definition file is an xml file (see Figure 3 for an example) that specifies list of schemas used to interpret the binary data. The file stores the mission *name*, and it also provides for each type of file, additional information used by the GUI.

Each *schema* element in the file defines:

- The *file*, which is the actual file containing the schema definition
- The *search* element, that stores the list of fields that should appear as option in the find bar
- The *invisible* element that stores the list of fields that should be hidden from the hierarchical representation in the Content Details view.
- The *packet_list_columns* element that stores the list of fields to be displayed as extra columns in the Contents List view.
- The *masks* element specifies the mask(s) used for synchronization detection.
- The *plots* element defines the charts applicable to the data.
- The *transformation* element specifies the data transformation applicable to the data.

A *field* is defined by a name attribute, with the path of the field as value. The path of the field is defined by the concatenation of the several item names in the hierarchical structure defining the binary data.

Only a *field*, defined under the *packet_list_columns*, can have an attribute *indicator* (with value colour). The presence of this indicator is used to select the value during colour coding of the packets.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mission_definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../Mission.xsd">
    <name>Sentinel 1 X-band TM</name>
    <version>1.1</version>
        <schema type="CADU" name="ScrambledCADU"

            [...snip...]

        </schema>
        <schema type="CADU" name="CADU"

            [...snip...]

        </schema>
        <schema type="TF" name="TF"

            [...snip...]

