


# ***DFDL4S library***

## **Developer's Manual**

**Code :** S2G-DME-TEC-SUM078  
**Issue :** 1.K  
**Date :** 31/08/2020

	<b>Name</b>	<b>Function</b>	<b>Signature</b>
<b>Prepared by</b>	Sergio Saraiva	Project Manager	
<b>Reviewed by</b>	Sergio Saraiva	Project Manager	
<b>Approved by</b>	Antonio Gutiérrez	Technical Consultant	
<b>Signatures and approvals on original</b>			

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 <input type="checkbox"/>	Restricted <input checked="" type="checkbox"/>	Confidential <input type="checkbox"/>

External Distribution		
Name	Organisation	Copies
Free for distribution	-	-

Archiving	
Word Processor:	MS Word 2000
File Name:	S2G-DME-TEC-SUM078-1K_MB.docx

## Document Change Log

Issue	Change description	Date	Pages Affected
1.A	First issue of the document.	05/12/2014	All
1.B	Updated according to ESA review comments.	15/01/2015	§2.2, §3.1.1, §4.2 and Appendix A
1.C	Documented the differences between DFDL properties and DMX properties.	06/03/2015	§3.1.1
1.D	Updated to reflect: (a) added API for DFDL creation capability; (b) re-organization of the deployment package.	11/11/2015	§3.3, §4.2.2, §4.3, §4.4
1.E	Chapter 6 added.  Chapter 7 added. (DFDL4S-AN-176a8 - discriminator expressions described; DFDL4S-AN-176a10 - in and inrange described; DFDL4S-AN-176a11 - dfdl:contentLength path argument described).  Appendix A improved and updated. (DFDL4S-AN-176a3 - supported DFDL types mentioned; DFDL4S-AN-176a6 and DFDL4S-AN-176a7 - supported values, including for dfdl:encoding, added)  References updated.	15/04/2016	Chapter 6, Chapter 7,  Appendix A,  Table 2
1.F	Chapter 4 updated:  <ol style="list-style-type: none"> <li>1. Changed Figure 4-3, 4-4 and 4-5 to Figure 3, 4 and 5</li> <li>2. Updated Figure 3 with updated diagram</li> <li>3. Reordered Table 5 DFDLLib methods by name</li> <li>4. Updated contents of Table 5 DFDLLib: <ol style="list-style-type: none"> <li>a. Changed inputs names: appendElements(1), appendElements(2), initLib,</li> <li>b. Changed description: createDocument(1)</li> <li>c. Added missing method name, input, ouput and description: createDocument(2), getSchemaDefinition, interpretDocument(2), main</li> </ol> </li> <li>5. Updated Figure 4 with updated diagram</li> <li>6. Updated contents of Table 6 Document: <ol style="list-style-type: none"> <li>a. Added missing method name, input, ouput and description: childAdd, evaluate</li> <li>b. Changed inputs names: childAt(1), childAt(2), childAtOffset, childCount, elementContains</li> </ol> </li> </ol>	19/09/2017	Chapter 4  Figure 3 Table 4 Table 4  Figure 4 Table 5

	<ul style="list-style-type: none"> <li>c. Corrected input, output and description: close</li> <li>d. Removed method: getElement</li> </ul> <p>7. Split contents of Table 6 into logical groups and by updated its operations:</p> <ul style="list-style-type: none"> <li>a. Added new methods: getHexadecimalValue, getIntrinsicType, getRangeMaximum, getRangeMinimum, getValueBinary, getValueChar8, getValueFloat32, getValueFloat64, getValueInteger, getValueTime, setData(1), setData(2), setRawData(1), setRawData(2), setValueFloat32, setValueFloat64, setValueInteger, setValueTime, value(3)</li> <li>b. Changed inputs names: absoluteName, name, offset, path, retrieveCleanAlignedData, retrieveCleanData, retrieveRawData(1), retrieveRawData(2), size, type, value(1)</li> </ul> <p>8. Updated contents of chapter 4.4 by replacing the example code with a new use case of creating a file based on a schema and setting fields.</p>		<p style="text-align: center;">Table 6</p> <p style="text-align: center;">Chapter 4.4</p>
1.G	<p>Removed text, figures and tables not related with the supported DFDL4S API.</p> <p>Removed from tables the methods that do not belong to the DFDL4S API.</p>	30/03/2018	<p>Tables: 9-20 and 22</p> <p>Figures: 5</p> <p>Sections: 4.2.2.3 4.2.3 4.3.1.3</p> <p>Tables 4-9</p>
1.H	<p>Updated text, tables and diagrams of DFDL4S API</p> <p>Added Appendix B with build instructions for DFDL4S</p>	30/08/2018	<p>Section 4</p> <p>Appendix B</p>
1.I	<p>Added new Section 6.2</p> <p>Section 6.2 - Representation types after DFDL4S v1.5.2</p>	17/10/2018	Section 6.2

	<p>Added new Section 6.3 Section 6.3 - Time Code Representations</p> <p>Updated Section 7 - Expression Language Added <code>dfdl:checkRangeInclusive(\$node,\$val1,\$val2)</code>. Added intersect operator Added <code>dfdl:defineFormat</code>.</p> <p>Updated Section Appendix B Added location for unit tests report</p>		<p>Section 6.3</p> <p>Section 7</p> <p>Appendix B</p>
1.J	<p>Replace all instances of <code>lengthKind="expression"</code> with <code>lengthKind="explicit"</code></p> <p>Multiplication operator ( * ) in DFDL4S expression language</p> <p>Added new methods in DFDLLib class:</p> <ul style="list-style-type: none"> <li>• <code>createNewDocument</code></li> <li>• <code>storeDocument (1)</code></li> <li>• <code>storeDocument (2)</code></li> </ul> <p>Added new methods in Document class:</p> <ul style="list-style-type: none"> <li>• <code>createElement (1)</code></li> <li>• <code>createElement (2)</code></li> <li>• <code>createElementTime</code></li> <li>• <code>createElementInteger</code></li> <li>• <code>createElementFloat32</code></li> <li>• <code>createElementFloat64</code></li> <li>• <code>createElementBytes</code></li> <li>• <code>validate</code></li> </ul> <p>Replace the example to create new document with DFDL lib</p> <p>Add <code>CCSDTimeTrait</code> utility</p>	17/02/2020	Figure 8
1.K	<p>Update Figure 3 and Table 5 with the new methods in DFDLLib class</p> <p>Update Figure 4 and tables Table 6, Table 7, Table 8, Table 9 and Table 10 with the new classes and methods that belong to the Domain Entities of DFLD4S. Table 15</p> <p>Update Figure 5 and tables Table 11, Table 13, Table 14, <b>Erro! A origem da referência não foi encontrada.</b> and Table 15 with the new classes and methods that belong to the Utilities of DFLD4S. Table 15</p>	31/08/2020	<p>Section 4.2.1</p> <p>Section 4.2.2</p> <p>Section 4.2.3</p>

Added new methods in DFDLLib class:

- interpretDocument(String, byte[], InterpreterMonitor)
- interpretDocument(String, byte[], byte[][], byte[][], HashMap<DataSize,DataSize>, String, InterpreterMonitor)
- interpretDocument(String, String, InterpreterMonitor)
- interpretDocument(String, String, byte[][], byte[][], HashMap<DataSize,DataSize>, String, InterpreterMonitor )
- storeDocument(Document, String, int, int)

### DOMAIN ENTITIES

Added new methods in Document class:

- size()
- getCleanAlignedData(DataSize, DataSize)
- getRawData(DataSize, DataSize)
- childAt(int, String, String)
- getSyncHistory()

Create enum REPRESENTATION\_TYPE in Element class.

Added new methods in Element class:

- parent()
- root()
- childCount()
- propertyValueGet(String)
- size()
- sizeAvailable()
- offset()
- retrieveRawData()
- retrieveRawData(int, int)
- getValueAsRepresentation()
- getValueAsRepresentation(REPRESENTATION\_TYPE)
- getValueBinary() - old getValueAsString()
- evaluate(String) - old ElementFinder.getElement(Element, String)
- evaluateBoolean(String)
- evaluateInteger(String)

Removed methods in Element class:

- getChildErrors()
- hasError()
- hasSevereError()

Added new methods in ErrorIndicator class:

- hasErro() - old errorStatus()
- isPropagateabel()
- isSevere()
- isExpressionMalformed
- getPathElementNotFound()
- hasNonSevereError()

#### UTILITIES

Removed ElementFinder and BinaryBuffer class

Added new class DataSize and ReportEvent

Added new interfaces Report and InterpreterMonitor

Added section 6.1.17 to describe the way partial signed integers are processed by the DFDL.



## Table of Contents

<b>1. Introduction</b>	<b>13</b>
<b>1.1. Purpose</b>	<b>13</b>
<b>1.2. Scope</b>	<b>13</b>
<b>1.3. Acronyms and Abbreviations</b>	<b>13</b>
<b>2. Related Documents</b>	<b>15</b>
<b>2.1. Applicable Documents</b>	<b>15</b>
<b>2.2. Reference Documents</b>	<b>15</b>
<b>3. Getting Started</b>	<b>17</b>
<b>3.1. Introduction</b>	<b>17</b>
3.1.1. DFDL Grammar	17
<b>3.2. Initial Requirements</b>	<b>18</b>
<b>3.3. Installation</b>	<b>18</b>
<b>4. DFDL4S library</b>	<b>19</b>
<b>4.1. Architectural Overview</b>	<b>19</b>
<b>4.2. DFDL4S library API</b>	<b>20</b>
4.2.1. DFDLLib	20
4.2.2. Domain Entities	22
4.2.2.1. Document class	24
4.2.2.2. <i>Element</i> class	25
4.2.2.3. ErrorIndicator	28
4.2.3. Utilities	29
4.2.3.1. DataSize	29
4.2.3.2. CCSDSTimeTrait	30
4.2.3.3. Report	31
4.2.3.4. ReportEvent	32
4.2.3.5. InterpreterMonitor	33
4.2.4. Exceptions	33
4.2.4.1. ErrorLoadingException	33
<b>4.3. Process logic</b>	<b>33</b>
4.3.1. Java Programming Language	33
4.3.1.1. DFDLLibrary	33
4.3.1.2. Domain Entities	34
<b>4.4. Example of use</b>	<b>34</b>

---

4.4.1. Java Programming Language	34
4.4.1.1. Java Build and Execution process	35
<b>5. Mission Configuration</b>	<b>37</b>
<b>5.1. Mission Data Definition Schemas</b>	<b>37</b>
5.1.1. Mission Schema Files	37
<b>6. Representation Types</b>	<b>40</b>
<b>6.1. Basic Types Representations</b>	<b>40</b>
6.1.1. Character8	40
6.1.2. Character16	41
6.1.3. String	41
6.1.4. Integer8	41
6.1.5. UInteger8	41
6.1.6. Integer16	41
6.1.7. UInteger16	41
6.1.8. Integer32	41
6.1.9. UInteger32	42
6.1.10. Integer64	42
6.1.11. UInteger64	42
6.1.12. Float16	42
6.1.13. Float32	42
6.1.14. Float64	42
6.1.15. Binary	42
6.1.16. Hexacimal	42
6.1.17. Fragmented Signed Integers	43
<b>6.2. Representation types after DFDL4S v1.5.2</b>	<b>43</b>
<b>6.3. Time Code Representations</b>	<b>43</b>
6.3.1. CUCTime	44
6.3.2. CUCSecondsNanosecondsTime	45
6.3.3. CDSTime	45
6.3.4. CCSTime	46
6.3.5. CCSASCITime	47
6.3.6. AgencyDefinedTime	47
<b>7. Expression language</b>	<b>49</b>
<b>Appendix A – Compatibility with DFDL Core Set</b>	<b>55</b>

---

## List of Tables

Table 1: Applicable documents .....	15
Table 2: Reference documents .....	15
Table 3: Installation Archives.....	18
Table 4: Minimum System Requirements .....	18
Table 5: List of operations of the DFDLLib class .....	20
Table 6: List of operations of the Document class .....	24
Table 7: Element class: List of methods for getting the value of an element .....	26
Table 8: Element class: List of methods for setting the value of an element .....	27
Table 9: Element class: List of methods for getting properties of an element .....	27
Table 10: List of operations of the ErrorIndicator class .....	29
Table 11: List of operations of the DataSize class .....	30
Table 12: List of operations of the CCSDSTimeTrait class .....	30
Table 13: List of operations of the Report interface .....	32
Table 14: List of operations of the ReportEvent class.....	32
Table 15: List of operations of the InterpreterMonitor class .....	33
Table 17: Description of epoch property to support time definitions .....	44
Table 18 Allowable Specified Lengths in Bits for Base-2 Binary Number Elements .....	55
Table 19 DFDL core properties compatibility .....	56

## List of Figure

Figure 1: Hierarchy of Data received by the Ground Sensor Stations.....	17
Figure 2: DFIDL4S library high-level package diagram.....	19
Figure 3: DFDLLib class diagram.....	20
Figure 4: Domain Entities package class diagram.....	23
Figure 5: Utilities package diagram.....	29
Figure 6: Mission Configuration files structure .....	37
Figure 7: Mission Schema file (Data Unit definition).....	38
Figure 8: Mission Schema file (Data Unit types definition).....	39
Figure 9: Example of representation type usage.....	40
Figure 10: CUC Time Code example .....	44
Figure 11: CUCSecondsNanoseconds Time Code example .....	45

Figure 12: DFDL4S Expression Language ..... 50  
Figure 13: DFDL Simple Types ..... 55

## 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, update and create CADU, TF or ISP data files.

### 1.1. Purpose

The objective of this manual is to provide an operation manual of the use of DFDL4S library to read, parse, inspect, update or create files storing CADUs, TFs and ISPs.

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

This document shows a detailed description of the DFDL4S library and an API that should be used as a reference manual by model developers. It also includes a brief architecture description and some examples of use.

The following sections of this document are organized as follows:

- Section 2 lists applicable and reference documents
- Section 3 provides instructions to install and deploy the library
- Section 4 provides a description of the library architecture, the process logic and some examples of use. It also includes the coding guidelines.

### 1.3. Acronyms and Abbreviations

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

Acronym	Description
CADU	Channel Access Data Unit
DFDL4S	DFDL for Space
ISP	Instrument Source Packet
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-ECP120	S2G Data Viewer: Proposal for CCN9 Activities	1.B

### 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]	GFD.207	Data Format Description Language (DFDL) v1.0	1.0
[RD.2]	ECSS E-70-41	Ground systems and operations – Telemetry & telecommand packet utilisation	
[RD.3]	REC-xml20081126	Extensible Markup Language (XML) 1.0 (Fifth Edition)	1.0
[RD.4]	REC-xpath20-20101214	XML Path Language (XPath) 2.0 (Second Edition)	2.0

This page intentionally left blank



## 3. GETTING STARTED

### 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.

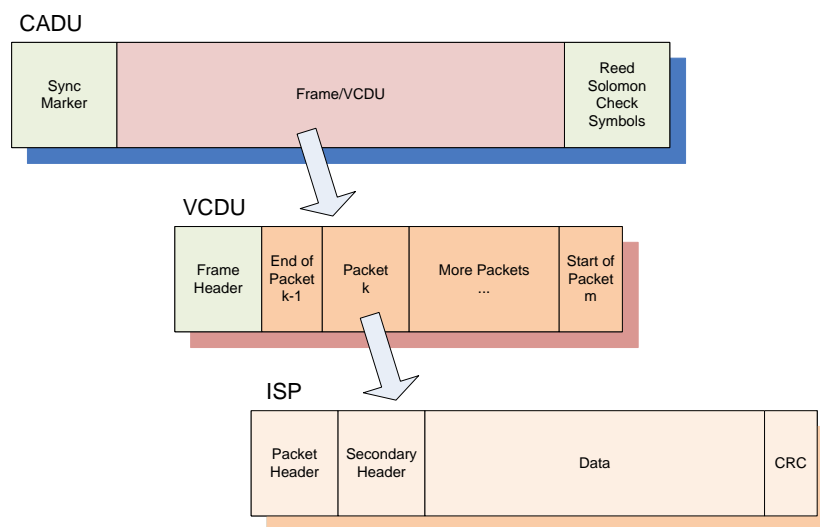


Figure 1: Hierarchy of Data received by the Ground Sensor Stations

The DFDL4S library interprets the contents of the communication channels between the signal-in-space and the ground systems apparatus. It interprets files containing concatenated CADUs, TFs or ISPs, and lists of available data units and allows reading the fields and associated values inside each data unit. The library also supports the update (write) of the values in each data unit.

This document uses the designation of *data unit* when the type of the data item (CADUs, TFs or ISPs) is not relevant for the context.

#### 3.1.1. DFDL Grammar

DFDL4S is a generic binary data binding library based on the Data Format Description Language [RD.1]. An extension of the original specification has been introduced in the scope of S2G DataViewer development. These extensions are necessary to cope with S2G specific requirements.

Data Binding language based on the DFDL makes use of the standard “dfdl:” properties as tags in the xml definition. The compliance of “dfdl:” properties used by DFDL4S to the DFDL specification is detailed in Appendix A – Compatibility with DFDL Core Set. On the other hand the extension to the original specification introduces the use of “dmx:” properties. DMX attributes are not processed by DFDL4S library and are mentioned here for information only since they are part of the schemas used to process space-to-ground-data. It should be noted that in the case when an external application makes use of the DFDL4S library an advanced model developer should be aware of the existence of both types of properties to properly interpret data files.

## 3.2. Initial Requirements

DFDL4S is a Java library therefore it is available for several platforms. The installation should consider the minimum requirements presented in Table 4. The platforms presented have been used to support testing activities.