            <transformations>
                <transformation transformationName="TF to ISP" transformationType="TF2ISP"/>
            </transformations>
        </schema>
        <schema type="ISP" name="ISP"
                idle="{false}"
                locationAPID="/Packet_Primary_Header/Packet_Identification/APID"
                locationSSC="/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC"
                locationTimeStamp="/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code">
            <file>Sentinel1X-bandTMISP.xsd</file>
            <search>
                <field name="SSC">/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC</field>
                <field name="APID">/Packet_Primary_Header/Packet_Identification/APID</field>
                <field name="Timestamp">/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code</field>
            </search>
            <invisible/>
            <packet_list_columns>
                <field name="SSC">/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC</field>
                <field name="APID" indicator="color">/Packet_Primary_Header/Packet_Identification/APID</field>
                <field name="Time_Code_Field">/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code</field>
            </packet_list_columns>
            <masks>
                <mask referenceValue="0800C0000000" maskValue="F800C0000000"/>
            </masks>
            <plots>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="SCC" yyPath="./Packet_Primary_Header/Packet_Sequence_Ctrl/SSC" yyType="Number"/>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="APID" yyPath="./Packet_Primary_Header/Packet_Identification/APID" yyType="String"/>
                <plot_path xxName="Timestamp" xxPath="./Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code" xxType="Timestamp"
                           yyName="APID" yyPath="./Packet_Primary_Header/Packet_Identification/APID" yyType="String"/>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="Timestamp" yyPath="./Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code" yyType="Timestamp"/>
            </plots>
        </schema>
    </schemas>
</mission_definition>
```

Figure 3: Example of Mission Definition file

3.2.2. Mission Data Definition Schemas

The Mission Data Definition schemas are XSD schemas adapted (according to [RD.1]) to describe the structure of the binary items inside the data files. Although each schema file could have been defined independently, considering that they can share schema types, the structure shown in Figure 2 has been used for the default missions provided with S2G. Section 3.3 provides some guidelines on how the user can customize an existing mission configuration.

3.3. Customize Mission Data Specification

S2G is a highly customizable application allowing the user to add support for new missions. Apart from the default missions, the tool allows the user to include additional custom missions.

The application will only successfully load the custom mission 'jar' if these structure guidelines are followed:

1. archive each mission definition (Definition XML and Schemas XSD) in a separate 'jar' file (e.g. MyCustomMissionX-BandTM.jar);
2. keep all mission definition files contained in the 'jar' file inside a single folder

The structure of a mission configuration is described in Section 3.2 and includes:

- a set of mission data schemas XSD files that use DFDL to describe the structure of binary data.
- a mission definition XML file, describing the searchable, visible and extra column fields and the schema file used to interpret the related data units;

The proposed approach to create a customized mission is to use an existing working mission and update it to comply with the new mission specification or binary data format. Detailed instructions on how to import/export mission specifications can be found in the S2G User Manual [RD.3]. The following sections present some guidelines of how to customize an existing mission.

3.3.1. Customization of Mission Definition XML file

The mission definition file is shown in Figure 4 with annotations identifying the several items that can be customized.

3.3.1.1. General Indications

While making any customization of the mission definition file, consider the following indications:

- Ensure that the mission has a unique name (tag `<name>`). Since this field is used as internal identifier, the mission might not appear if the mission name is duplicated.
- Ensure that at least three `<schema>` are present, related to types `"CADU"`, `"TF"` and `"ISP"`.
- The `<file>` tag indicates the file containing the binary data schema, to be located in the same directory as the mission definition file.

3.3.1.2. Customize Searchable, Invisible or Extra Columns fields

The tags `<search>`, `<invisible>` and `<packet_list_columns>` are mandatory and contain a list (potentially empty) of fields described by the element with `<field>` tag.

Each field is described by a `<field>` tag with attribute `name` and the path to the field inside the data unit structure as value.

The list of searchable, invisible or extra columns fields can be customized by adding or removing `<field>` elements from the XML file. In order to reload the latest customization, the application must be restarted.

A single field in the `<packet_list_columns>` list can have an attribute `indicator` with value `"color"` to indicate that this field is used to color code the items in the Data Units list view.

3.3.1.3. Customize Masks, Plots and Transformation

The user can specify the mask(s) used for synchronization detection. The <masks> tag contains a list of <mask> elements, each with a reference value and an associated mask (both specified in hexadecimal with the same digits in length). S2G shall consider that the data stream is synchronized if any of the masks in the list are a successful match to the data stream.

The user can also customize the plots and transformation for each schema:

- Plots can be defined with a <plot_path> tag that specifies the values and types to be used for the xx and yy axis by the xx/yyName of the value being plotted, the xx/yyPath to the element where the plotted value should be extracted and the xx/yyType of the value (which can be Number, String or Timestamp).
- Transformation can be defined by adding a new <transformation> tag, specifying its type (ScrambleCADU2CADU, CADU2TF or TF2ISP).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mission_definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../Mission.xsd">
    <name>Sentinel 1 X-band TM</name>
    <version>1.1</version>
        <schema type="CADU" name="ScrambledCADU"

            [...snip...]

        </schema>
        <schema type="CADU" name="CADU"

            [...snip...]

        </schema>
        <schema type="TF" name="TF"

            [...snip...]

            <transformations>
                <transformation transformationName="TF to ISP" transformationType="TF2ISP"/>
            </transformations>
        </schema>
        <schema type="ISP" name="ISP"
                idle="{false}"
                locationAPID="/Packet_Primary_Header/Packet_Identification/APID"
                locationSSC="/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC"
                locationTimeStamp="/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code">
            <file>Sentinel1X-bandTMISP.xsd</file>
            <search>
                <field name="SSC">/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC</field>
                <field name="APID">/Packet_Primary_Header/Packet_Identification/APID</field>
                <field name="Timestamp">/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code</field>
            </search>
            <invisible/>
            <packet_list_columns>
                <field name="SSC">/Packet_Primary_Header/Packet_Sequence_Ctrl/SSC</field>
                <field name="APID" indicator="color">/Packet_Primary_Header/Packet_Identification/APID</field>
                <field name="Time_Code_Field">/Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code</field>
            </packet_list_columns>
            <masks>
                <mask referenceValue="0800C0000000" maskValue="F800C0000000"/>
            </masks>
            <plots>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="SSC" yyPath="./Packet_Primary_Header/Packet_Sequence_Ctrl/SSC" yyType="Number"/>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="APID" yyPath="./Packet_Primary_Header/Packet_Identification/APID" yyType="String"/>
                <plot_path xxName="Timestamp" xxPath="./Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code" xxType="Timestamp"
                           yyName="APID" yyPath="./Packet_Primary_Header/Packet_Identification/APID" yyType="String"/>
                <plot_path xxName="Packet #" xxPath="" xxType="Number"
                           yyName="Timestamp" yyPath="./Packet_Data_Field/(.*)Packet_Secondary_Header/Time_Code_Field/Time_Code" yyType="Timestamp"/>
            </plots>
        </schema>
    </schemas>
</mission_definition>
```

Figure 4: Mission Definition file

3.3.2. Mission Schema Files

The mission schema files are regular XML Schema files (XSD) with extra properties allowing the definition of binary data instead xml data. The properties that enable binary data definition are defined according to the DFDL standard [RD.1].