**Table 3: Installation Archives**

Archive	Supported Platform
dfdl4s-X.Y.Z-bin.zip	Linux (64 bit) macOS (64 bit) Windows (64 bit)

**Table 4: Minimum System Requirements**

Platform	Requirements
Linux (64 bit)	RAM: 2 GB Disk Space: 50 MB Dependencies: Java 1.8
macOS (64 bit)	RAM: 2 GB Disk Space: 50 MB Dependencies: Java 1.8
Windows (64 bit)	RAM: 1 GB Disk Space: 50 MB Dependencies: Java 1.8

## 3.3. Installation

To install DFDL4S library simply unzip the distribution archive (see Table 3) into the installation directory. The folder structure resultant of this action is as follows:

- DFDL4S: main folder containing the LICENSE and README files;
- DFDL4S/docs: Doxygen generated documentation of the library API in html format;
- DFDL4S/examples: Ready-to-use example including a standalone Java program (which source code can be adapted) and a script showing how to compile and execute the code;
- DFDL4S/lib: dfdl4s.jar (the DFDL4S library itself) + external libraries used by DFDL4S;

The mission configuration files are described in section 5. The configuration file is an XML file that provides information required by the library to interpret the data. The definition of the mission binary data, namely the data fields for CADU, TF and ISPs, is defined using DFDL [RD.1].

## 4. DFDL4S LIBRARY

In this section, the following is presented:

An architectural overview, giving structural descriptions of the elements offered in the APIs (such as inheritance diagrams for classes).

A complete set of examples of how to use the APIs and how to include them in model implementation.

### 4.1. Architectural Overview

The DFDL4S library provides capabilities for parsing files based on DFDL definitions. It is a Java library (packaged as a simple to use .jar file). The library provides developers with a set of routines with a well-defined public interface hiding the implementation details. The library interface enables a set of data manipulation operations based on DFDL schemas used to interpret binary data<sup>1</sup>. The operations foreseen include: loading binary data into a DFDL tree structure, navigate/inspect thru a DFDL tree, read a DFDL tree node value and update a DFDL tree node value (writing it to the underlying file support).

DFDL4S library is decomposed in the following conceptual packages (as depicted in Figure 2):

**DFDLLib**: main entry point for parsing a binary file;

**Domain Entities**: a set of classes mapping the binary file into structures enabling traversing and accessing the binary data;

**Utilities**: a set of utility classes;

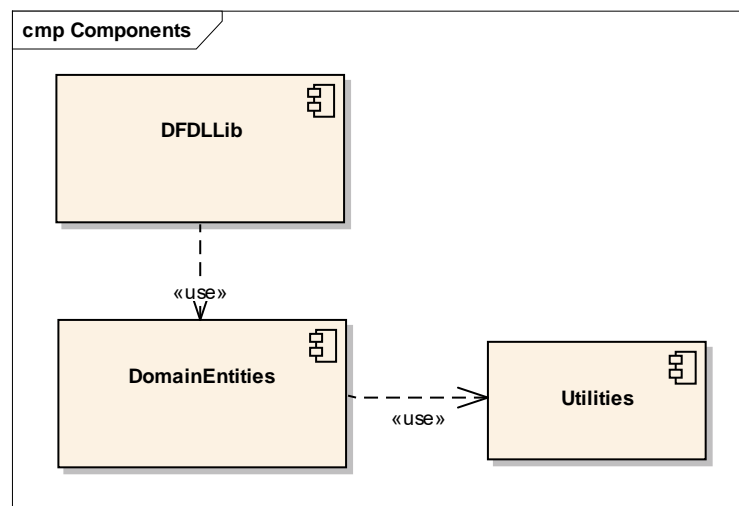


Figure 2: DFDL4S library high-level package diagram

<sup>1</sup> DFDL also supports text data, but due to the intended use of DFDL4S that support has not been considered necessary and is not covered by the current implementation.

## 4.2. DFDL4S library API

The following sections describe the API provided by DFDL4S library. It should be noted that besides this manual a model developer can also refer to the Doxygen generated documentation of the library API in html format provided in the library deployment package (refer to section 3.3 for installation details).

### 4.2.1. DFDLLib

The DFDLLib class provides the capability to interpret the contents of a binary file according to the specifications of a schema.

Figure 3 shows the DFDLLib class diagram, listing interface methods.

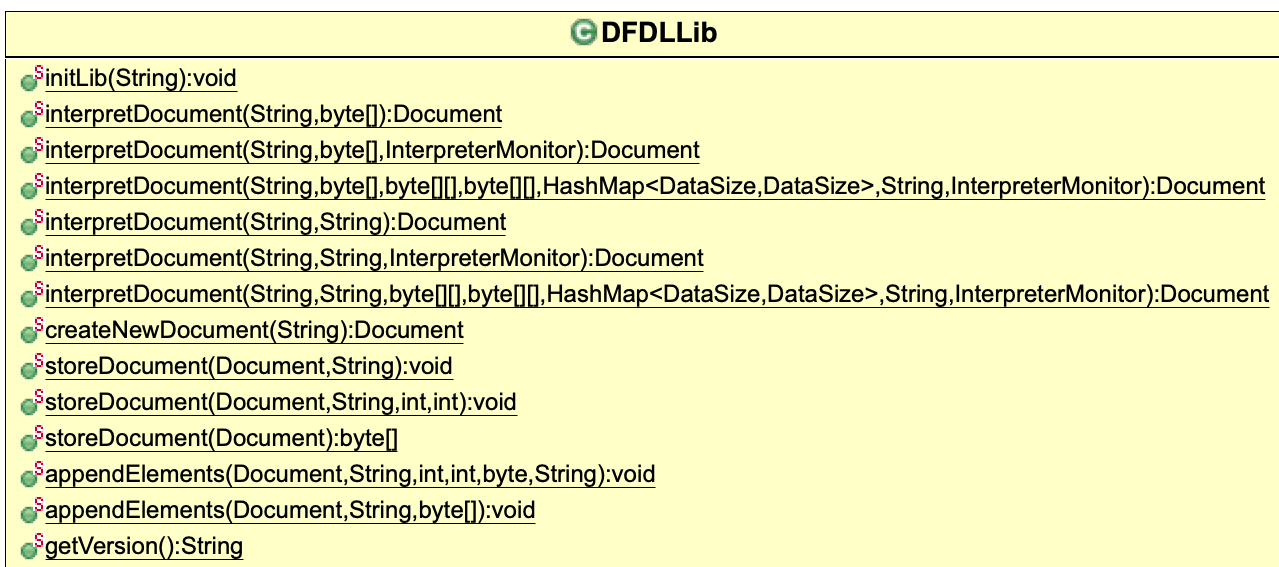


Figure 3: DFDLLib class diagram

Table 5: List of operations of the DFDLLib class

Operation name	Input	Output	Description
initLib	String	<void>	Initialize the DFDL4S library respecting the time setting passed (UTC / TAI)
interpretDocument	String String	Document	This method interprets the contents of a binary file according to the specifications of a schema file. Returns the element (document) containing all element items available in the binary file.
interpretDocument	String byte[]	Document	This method interprets the contents of a memory block according to the specifications of a schema file. Returns the element (document) containing all element items available in the binary file.

Operation name	Input	Output	Description
interpretDocument	String String InterpreterMonitor	Document	<p>This method interprets the contents of a binary file according to the specifications of a schema file.</p> <p>The InterpreterMonitor parameter allow the user to register/defined their own interpretation/load status of binary data file.</p> <p>Returns the element (document) containing all element items available in the binary file.</p> <p>Note: This method is not available in dfdl4s++ library</p>
interpretDocument	String byte[] InterpreterMonitor	Document	<p>This method interprets the contents of a memory block according to the specifications of a schema file.</p> <p>The InterpreterMonitor parameter allow the user to register/defined their own interpretation/load status of binary data file.</p> <p>Returns the element (document) containing all element items available in the binary file.</p> <p>Note: This method is not available in dfdl4s++ library</p>
interpretDocument	String String byte[][] byte[][] HashMap<DataSize, DataSize> String InterpreterMonitor	Document	<p>This method interprets the contents of a binary file according to the specifications of a schema file. Also allow the synchronization of data and solve possible ambiguities in schemas (ex: products with same APID).</p> <p>The InterpreterMonitor parameter allow the user to register/defined their own interpretation/load status of binary data file.</p> <p>Returns the element (document) containing all element items available in the binary file.</p> <p>Note: This method is not available in dfdl4s++ library.</p>

Operation name	Input	Output	Description
interpretDocument	String byte[] byte[][] byte[][] HashMap<DataSize, DataSize> String InterpreterMonitor	Document	This method interprets the contents of a memory block according to the specifications of a schema file. Also allow the synchronization of data and solve possible ambiguities in schemas (ex: products with same APID). The InterpreterMonitor parameter allow the user to register/defined their own interpretation/load status of binary data file. Returns the element (document) containing all element items available in the binary file. Note: This method is not available in dfdl4s++ library.
createNewDocument	String	Document	Create new document with the specific schema file
storeDocument	Document String	<void>	This method stores the document in the specific path file.
storeDocument	Document String Int Int	<void>	Export a given range of packets to a given file.
storeDocument	Document	byte[]	This method stores the document in a byte[]. Return the byte[] with the document data.
appendElements	Document String byte[]	<void>	This method adds new elements to a document based on data generation parameters
appendElements	Document String int int byte String	<void>	This method adds new elements to a document based on raw data.
getVersion	-	String	The version number and release date of the library.