Figure 5 provides an example of the data schema definition for the higher level structure. The file defines a schema with three top elements:

1. An annotation reporting the version of the schema file
2. An include directive, that indicates that types defined in the indicated file are available to define the data unit;
3. An annotation describing the DFDL format (e.g. encoding and byteOrder) to apply in data parsing;
4. The top level element definition - the example shows the specification of a data unit denominated `"ISP"`, and defined as a sequence of two elements, named `"Packet_Primary_Header"` and `"Packet_Data_Field"`; the types of the two sub-elements (`"TypePacketPrimaryHeader"` and `"TypePacketData"`, respectively) are defined in the detail definition files.
    a. The optional `dmx:representation="Complex"` tag is used when sub-elements include an assertion that depends on the value of a parent element (refer to Annex 1.II **New features**)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:dmx="http://www.deimos.com.pt/dmx/dmx-1.0"
   xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0">

   <xs:annotation>
      <xs:documentation>$Revision: 496 $ $Date:: 2012-03-12 14:08:15#$</xs:documentation>
   </xs:annotation>

   <xs:include schemaLocation="Sentinel3X-bandTMISPTypes.xsd"/>

   <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/dfdl/">
         <dfdl:format byteOrder="bigEndian" bitOrder="mostSignificantBitFirst" alignment="implicit"
         lengthKind="explicit" representation="binary" binaryFloatRep="ieee"
         binaryNumberRep="binary" binaryDecimalVirtualPoint="0" binaryBooleanTrueRep="1"
         binaryBooleanFalseRep="0" sequenceKind="ordered" initiatedContent="no" floating="no"
         choiceLengthKind="implicit" leadingSkip="0" encoding="ISO-8859-1" trailingSkip="0"
         initiator="" terminator="" lengthUnits="bytes" length="0" fillByte="0" separator=""
         escapeSchemeRef="" occursCountKind="fixed"/>
      </xs:appinfo>
   </xs:annotation>

   <xs:element name="ISP" dmx:representation="Complex">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Packet_Primary_Header" type="TypePacketPrimaryHeader"/>
            <xs:element name="Packet_Data_Field" type="TypePacketData"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Figure 5: Mission Schema file (Data Unit definition)

The Figure 6 shows a schema that defines the low level types to be combined in order to define the top level data unit described in the previous paragraph. The file defines a schema with the following elements:

1. An annotation reporting the version of the schema file
2. A list of complex types (`"TypeAPID"`, `"TypePacketPrimaryHeader"`, `"TypeDataFieldHeader_OLCI"`, `"TypeDataHeader"`)

When necessary, complex types can be built upon the definition of other types (e.g. the type `"TypePacketPrimaryHeader"` defines a sequence containing an element of type `"TypeAPID"`); or can be defined over simple types (e.g. the element `"SSC"` is described directly as an `"xs:int"` with additional DFDL properties). More information on the DFDL properties used to define mission schemas is available in [RD.1].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:dmx="http://www.deimos.com.pt/dmx/dmx-1.0"
    xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0">

    <xs:annotation>
        <xs:documentation>$Revision: 470 $ $Date:: 2012-02-23 08:56:15#$</xs:documentation>
    </xs:annotation>

    <xs:complexType name="TypeAPID">
        <xs:sequence>
            <xs:element name="PID"
              type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="7"
              dmx:representation="Binary"/>
            <xs:element name="PCAT"
              type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="4"
              dmx:representation="Binary"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TypePacketPrimaryHeader">
        <xs:sequence>
[...snip...]
            <xs:element name="APID" type="TypeAPID"/>
[...snip...]
            <xs:element name="SSC"
              type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="14"
              dmx:representation="Integer16"/>
            <xs:element name="Packet_Data_Length"
              type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="16"
              dmx:representation="Integer16"/>
        </xs:sequence>
    </xs:complexType>

[...snip...]