### 4.2.2. Domain Entities

The classes contained in this package are responsible for modelling the contents of a binary file.

Figure 4 shows class diagram of the Domain Entities package.



**Figure 4: Domain Entities package class diagram**

Classes contained in this package are:

- *Document*: The Document class represents the root of the domain element that is used to structure the binary data.
- *Element*: The Element class represents a domain element that is used to structure the binary data.
- *Element.REPRESENTATION\_TYPE*: The REPRESENTATION\_TYPE enumerator contains all the different representation types that can be present in the schemas.
- *DataSize*: The DataSize class represents the size of a data block. It provided byte and bit reference.
- *ErrorIndicator*: The ErrorIndicator class represent error information related to instances of Element.

The above-mentioned classes are described more in detail in the next sections.

#### 4.2.2.1. Document class

The Document class provides methods to interact with the domain structure that represents the binary data through Elements (like nodes in a binary tree). These methods are listed in Table 6.

The typical usage for this class is to get the root element – document.childAt(0) – and to close the document - document.close().

**Table 6: List of operations of the Document class**

Operation name	Input	Output	Description
size	-	Long	Access the size of the stored data.
getCleanAlignedData	DataSize DataSize	byte[]	Retrieve the array of bytes storing information of the given element. The content of the byte array is shifted to guarantee that significant bits begin at the least significant bit of the array. Returns the byte array containing data (in big endian) for the element
getRawData	DataSize DataSize	byte[]	Retrieve the array of bytes storing information of the given element. According to the element size/offset, this byte array may contain left and/or right bits that are not relevant. Returns the byte array containing data (in big endian) for the element
close	-	<void>	Close the document, releasing all associated resources.
childAt	int	Element	Access the child at the given index. Returns the requested child element.
childAt	int String String	Element	Access the child at the given index, with the possibility of evaluated internal ISP in TF files or internal TF in CADU files.
childCount	-	int	Returns the number of children of the element.



Operation name	Input	Output	Description
getSyncHistory	-	List<ReportEvent>	Get the set of errors registered during document interpretation. Return the List of ReporEvents containing all the synchronization errors registered during document interpretation
createElement	String	Element	Create a root element
createElement	String Element	Element	Create a child element of the given element
createElementTime	String Element String	Element	Create a child element of the given element with the specific String Time value
createElementInteger	String Element BigInteger	Element	Create a child element of the given element with the specific BigInteger type value
createElementFloat32	String Element Float	Element	Create a child element of the given element with the specific Float type value
createElementFloat64	String Element Double	Element	Create a child element of the given element with the specific Double type value
createElementBytes	String Element byte[]	Element	Create a child element of the given element with the specific byte[] type value
validate	-	<void>	Validate "handcrafted" document

#### 4.2.2.2. Element class

The Element class represents a domain element that is used to structure the binary data.

The Element class provides:

- Methods to get the value of an element (see Table 7);
- Methods to set the value of an element (see Table 8);
- Methods to get the properties of an element (see Table 9);

**Table 7: Element class: List of methods for getting the value of an element**

Method name	Input	Output	Description
getValueAsRepresentation	-	String	Access the value of the element according the representation (if dmx:representation is present in schema) or intrinsic type (more information about representation type can be found in section 6).
getValueAsRepresentation	REPRESENTATION_TYPE	String	Access the value of the element according to the given representation type (more information about representation type can be found in section 6).
getValueBytes	-	byte[]	Access the clean and aligned data of the element
getValueBinary	-	String	Access the value of the element (according to the representation specified in the binary definition)
getValueFloat32	-	Float	Access the value of the element (according to the 'FLOAT_32' representation)
getValueFloat64	-	Double	Access the value of the element (according to the 'FLOAT_64' representation)
getValueHexadecimal	-	String	Access the value of the element (according to the 'HEXADECIMAL' representation)
getValueInteger	-	BigInteger	Access the integer value of the element
getValueTime	-	String	Access the value of the element (according to the 'TIME' representation)
retrieveRawData	-	byte[]	Access the raw data of the element. Notice that data is not word aligned and may require cleaning of leading and trailing bits.
retrieveRawData	int int	byte[]	Access the raw data of the element according to the desired size and offset. Notice that data is not word aligned and may require cleaning of leading and trailing bits.

**Table 8: Element class: List of methods for setting the value of an element**

Method name	Input	Output	Description
setValueBytes	byte[]	<void>	Update the raw data of the element
setValueFloat32	Float	<void>	Set the value of the element (according to the 'FLOAT_32' representation)
setValueFloat64	Double	<void>	Set the value of the element (according to the 'FLOAT_64' representation)
setValueInteger	BigInteger	<void>	Set the value of the element (according to the 'INTEGER' representation)
setValueTime	String	<void>	sets the element value (element shall be of type Time, otherwise an exception is raised)

**Table 9: Element class: List of methods for getting properties of an element**

Operation name	Input	Output	Description
name	-	String	Get the name of the element.
absoluteName	-	String	Get the absolute path to the Element, containing the root node.
parent	-	Element	Get the parent of the element
root		Element	Get the packet element (below document) that contains this element.
childCount	-	int	Get the number of children of the element in the data stream
childAvailableCount	-	int	Get the number of children available in the element according to the schema definition
childAt	int	Element	Access the child at the given index
propertyValueGet	String	String	Access the value of a property with a given name

Operation name	Input	Output	Description
size	-	DataSize	Get the DataSize object containing the expected size (number of bytes and bits) of the Element according to the schema.
sizeAvailable	-	DataSize	Get the DataSize object containing the size available (number of bytes and bits) of the Element according to the schema.
offset	-	DataSize	Get the DataSize object containing the number of bytes and bits corresponding to the offset of the element
getError	-	ErrorIndicator	Access error indicator related to this element
getIntrinsicType	-	String	Gets the element intrinsic type (xsd type)
getRangeMaximum	-	BigInteger	For XSD_TYPES BYTE, SHORT, INT or LONG, return the maximum type value based on size
getRangeMinimum	-	BigInteger	For XSD_TYPES BYTE, SHORT, INT or LONG, return the minimum type value based on size
is	Element	boolean	Check if this element is an ElementTrait instance
evaluate	Element	String	Gets the element for a given path expression
evaluateBoolean	Element	String	Evaluate an expression known to be a boolean value
evaluateInteger	Element	String	Evaluate an expression known to be an integer value

#### **4.2.2.3. ErrorIndicator**

The ErrorIndicator class stores error information related to instances of Element. This information can be retrieved from Element instances through the Element class member functions (methods) getChildErrors (for the object's children) and getError (for the Element object itself).

Table 10: List of operations of the ErrorIndicator class

Operation name	Input	Output	Description
hasError	-	boolean	Access the error status
errorMessage	void	std::string	Access the error message
isPropagateable	-	boolean	Indicates the presence of propagateable errors
isSevere	-	boolean	Indicates the occurrence of severe error
isExpressionMalformed	-	boolean	Indicates the presence of expression malformed
getPathElementNotFound	-	boolean	Indicates the use of invalid path to find an element
hasNonSevereError	-	boolean	Indicates the occurrence of a non severe error

### 4.2.3. Utilities

The classes and interfaces contained in this package are utility classes/interfaces.

Figure 5 shows diagram of the Utilities package.