    <xs:complexType name="TypeSecondsryHeader_OLCI">
        <xs:sequence>
[...snip...]
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="TypeISPData_OLCI">
        <xs:sequence>
            <xs:element name="data"
              type="xs:byte" dfdl:lengthKind="expression" dfdl:lengthUnits="bytes"
              dfdl:length="{/Packet_Primary_Header/Packet_Data_Length
                            + 1 - contentLength(/Packet_Data_Field/(.*)Packet_Secondary_Header, 'bytes') - 2}"
              dmx:representation="Hexadecimal"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TypeISPData_UNKNOWN">
        <xs:sequence>
            <xs:element name="data"
              type="xs:byte" dfdl:lengthKind="expression" dfdl:lengthUnits="bytes"
              dfdl:length="{/Packet_Primary_Header/Packet_Data_Length + 1}"
              dmx:representation="Hexadecimal"/>
        </xs:sequence>
    </xs:complexType>


[...snip...]


    <xs:complexType name="TypeUserData_OLCI">
        <xs:sequence>
            <xs:element name="ISP_Data" type="TypeISPData_OLCI"/>
            <xs:element name="Packet_Error_Control" type="TypeCRC"/>
        </xs:sequence>
```

```
    </xs:complexType>


    <xs:complexType name="TypeUserData_UNKNOWN">
        <xs:sequence>
            <xs:element name="ISP_Data" type="TypeISPData_UNKNOWN"/>
        </xs:sequence>
    </xs:complexType>



    <xs:complexType name="TypePacketData">
        <xs:sequence>
            <xs:choice>
                <xs:sequence> <!-- Choice for OLCI -->
                    <xs:annotation>
                        <xs:appinfo source="http://www.ogf.org/dfdl/dfdl-1.0">
                            <dfdl:discriminator
                                test="{/PacketPrimaryHeader/APID
                                      in [1056,1057,1072,1073,1088,1089,1104,1105,1120,1121]}"/>
                        </xs:appinfo>
                    </xs:annotation>
                    <xs:element name="OLCI_Packet_Secondary_Header" type="TypeSecondaryHeader_OLCI"/>
                    <xs:element name="OLCI_User_Data_Field" type="TypeUserData_OLCI"/>
                </xs:sequence>
        [...snip...]
                <xs:sequence> <!-- Choice for UNKNOWN -->
                    <xs:annotation>
                        <xs:appinfo source="http://www.ogf.org/dfdl/dfdl-1.0">
                            <dfdl:discriminator
                                test="{fn:true()}"/>
                        </xs:appinfo>
                    </xs:annotation>
                    <xs:element name="UNKNOWN_User_Data_Field" type="TypeUserData_UNKNOWN"/>
                </xs:sequence>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Figure 6: Mission Schema file (Data Unit types definition)

3.3.3. Customizable Schemas Example

A customizable mission schema jar file (CCSDS-bandTM.jar) can be found inside the S2G installation folder (workspace\resources\data).

The content of the jar file is:

- Mission Definition (see section 3.2.1)
    - CCSDSX-BandTM.xml
- Mission Schema Files (see section 3.2.2)
    - CCSDSX-BandTMCADU.xsd
    - CCSDSX-BandTMCADUScrambled.xsd
    - CCSDSX-BandTMCADUTypes.xsd
    - CCSDSX-BandTMTF.xsd
    - CCSDSX-BandTMTFTypes.xsd
    - CCSDSX-BandTMISP.xsd
    - CCSDSX-BandTMISPData.xsd
    - CCSDSX-BandTMISPTypes.xsd

- Default CCSDSTime schema
    - CCSDSTime.xsd

Customization points can be found inside mission definition and mission schema files listed above. An example of a customization point is provided in Figure 7.

```
<!--
      ***CUSTOMIZATION POINT***

      $SATELITE_NAME$
            - Enter the satellite mission name
                  (For this example will use CCSDS as $SATELITE_NAME$)
      $TYPE_SCHEMA$
            - Enter the shema type
      $TYPE_BAND$
            - Enter the band type (S or X or Ka)

      Example:
            $SATELITE_NAME$     = CCSDS
            $TYPE_SCHEMA$       = CADU
            $TYPE_BAND$         = X

            Result: CCSDSX-BandTMCADU.xsd
-->
```

Figure 7: Example of customization point in mission definition and mission schema files.

Relevant information about the CADU/TF length-related fields is presented in Figure 8. This information can facilitate schemas customization.

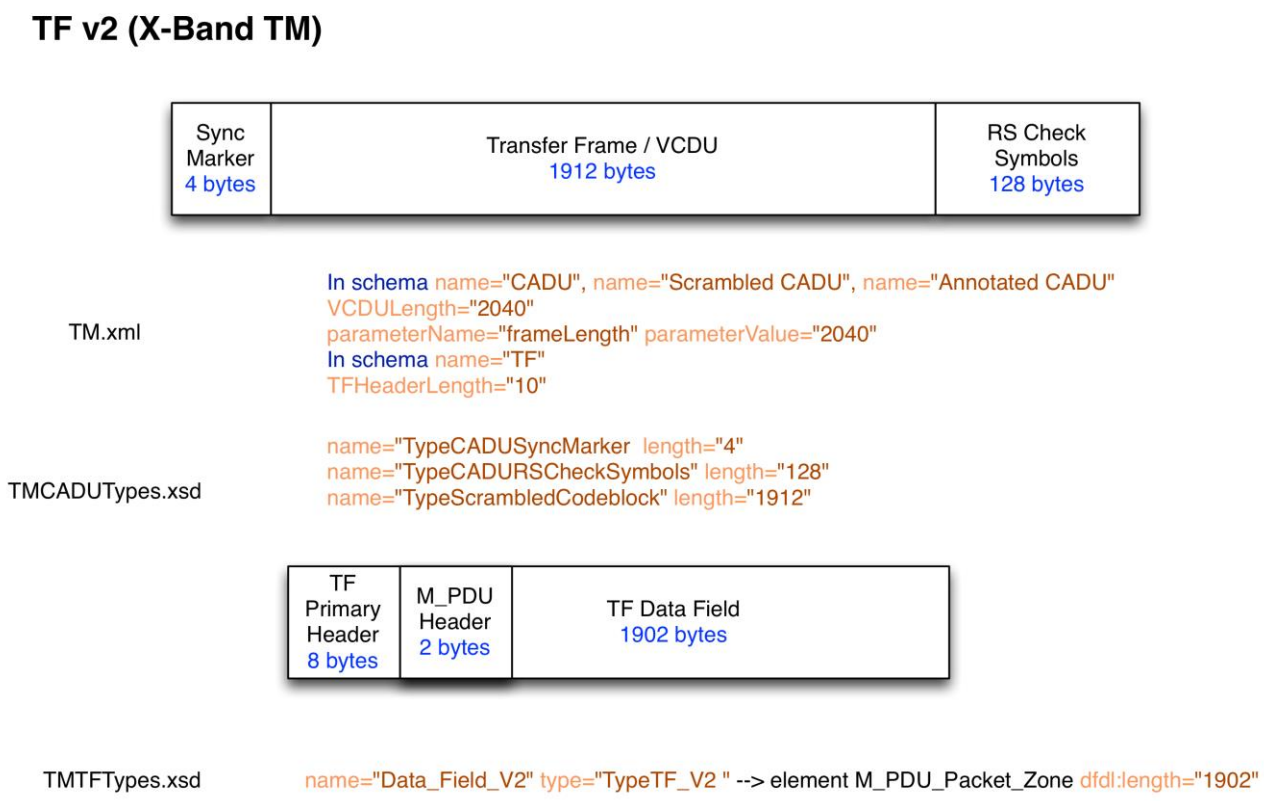


Figure 8: Example of CADU/TF length-related fields.