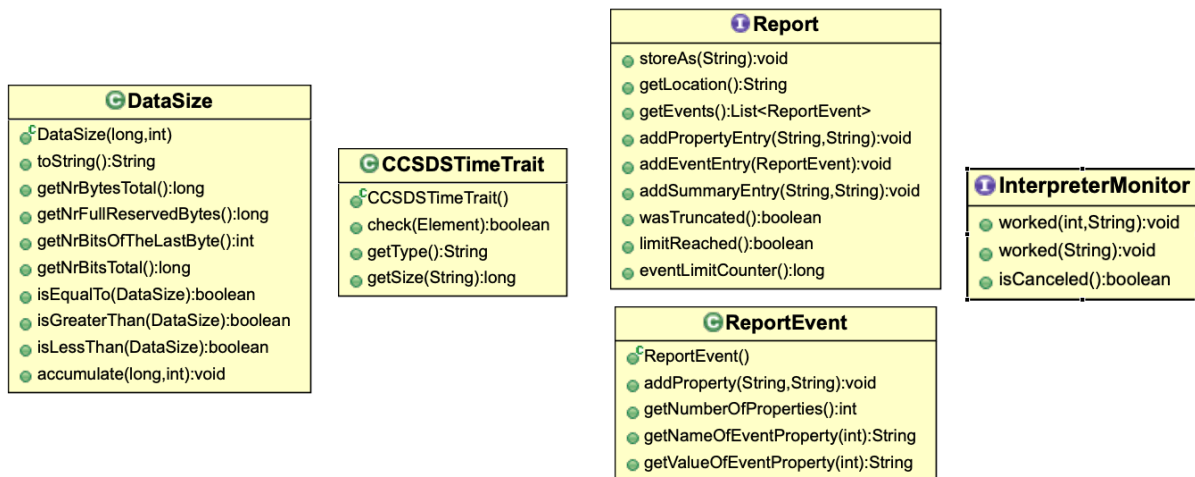


Figure 5: Utilities package diagram

#### 4.2.3.1. DataSize

The class DataSize represents the size of a data block. It provided byte and bit reference

**Table 11: List of operations of the DataSize class**

Operation name	Input	Output	Description
DataSize	long int	DataSize	Constructor that allow the user create a new instance of DataSize object containing the given number of bytes and bits
getNrBytesTotal	-	long	Get the number of bytes necessary to store a block of data of size this.
getNrFullReservedBytes	-	long	Get the number of full reserved bytes of the element
getNrBitsOfTheLastByte	-	int	Access the number of bits of the last byte reserved for the element
getNrBitsTotal	-	long	Access the total number of bits covering the data of this size
isEqualTo	DataSize	Boolean	Checks if this object equals another size instance
isEqualTo	DataSize	Boolean	Checks if this object equals another size instance
isGreaterThan	DataSize	Boolean	Checks if this object is greater than another size instance
isLessThan	DataSize	Boolean	Checks if this object is less than another size instance
accumulate	long int	Void	Accumulate this size object with another size object

#### 4.2.3.2. CCSDSTimeTrait

CCSDSTimeTrait implements a generic trait that can be use to check for traits on Elements.

**Table 12: List of operations of the CCSDSTimeTrait class**

Operation name	Input	Output	Description
check	Element	Boolean	Check for traits on Element
getType	-	String	Get the CCSDSTime type. The result string is restricted to the list of currently supported time types, as specified by CCSDSTime.Type enum.

Operation name	Input	Output	Description
getSize	String	long	<p>Get the size in bits associated with the given property.</p> <p>The list of properties, related to a given time code field, is composed of the names of the fields that compose it.</p> <p>According to the current schema conventions, consider the following properties per each time type:</p> <ul style="list-style-type: none"> <li>- CUC/CUC_seconds_nanoseconds <ul style="list-style-type: none"> <li>- Coarse_Time</li> <li>- Fine_Time</li> </ul> </li> <li>- CDS <ul style="list-style-type: none"> <li>- Days</li> <li>- Milliseconds</li> <li>- Submilliseconds</li> </ul> </li> <li>- CCS <ul style="list-style-type: none"> <li>- Year</li> <li>- Month</li> <li>- Hours</li> <li>- Minutes</li> <li>- Seconds</li> <li>- Subseconds</li> <li>- Day_of_Month</li> <li>- Day_of_Year</li> </ul> </li> <li>- A_DEF <ul style="list-style-type: none"> <li>- Counter</li> </ul> </li> </ul> <p>Use enum CCSDSTimeTrait.Property as reference property values.</p>

This functionality can be used as:

```

CCSDSTimeTrait ccldsTimeTrait = new CCSDSTimeTrait();
if (element.is(ccldsTimeTrait)) {
    if (ccldsTimeTrait.getType().equals(CCSDSTime.Type.CUC.name())) {
        long coarseTimeSize = ccldsTimeTrait.getSize(CCSDSTimeTrait.Property.CUC_Coarse_time.getValue());
        long fineTimeSize = ccldsTimeTrait.getSize(CCSDSTimeTrait.Property.CUC_Fine_time.getValue());
    }
}

```

#### 4.2.3.3. Report

The Report interface defines the interaction methods related to reporting. The Report is divided in 3 sections:

1. Properties Entries
  - a. The user can add properties related to the report (For example: File name, file path, file size, date...)
2. Summary Entries

- a. The user can add summary into the report (For example: number of isp analysed, the number o CRC errors found in the isp file, ...)

### 3. Event Entries

- a. The user can add events into the report. This events should report specific properties of a given element. (For example: reporting the presence of CRC error in a specific element)

An example of Report usage can be found in S2G client. S2G allow users to generate reports where different type of errors can be detected.

Currently, the implementation of this interface is only supported in DFDL4S java library.

This interface will be implemented in DFDL4S v1.7.0 to create a base Report.

**Table 13: List of operations of the Report interface**

Operation name	Input	Output	Description
storeAs	String	Void	Store the report in the given path location
getLocation	-	String	Access the default location where the report was save.
getEvents	-	List	Return events contained in the report
addPropertyEntry	String String	Void	Add a new property entry to the report
addEventEntry	ReportEvent	Void	Add a new event entry to the report
addSummaryEntry	String String	Void	Add a new summary entry to the report
wasTruncated	-	Boolean	Check if the report has been truncated
limitReached	-	Boolean	Check if the limit of the report was reached
eventLimitCounter	-	long	Access the maximum number of events in the report

#### 4.2.3.4. ReportEvent

The ReportEvent class defines an event to be included in the report generation.

**Table 14: List of operations of the ReportEvent class**

Operation name	Input	Output	Description
addProperty	String String	Void	Add a property to describe the event
getNumberOfProperties	-	Int	Gets a set with the names of the properties of the event
getNameOfEventProperty	Int	String	Get the name of the property of the event in the given index
getValueOfEventProperty	Int	String	Get the value of the property of the event in the given index



#### 4.2.3.5. InterpreterMonitor

The InterpreterMonitor interface described the methods used to monitor the Interpreter Progress.

This interface support the callback functionality and can be used in:

- DFDLLib.interpreterDocument(String, String, InterpreterMonitor);
- DFDLLib.interpreterDocument(String, String, byte[[]], byte[[]], HashMap<DataSize, DataSize>, ambiguityChoice, InterpreterMonitor);

Currently, this functionality/interface is only supported in DFDL4S java library.

*Table 15: List of operations of the InterpreterMonitor class*

Operation name	Input	Output	Description
worked	int String	Void	Register the current progress of the Interpreter
worked	String	Void	Register a message about progress of the Interpreter
isCanceled	Int	boolean	Checks is monitor has been canceled

### 4.2.4. Exceptions

#### 4.2.4.1. ErrorLoadingException

Class that represents a Java exception of type ErrorLoadingException. It inherits the Java Exception methods and properties.

## 4.3. Process logic

In this section, the process logic of using the library in model's source code is shown. It is described for Java developments.

### 4.3.1. Java Programming Language

#### 4.3.1.1. DFDLLibrary

Steps for using the DFDLLibrary module.

1. Import package org.esa.s2g.dfllib.DFDLLib in your code;
2. Use the static functions as a library of functionalities. No instance creation is needed:
  - 2.1. Initialize the library's time definitions using method `initLib()` and passing it the path to the time definition file;

```
DFDLLib.initLib("resources/time");
```

- 2.2. Access a data file using method `interpretDocument()` and passing it the path to the Mission Specification Schema and the path to the data file.

```
DFDLlib.interpretDocument("resources/data/Sentinel2X-bandTM/Sentinel2X-bandTMCADU.xsd",  
"tests/data/Sentinel 2 X-band TM/Sentinel 2 X-band TM CADU.bin");
```

#### 4.3.1.2. Domain Entities

Steps for using data structures in the DomainEntities module.