ANNEX 1 – PORTING GUIDE FROM V2.0.X TO V2.2.X

I. Features to upgrade

Time customization

The time customization has improved, allowing defining a customized reference epoch. Up till v2.0.X there was in S2G no clear separation in the definition of epoch from time representation in the mission specification schema. Since v2.2 there is now defined a container of time types which includes and clearly separates both the epoch and the representation. This container (CCSDSTimes.xsd) has a definition of a set of time types which can be integrated elegantly in the existing schemas.

Important: using schemas from v2.0.X in S2G v2.2 without upgrading to use the new time definitions shall imply that the data handling shall be unable to parse dates and times. Instead a raw hexadecimal value shall be presented were a time would be expected.

Refer to [RD.1] for the definition of times in S2G.

"references" tag added to the mission configuration file

A "references" tag was added to the mission configuration file with <filename> and <date> sub-tags referring to the applicable spreadsheet. For custom mission it is recommended to include the documentation describing the data structure.

Important: using schemas from v2.0.X in S2G v2.2 without including the new "references" tag will prevent S2G from reading the mission specification properly. Even if no information is available to include as reference the tags must be included, even if left with no value.

II. New features

Assert depending on the value of a parent element

It is possible to perform an assertion on a given field that depends on the value of a parent element. Accessing parent fields is achieved using '..' and '.' operators for moving up in the DFDL hierarchy. As in any DFDL expression tests can be combined with logical operators.

An example in an assertion on the Secondary Header Flag field for Sentinel-1 ISP data since this field should only exist for TF version 1.

```
   <xs:complexType name="TypePacketIdentification">
      <xs:sequence>
[...snip...]
         <xs:element name="Secondary_Header_Flag" type="xs:int" dfdl:lengthKind="explicit"
            dfdl:lengthUnits="bits" dfdl:length="1" dmx:representation="Binary"
            dmx:assertExpression="../../.. eq / or
../../../../../../Primary_Header/Master_Channel_Id/Transfer_Frame_Version eq 1"
            dmx:assertMessage="Invalid secondary header flag detected."/>
         <xs:element name="APID" type="TypeAPID" dmx:representation="Hexadecimal"/>
      </xs:sequence>
   </xs:complexType>
```

The assert expression is defined so that if TF fields are not visible the check is ignored:

- we test TF version when TF exists;

- we test ISP is root (/) when TF does not exist - three levels up should be root.

Note that for the above test, all elements represented by '/' - i.e. ISP, Transfer Frame and CADU - MUST have the `dmx:representation="Complex"` tag added to the top level element (see Figure 5).

Support for multiple definitions of the same data type

Mission definition now includes support for multiple definitions of the same data type. An example of application of this capability is the handling of the same APID for different data (e.g. to support Sentinel-2).

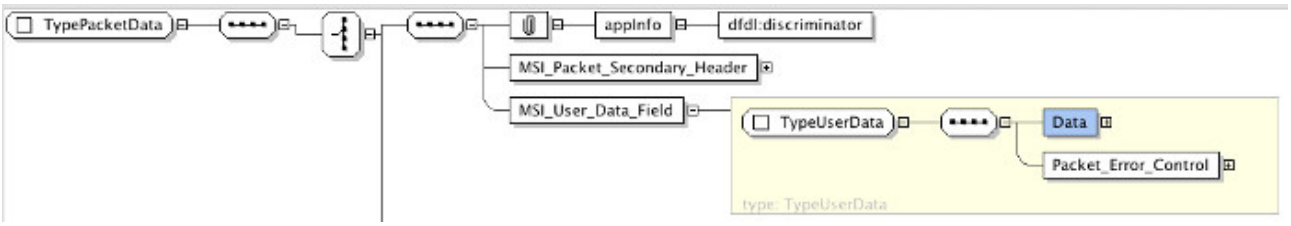
```
<xs:complexType name="TypePacketData">
   <xs:sequence>
      <xs:choice>
         <!-- Choice for MSI -->

         <xs:sequence
            dmx:switchValue="MSI"
            dmx:switchTest="{/Packet_Primary_Header/Packet_Identification/APID in [0]}">
            <!-- APID 0 here and not with Time Packet (9,2) -->
            <xs:annotation>
               <xs:appinfo source="http://www.ogf.org/dfdl/dfdl-1.0/">
                  <dfdl:discriminator
                     test="{/Packet_Primary_Header/Packet_Identification/APID inrange [  0, 12]
[...snip...]
         </xs:sequence>
[...snip...]

         <!-- Choice for TM_Time_Packet (9,2) -->
         <xs:sequence
            dmx:switchValue="TMT92"
            dmx:switchTest="{/Packet_Primary_Header/Packet_Identification/APID in [0]}">
            <!-- Overlaps MSI APID 0. Removed here (moved to 99999) as it is SCIENCE config -->
               <xs:annotation>
                  <xs:appinfo source="http://www.ogf.org/dfdl/dfdl-1.0/">
                     <dfdl:discriminator
                        test="{/Packet_Primary_Header/Packet_Identification/APID in [99999]}" />
                  </xs:appinfo>
               </xs:annotation>
               <xs:element name="TM_Time_Packet_Secondary_Header" type="TypePacketData_TMT92" />
               <xs:element name="TM_Time_Packet_User_Data_Field" type="TypeUserData_TMT92" />
            </xs:sequence>
[...snip...]
         </xs:choice>
      </xs:sequence>
   </xs:complexType>
```