1. Import package org.esa.s2g.dfdl.\* in your code;
2. Create an instance of a Document class from the return value of DFDLlib.interpretDocument() method. The method may throw an exception in case of error, so remember to catch it;

```
Document document = DFDLlib.interpretDocument("resources/data/Sentinel2X-bandTM/Sentinel2X-  
bandTMTF.xsd", "tests/data/Sentinel 2 X-band TM/Sentinel 2 X-band TM TF.bin");
```

3. Access the data using the methods explained in section 4.2.2.

```
System.out.println(document.getChildAt(0)); //output the entire value of the first data unit in the file
```

4. Close the document once not needed.

```
document.close();
```

## 4.4. Example of use

### 4.4.1. Java Programming Language

Below is an example of Java code that uses basic capabilities modules of the DFDL4S library to create a document based on a Sentinel 3 schema. The result of this example is an ISP document file containing a number of packets – see distribution package for the entire example.

```
import java.io.IOException;  
import java.math.BigInteger;  
import java.util.Arrays;  
import java.util.List;  
  
import org.esa.s2g.dfdlib DFDLlib;  
import org.esa.s2g.dfdlib Document;  
import org.esa.s2g.dfdlib Element;  
import org.esa.s2g.dfdlib ElementFinder;  
import org.esa.s2g.dfdlib ErrorLoadingException;  
  
/**  
 * CreateNewDocumentExample is a simple howto example on how to exercise the DFDL4S Packet creation capabilities  
 */  
public class CreateNewDocumentExample {  
  
    private static final String OUTPUT_DIRECTORY = "./data";  
  
    public static void main(String[] args) {  
  
        try {  
            System.out.println("\n\n***** Example BEGIN *****\n");  
  
            DFDLlib.initLib("resources/time");  
  
            final String schemaFile = "schema/Sentinel3/Sentinel3X-bandTMISP.xsd";
```

```

final String outputFilename = OUTPUT_DIRECTORY + "/S3_SYNT_RAW_ISPEXAMPLE.RAW";

Document document = createDocument(schemaFile, PacketData.create());
DFDLlib.storeDocument(document, outputFilename);

System.out.println("\n\n***** Example END *****\n\n");

} catch (IOException e) {
    System.err.println(e.getMessage());
    System.exit(-1);
}
}

private static Document createDocument(String schemaFile, List<PacketData> packets) {
    Document document = DFDLlib.createNewDocument(schemaFile);
    fillDocument(document, packets);
    document.validate();
    return document;
}

private static void fillDocument(Document document, List<PacketData> packets) {
    for (PacketData packet : packets) {
        Element isp = document.createElement("ISP");
        fillISP(document, isp, packet);
    }
}

private static void fillISP(Document document, Element isp, PacketData packet) {
    Element primaryHeader = document.createElement("Packet_Primary_Header", isp);
    fillISPPrimaryHeader(document, primaryHeader, packet);

    Element dataField = document.createElement("Packet_Data_Field", isp);
    fillISPDDataField(document, dataField, packet);
}

private static void fillISPPrimaryHeader(Document document, Element primaryHeader, PacketData packet) {
    document.createElementInteger("Packet_Version", primaryHeader, PacketData.version);

    Element packetIdentification = document.createElement("Packet_Identification", primaryHeader);

    document.createElementInteger("Packet_Type", packetIdentification, PacketData.type);
    document.createElementInteger("Secondary_Header_Flag", packetIdentification, PacketData.secondaryHeaderFlag);

    Element apid = document.createElement("APID", packetIdentification);
    document.createElementInteger("PID", apid, packet.pid);
    document.createElementInteger("PCAT", apid, packet.pcat);

    Element packetSequenceCtrl = document.createElement("Packet_Sequence_Ctrl", primaryHeader);

    document.createElementInteger("Sequence_Flags", packetSequenceCtrl, PacketData.sequenceFlags);
    document.createElementInteger("SSC", packetSequenceCtrl, packet.ssc);

    document.createElementInteger("Packet_Data_Length", primaryHeader, packet.dataLength);
}

[...snip...]
}

/**
 * PacketData stores the information used to describe an instrument source packet
 */
class PacketData {

[...snip...]

    static List<PacketData> create() {
        return Arrays.asList(
            // |----- Header -----|
            // | |APID| |SERVICE| |TIME| |INST |
            // |PID PCAT SSC LENGTH TYPE SUBTYPE COARSE FINE STATUS NAME |
            new PacketData(22, 9, 15875, 189, 3, 25, 1103186353, 0, 1, "NAVATT"),
            new PacketData(82, 11, 285, 51, 201, 21, 1174739426, 11826142, 1, "MWR" ),
            new PacketData(74, 0, 14662, 19174, 201, 31, 1103206220, 13913716, 0, "SLSTR"),
            new PacketData(22, 9, 15875, 189, 3, 25, 1103186354, 0, 1, "NAVATT"),
[...snip...]
        );
    }
}

```

#### 4.4.1.1. Java Build and Execution process

To compile the sources you must specify the location of the source file and the DFDL4S library jar file:

```
javac -cp <path to dfdl4slib jar file> <source code java file>
```

This command is a valid example for compiling the above example (assuming the working directory is the DFDL4S installation folder):

```
javac -cp ../lib/dfdl4s.jar DFDLLibTest.java
```

The result of this command is a compiled code “.class” file (e.g. DFDLLibTest.class)

For executing the compiled code you must specify the location of all required jar files and the full class name (including package definition when applicable) of the main class.

```
java -cp <composite path to jar files> <full class name>
```

The command for executing the example is:

```
java -cp ../lib/dfdl4s.jar:../lib/*:. DFDLLibTest
```

## 5. MISSION CONFIGURATION

DFDL4S takes a set of mission configurations as inputs, composed of several separate files. The library provides a set of sample mission configurations, that the user can expand.

The files composing the Mission Configuration (Figure 6) provide a wide range of configuration parameters used by the library, grouped as Mission Data Definition schemas. The Mission Data Definition schema files are a set of schemas that define the binary contents of the several levels of packages (CADU, TF and ISP) based on DFDL [RD.1].

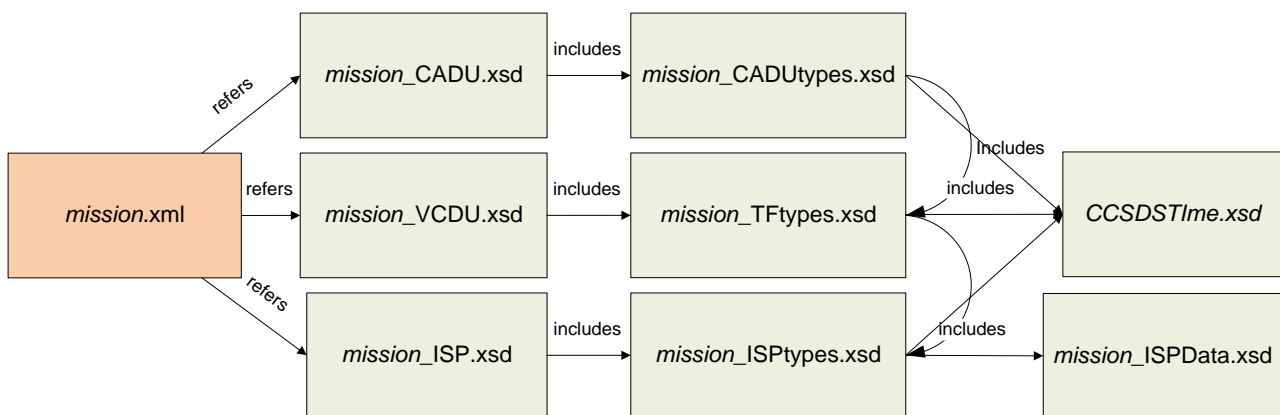


Figure 6: Mission Configuration files structure

The following sections present details of the Mission Data Definition Schemas.

### 5.1. Mission Data Definition Schemas

The Mission Data Definition schemas are XSD schemas adapted 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 6 has been used. Section 5.1.1 provides some guidelines on how the user can customize an existing mission configuration.

#### 5.1.1. 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 7 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 (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 "PacketHeader" and "PacketData"; the types of the two sub-elements ("TypePacketHeader" 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.

```
<?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="Sentinel13X-bandTMISPTypes.xsd"/>

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/dfdl/">
      <dfdl:format byteOrder="bigEndian" encoding="utf-8" />
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="ISP" dmx:representation="Complex">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PacketHeader" type="TypePacketHeader"/>
        <xs:element name="PacketData" type="TypePacketData"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

**Figure 7: Mission Schema file (Data Unit definition)**

The Figure 8 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", "TypePacketHeader", "TypeDataFieldHeader\_OLCI", "TypeDataHeader")

When necessary, complex types can be built upon the definition of other types (e.g. the type "TypePacketHeader" defines a sequence containing an element of type "TypeAPID"); or can be defined over simple types (e.g. the element "seqCount" 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 Appendix A – Compatibility with DFDL Core Set and detailed 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="TypePacketHeader">
    <xs:sequence>
[...snip...]
      <xs:element name="APID" type="TypeAPID"/>
[...snip...]
      <xs:element name="seqCount"
        type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="14"
        dmx:representation="Integer16"/>
      <xs:element name="dataFieldLength"
        type="xs:int" dfdl:lengthKind="explicit" dfdl:lengthUnits="bits" dfdl:length="16"
        dmx:representation="Integer16"/>
    </xs:sequence>
  </xs:complexType>

[...snip...]

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

  <xs:complexType name="TypeDataHeader">
    <xs:choice>
      <xs:element name="OLCIHeader" type="TypeDataFieldHeader_OLCI">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/dfdl-1.0">
            <dfdl:discriminator
              test="{../../../../PacketHeader/APID
                in [1056,1057,1058,1059,1060,1061,1062,1063,1064,1065]}"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:choice>
  </xs:complexType>

[...snip...]

  <xs:complexType name="TypeData">
    <xs:sequence>
      <xs:element name="data"
        type="xs:byte" dfdl:lengthKind="explicit" dfdl:lengthUnits="bytes"
        dfdl:length="{../../../../PacketHeader/dataFieldLength
          + 1 - length(../../../../PacketData/DataHeader) - 2}"
        dmx:representation="Hexadecimal"/>
    </xs:sequence>
  </xs:complexType>

[...snip...]

  <xs:complexType name="TypePacketData">
    <xs:sequence>
      <xs:element name="DataHeader" type="TypeDataHeader"/>
      <xs:element name="Data" type="TypeData"/>
      <xs:element name="CRC" type="TypeCRC"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

Figure 8: Mission Schema file (Data Unit types definition)

## 6. REPRESENTATION TYPES

As expected from a low level definition language, DFL does not provide a way to specify the representation of the nodes in the information model, meaning that although DFL allows to access the data in a structured approach it does not provide insight as how it should be interpreted by the user. Such interpretation of the data must be done according to a semantic meaning relevant to the user.

Since there is the intent to help the operator to inspect and understand the raw data contents of binary packets, DFL4S must extend DFL with a mechanism to interpret the raw data and transform it into a human readable format.

To accomplish this goal, DFL4S provides the concept of representation type to be the semantic type used to represent the data in the viewer. Although the representation type can be understood as the default type of the data, the tool also allows the user to view the raw data (presented as binary or hexadecimal) in the hexadecimal pane.

The representation type is thus closely related to the data definition, and is expected to be provided by the user upon mission schema definition. As DFL extends the XML Schema definition by adding attributes that describe properties of the represented binary data, DFL4S relies upon one such extension to define the representation type of elements in the information model. As presented in Figure 9, a representation attribute can be used to provide information to about the representation type.