Multiple definitions for the same type are identified in a choice construct by indicating the pair of tags `dmx:switchValue` and `dmx:switchTest`. `dmx:switchTest` is a DFDL expression to be applied for identifying detecting a multiple definition. When the test of more than one choice is valid this identifies multiple possible choices for describing a given type. When this occurs the user is presented in the GUI with the option to select among which to use for applying to the data. The values in `dmx:switchValue` are used as identifiers (and should be unique) for labelling each alternative.

New level of ISP data type definitions

A new level of ISP data type definitions to be included by the ISP Type definition has been introduced. The ISP schema was extended with an ISPDataType level. An additional node is introduced at the level of the User Data:

- V2.0.X



- V2.2

Even though this update is not mandatory for custom mission definitions it is recommended to introduce this extra definition to support ISP extensions without having to modify the 'official' ISP schemas.

ANNEX 2 – PORTING GUIDE FROM V2.2.X TO V2.3.X

Some upgrades were made, both in S2G and official mission definitions, in order to increase compliance with the DFDL specification. Some of the S2G upgrades imply additional validations that can prevent the loading of existing mission definitions. The remaining S2G upgrades only increase the DFDL compliance of those definitions.

I. Features to upgrade

1. Types and length correspondence

According to DFDL specification, *element* and *simple type*'s definitions cannot have a length exceeding the capacity of its type (the capacities of simple types can be found in Table 22 of [RD.4]). The validation of such condition was included in S2G. Therefore existing schemas not compliant with it have to be changed in order to be accepted.

In the official mission definitions, for example, there were some elements with type `xs:byte` and `xs:int` with lengths greater than 1 and 4 bytes, respectively. To obtain the compliance with the described condition the following changes were made:

- types of *elements* and *simple type* definitions with type `xs:byte` and length greater than 8 bits were changed to `xs:hexBinary`;
- types of *elements* and *simple type* definitions with type xs:int and length greater than 32 bits were changed to `xs:hexBinary` or `xs:integer`, depending on their content – `xs:hexBinary` was applied when the representation was `Hexadecimal` or `Binary`.

2. Relative path expressions

In DFDL, the context for the evaluation of a relative path expression is the info item corresponding to the component containing the expression. However, in previous versions of S2G, the test expressions of choice branch's discriminator annotations were evaluated at the info item corresponding to the nearest *simple type* or *element* where it was contained, and had such item as context. To improve the compliance with DFDL specification, those expression's evaluations were changed to apply the standard context. Therefore, existing schemas should be changed accordingly: every choice branch's discriminator test `./SUBPATH` or `../SUBPATH` should be replaced by `ROOT/SUBPATH` or `ROOT/../SUBPATH`, respectively, where `SUBPATH` is an arbitrary path expression and `ROOT` is the expression composed by a `..` step for each component containing (directly or indirectly) the expression and contained in the *simple type* or *element* nearest to it.

For example, in the file `Sentinel1X-bandISPType.xsd` of the official mission definitions, in the following *choice* definition, the first *discriminator*'s test expression was changed from `{./SSB_Flag in [0]}` to `{../SSB_Flag in [0]}`.

```
<xs:choice>
   <xs:element name="SAS_SSB_Message_Imaging_Noise" type="TypeSAS_SSBMessage_IM">
      <xs:annotation>
         <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator test="{../SSB_Flag in [0]}"/>
         </xs:appinfo>
      </xs:annotation>
   </xs:element>
   <xs:element name="SAS_SSB_Message_Calibration" type="TypeSAS_SSBMessage_CAL">
      <xs:annotation>
         <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator test="{../SSB_Flag in [1]}"/>
         </xs:appinfo>
      </xs:annotation>
   </xs:element>
</xs:choice>
```

3. contentLength function

The previous versions of S2G (up till v2.2) supported a function `length($node)` in expressions, which had a path expression as argument and were evaluated to the length, in bytes, of the info item corresponding to that path. However, in DFDL, there is a standard function `contentLength($node, $lengthUnits)`, where `$node` and `$lengthUnits` are arguments expecting a path expression and a unit, bit or byte, respectively, which also evaluates to the length of the item corresponding to the path argument, but in the specified units. Since version v2.3 of S2G the `length($node)` function is no more supported, and the standard one `contentLength($node, $lengthUnits)` is supported instead. Therefore, in existing schemas, in every expression using the `length` function, `length(PATH)`, where `PATH` is an arbitrary path expression, should be replaced by `dfdl:lengthContent(PATH,'bytes')`.

4. DFDL properties validation

Since v2.3 S2G performs a validation on DFDL properties occurring in schemas, to ensure that they have a valid name, according to DFDL specification, and a valid and supported value (note that S2G does not support all features of DFDL). Therefore, existing mission definitions (with invalid or unsupported properties names or values) correctly operating in previous versions, can now be prevented of being loaded in the new version of S2G. In such schemas DFDL properties values and names should be corrected according to Table 18 of [RD.5].

ANNEX 3 – PORTING GUIDE FROM V2.4.X TO V2.5.X

From S2G version 2.5.X onwards, it is implemented the ability to parse the new interpretation method for intersection and range checks when detecting and identifying data fields in the mission schema specifications.

Features to upgrade

Below, it is shown examples for 'in' and 'inrange' of replacing the previous syntax parsed by the S2G version 2.4.5 along with the newest syntax valid from version 2.5.X onwards.

1. Operator 'in' replacement with 'intersect' operator along with fn:exists function

'in' operator →	'intersect' operator
V2.4.5	`<dfdl:discriminator test="{/Packet_Primary_Header/Packet_Identification/APID in [1164, 1165]}"/>`
V2.5.0	`<dfdl:discriminator test="{fn:exists(/Packet_Primary_Header/Packet_Identification/APID intersect (1164, 1165))}"/>`

2. Operator 'inrange' replacement with 'intersect' operator or 'dfdl:checkRangeInclusive' function

Unlike 'in' operator, the 'inrange' operator can be translated in two ways according with the range check order of magnitude. If the range is small, it shall be used the intersect operator along with fn:exists function. Otherwise, it shall be used the checkRangeInclusive dfdl function.

'inrange' operator, small range →	'intersect' operator
V2.4.5	`<xs:element name="PCAT" type="xs:unsignedByte" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="4" dmx:representation="Binary" dmx:assertExpression="../PCAT inrange [0, 15]" dmx:assertMessage="Invalid PCAT."/>`
V2.5.0	`<xs:element name="PCAT" type="xs:unsignedByte" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="4" dmx:representation="Binary" dmx:assertExpression="fn:exists(../PCAT intersect (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15))" dmx:assertMessage="Invalid PCAT."/>`

The checkRangeInclusive function only accepts 'int', 'double', 'float' and 'TypeAPID' data types. If not one of these types, use intersect method.

'inrange' operator, large range → dfdl:checkRangeInclusive function	
V2.4.5	`/Packet_Primary_Header/Packet_Identification/APID inrange [64,543]`
V2.5.0	`dfdl:checkRangeInclusive(/Packet_Primary_Header/Packet_Identification/APID,64,543)`

End of Document