```
...  
<xs:element name="packet">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="counter" dfdl:length="2" dmx:representation="Float16"/>  
      <xs:element name="date" dfdl:length="4" dmx:representation="Integer32"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>  
...
```

**Figure 9: Example of representation type usage**

The value of the representation property must be defined as one of the values presented in the next paragraphs.

### 6.1. Basic Types Representations

#### 6.1.1. Character8

*Character8* represents a character stored as an unsigned integer value (with 1 byte or octet of size) ranging from 0 to 255, that is displayed using UTF-8 representation. Control characters – non-printing – are replaced by character '!'.  
.



### **6.1.2. Character16**

*Character16* represents a wide character stored as an unsigned integer value (with 2 bytes of size) ranging from 0 to 65535, that is displayed using UTF-16 representation. Control characters – non-printing – are be replaced by character '!'.

### **6.1.3. String**

*String* represents an octet string to be displayed in UTF-8. The size of this representation is determined by the size of the underlying element to be represented. For example, is an element says that it has length 100 bytes, its representation as String shall show in the GUI a string with at most 100 characters.

### **6.1.4. Integer8**

*Integer8* represents a signed integer value (with 1 byte or octet of size) ranging from -128 to 127, that is displayed as an integer.

### **6.1.5. UInteger8**

*UInteger8* represents an unsigned integer value (with 1 byte or octet of size) ranging from 0 to 255, that is displayed as an integer.

### **6.1.6. Integer16**

*Integer16* represents a signed integer value (with 2 bytes) ranging from -32,768 to 32,767, that is displayed as an integer.

### **6.1.7. UInteger16**

*UInteger16* represents an unsigned integer value (with 2 bytes) ranging from 0 to 65,535, that is displayed as an integer.

### **6.1.8. Integer32**

*Integer32* represents a signed integer value (with 4 bytes) ranging from -2,147,483,648 to 2,147,483,647, that is displayed as an integer.

### **6.1.9. *UInteger32***

*UInteger32* represents an unsigned integer value (with 4 bytes) ranging from 0 to 4,294,967,295, that is displayed as an integer.

### **6.1.10. *Integer64***

*Integer64* represents a signed integer value (with 8 bytes) ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, that is displayed as an integer.

### **6.1.11. *UInteger64***

*UInteger64* represents an unsigned integer value (with 8 bytes) ranging from 0 to 18,446,744,073,709,551,615, that is displayed as an integer.

### **6.1.12. *Float16***

*Float16* represents floating point value (with 2 bytes of size), otherwise referred to as half precision. This value shall be interpreted according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754).

### **6.1.13. *Float32***

*Float32* represents floating point value (with 4 bytes of size), otherwise referred to as single precision. This value shall be interpreted according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754).

### **6.1.14. *Float64***

*Float64* represents floating point value (with 8 bytes of size), otherwise referred to as double precision. This value shall be interpreted according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754).

### **6.1.15. *Binary***

*Binary* represents a binary string. The size of this representation is determined by the size of the underlying element to be represented. The viewer represents each of the octets related to the element.

### **6.1.16. *Hexacimal***

*Hexadecimal* represents a hexadecimal string. The size of this representation is determined by the size of the underlying element to be represented. The viewer represents each of the octets related to the element.

### 6.1.17. Fragmented Signed Integers

DFDL4S processes *fragmented* signed integers (i.e. the signed integer field is not byte aligned, or is laid in a number of bits other than 8, 16, 32, 64) represented considering *sign-magnitude representation* – the most significant bit is used as sign bit (0 for positive values, 1 for negative values).

DFDL4S first accesses the byte array according to the schema intrinsic type specification (e.g `xs:int`), and evaluates the signed integer value. If requested by the user, DFDL4S then adjusts the value to the respective representation type (e.g. `Integer32`).

For instance, the following are examples of the definition of fragmented signed integers:

- ```
<xs:element name="SSC" type="xs:int" dfdl:lengthKind="explicit"
dfdl:lengthUnits="bits" dfdl:length="14" dmX:representation="Integer32"/>
```

This would be a 14 bit array where the most significant bit is considered the sign bit. In this case, the valid range of the integer value is between  $-8192 (2^{13})$  and  $8191(2^{13}-1)$ .

- ```
<xs:element name="Flag" type="xs:int" dfdl:lengthKind="explicit"
dfdl:lengthUnits="bits" dfdl:length="1"/>
```

This edge case represents a flag as a one bit signed integer. Most likely a schema problem, DFDL4S would access the value as either -1 or 0.

## 6.2. Representation types after DFDL4S v1.5.2

From DFDL4S v1.5.2 on, the `dmX:representation` can be safely removed and use the standard types defined for `xs:type`.

Two representation types are not covered by standard `xs:types` that are most useful for standard S2G missions and are supported by the supporting DFDL4S methods:

- Binary:

```
String getValueAsString()
byte[] getValueBytes()
String getValueHexadecimal()
```

- Time:

```
String getValueAsString()
byte[] getValueBytes()
String getValueHexadecimal()
String getValueTime()
```

## 6.3. Time Code Representations

DFDL4S is capable of representing time code as specified by the CCSDS.

The Time Code Representations specified in this section depend a reference Epoch. To provide the reference Epoch, the schema of the mission must specify the attribute `dmX:epoch` with the name of a supported reference epoch (Table 16). The following snippet specifies a CUC Time Code element named `GNSR_TIME` based on the GPS reference epoch.

```
<xs:element name="GNSR_Time" type="TypeTimeCode_CUC_32_16" dmX:epoch="GPS"/>
```

**Table 16: Description of epoch property to support time definitions**

Property	Description
<b>dmx:epoch</b>	This property identifies the reference epoch for the time field. Available standard values are “GPS”, “CCSDS” and “GALILEO”. Additionally, the epoch may also be a custom value in the format YYYY-MM-DD'T'hhmmss (e.g. "2000-01-01T00:00:00").

### 6.3.1. CUCTime

CUC is used to represent CCSDS Unsegmented Time Code (Generic CCSDS), and it consists of a

- Preamble field (P-field) that gives information about:
  - Octet 1 (mandatory if P-Field is used)
    - time code ID to distinguish between 1958 Jan 1 epoch (Level 1) or Agency-defined epoch (Level 2)
    - number of octets used for the coarse and fine time.
  - Octet 2 (optional—presence is signaled in Octet 1)
    - Additional number of octets used for the coarse and fine time
- Time field (T-field) that consists of:
  - Coarse time: seconds from given reference epoch (ranging in length from 1 to 7 octets)
  - Fine time: sub-seconds within the second (ranging in length from 0 to 10 octets)

This time code is not UTC based, and is relative with respect of the given reference epoch.

		Time Code	
name		Coarse Time	Fine Time
bits		32	16
bytes		6	

**Figure 10: CUC Time Code example**

Figure 10 shows an example of a CUC Time code with 4 bytes allocated for the Coarse Time and 2 bytes for the Fine Time. This Time code can be defined with the following DFDL element:

```
<xs:complexType name="TypeTimeCode_CUC_32_16" dmX:ccsdsType="CUC" dmX:preambleIncluded="false">
  <xs:sequence>
    <xs:element name="Coarse_Time" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="32" dmX:representation="UInteger32"/>
    <xs:element name="Fine_Time" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="16" dmX:representation="UInteger32"/>
  </xs:sequence>
</xs:complexType>
```

```
</xs:sequence>
</xs:complexType>
```

### 6.3.2. CUCSecondsNanosecondsTime

*CUCSecondsNanoseconds* is used to represent special cases of CCSDS Unsegmented Time Code (Ex: Used in MetOp mission), and it consists of a

- Preamble field (P-field) that gives information about:
  - Octet 1 (mandatory if P-Field is used)
    - time code ID to distinguish between 1958 Jan 1 epoch (Level 1) or Agency-defined epoch (Level 2)
    - number of octets used for the coarse and fine time.
  - Octet 2 (optional—presence is signaled in Octet 1)
    - Additional number of octets used for the coarse and fine time
- Time field (T-field) that consists of:
  - Coarse time: seconds from given reference epoch (ranging in length from 1 to 7 octets)
  - Fine time: nanoseconds from given reference epoch (ranging in length from 0 to 10 octets)

This time code is not UTC based, and is relative with respect of the given reference epoch.

Time Code	
name	
	<i>Coarse Time</i> <i>Fine Time</i>
bits	56                                      80
bytes	17

**Figure 11: CUCSecondsNanoseconds Time Code example**

Figure 11 shows an example of a CUCSecondsNanoseconds Time code with 7 bytes allocated for the Coarse Time and 10 bytes for the Fine Time. This Time code can be defined with the following DFDL element:

```
<xs:complexType name="TypeTimeCode_CUCSecondsNanoseconds_56_80" dm:ccsdsType="
CUC_seconds_nanoseconds" dm:preambleIncluded="false">
  <xs:sequence>
    <xs:element name="Coarse_Time" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="56" dm:representation="UInteger32"/>
    <xs:element name="Fine_Time" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="80" dm:representation="UInteger32"/>
  </xs:sequence>
</xs:complexType>
```

### 6.3.3. CDSTime

*CDS* is used to represent CCSDS Day Segmented Time Code (Generic CCSDS), and consists of a

- Preamble field (P-field) that gives information about:
  - time code ID
  - reference epoch
  - length of the day segment and sub-millisecond segment
- Time field (T-field) that consists of:
  - days from given reference epoch (ranging in length from 2 to 3 octets)
  - Milliseconds within the day (with 4 octets length)
  - Sub-milliseconds within the day (with length 0, 2 or 4 octets)

This time code is UTC based, and represents absolute time.

Below is an example DFDL definition of a CDS time:

```
<xs:complexType name="TypeTimeCode_CDS_16_32_16" dmx:ccsdsType="CDS" dmx:preambleIncluded="false">
  <xs:sequence>
    <xs:element name="Days" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="16" dmx:representation="UInteger32"/>
    <xs:element name="Milliseconds" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="32" dmx:representation="UInteger32"/>
    <xs:element name="Submilliseconds" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="16" dmx:representation="UInteger32"/>
  </xs:sequence>
</xs:complexType>
```

### 6.3.4. CCSTime

CCS is used to represent CCSDS Calendar Segmented Time Code (Generic CCSDS), and consists of a

- Preamble field (P-field) that gives information about:
  - time code ID
  - calendar variation type
  - number of sub-second segments
- Time field (T-field) that can be defined according to several variations as specified in the P-field:
  - For the Year-Month-Day of month variation the T-field consists of
    - Year (with length 2 octets)
    - Month (with length 1 octet)
    - Day of month (with length 1 octet)
    - Hour (with length 1 octet)
    - Minute (with length 1 octet)
    - Second (with length 1 octet)
    - Sub-Second (with length ranging from 0 to 6 octets)
  - For the Year- Day of month variation the T-field consists of
    - Year (with length 2 octets)

- Day of year (with length 2 octets)
- Hour (with length 1 octet)
- Minute (with length 1 octet)
- Second (with length 1 octet)
- Sub-Second (with length ranging from 0 to 6 octets)

This time code is UTC based, and represents absolute time.

Below is an example DFDL definition of a CCS time Day-of-Month variation:

```
<xs:complexType name="TypeTimeCode_CCS_DOM_2DP" dmxc:ccsdsType="CCS" dmxc:preambleIncluded="false">
  <xs:sequence>
    <xs:element name="Year" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="16" dmxc:representation="UInteger32"/>
    <xs:element name="Month" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
    <xs:element name="Day_of_Month" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
    <xs:element name="Hours" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
    <xs:element name="Minutes" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
    <xs:element name="Seconds" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
    <xs:element name="Subseconds" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="8" dmxc:representation="UInteger32"/>
  </xs:sequence>
</xs:complexType>
```

### 6.3.5. CCSASCII Time

ASCII is used to represent CCSDS Calendar Segmented Time (ASCII) Code (Generic CCSDS), and it consists of a

- Time field (T-field) is a ASCII string defined according to several variations:
  - For Code A format: YYYY-MM-DDThh:mm:ss.d → dZ
  - For Code B format: YYYY-DDDThh:mm:ss.d → dZ

This time code is UTC based, and represents absolute time.

### 6.3.6. Agency Defined Time

A\_DEF is used to represent an Agency defined Time Code, and it consists of a

- Preamble field (P-field) that gives information about:
  - time code ID to define Agency-defined epoch (Level 3 or 4)
  - number of octets used for the time.
- Time field (T-field) that consists of:
  - time: time amount elapsed from given reference epoch
- Time Increment

- the Time Increment step as a decimal log multiple of a second (e.g. '-3' = millisecond, '0' = second, etc.)

This time code is not UTC based, and is relative with respect of the given reference epoch.

This Time code can be defined with the following DFDL element:

```
<xs:complexType name="TypeTimeCode_A_DEF_48_IDXXX" dmx:ccsdsType="A_DEF" dmx:timeIncrement="-3"
dmx:preambleIncuded="false">
  <xs:sequence>
    <xs:element name="Counter" type="xs:int" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bits" dfdl:length="48" dmx:representation="UInteger32"/>
  </xs:sequence>
</xs:complexType>
```



## 7. EXPRESSION LANGUAGE

The DFDL expression language allows the processing of values conforming to the data model defined in the DFDL Infoset. It allows properties in the DFDL schema to be dependent on the value of an occurrence of an element or the value of a DFDL variable. For example the length of the content of an element can be made dependent on the value of another element in the document.

The main uses of the expression language, in context of DFDL4S, are as follows<sup>2</sup>:

1. When a DFDL property needs to be set dynamically at parse time from the value of one or more elements of the data. That is the case of `length`.
2. In a `dfdl:discriminator` annotation to resolve uncertainty when parsing.

The DFDL expression language is a subset of XPath 2.0 [RD.4]. DFDL uses a subset of XML schema and has a simpler information model, so only a subset of XPath 2.0 expressions is meaningful in DFDL Schemas. For example there are no attributes in DFDL so the attribute axis is not needed. [RD.1]

DFDL4S supports only a subset of DFDL expression language, with some especial extensions, whose need was identified in the context of S2G DataViewer.

DFDL expressions follow the XPath 2.0 syntax rules but are always enclosed in curly braces "{" and "}".

The expression language supported by DFDL4S is given by the grammar of describes which of them are implemented - completely or partially - in the DFDL4S library. Column "DFDL4S Compliance" contains values: "C" – implemented in DFDL4S; "PC" – partially implemented in DFDL4S; "NC" – not implemented in DFDL4S and; "N/A" - not available in DFDL4S (DFDL supports both binary and text data, but due to the intended use of DFDL4S the support for text data has not been considered and is not covered by the current implementation; hence N/A is the classification given to properties related only to text data encoding).

In the scope of DFDL4S an element in the information set (also applicable to simple type definition) can be assigned with the attributes specified in Table 18. In the following description the concept of 'DFDL expression' is used. This corresponds to an expression defined by a path to access a given element in the data schema. DFDL expressions are not limited to the defining type. Instead they can be used to traverse the schema and reach any element. This can be done thru the use of the following path elements:

- ".": the element itself;
- "..": the parent element;
- "\*": the element's first child;
- "\*<name>": the elements child matching the given <name>.

Table 18 DFDL core properties compatibility <sup>3</sup>.

---

<sup>2</sup> In general, DFDL expressions are used in more situations (see [RD.1]).

<sup>3</sup> Refer to [RD.4] for a description of XPath 2.0 and to [RD.3] for name terminal symbols.

```
DFDL4SExpression ::= "{" Expr "}"
Expr ::= ExprSingle
ExprSingle ::= OrExpr
OrExpr ::= AndExpr ("or" AndExpr)*
AndExpr ::= ExtendedComparisionExpr( "and" ExtendedComparisionExpr )*
ExtendedComparisionExpr := ComparisionExpr | InExpr | InRangeExpr
InExpr ::= AdditiveExpr "in" "[" NumericalLiteral ("," NumericalLiteral "]" )? "]"
InRangeExpr ::= AdditiveExpr "in" "[" NumericalLiteral "," NumericalLiteral "]"
ComparisionExpr ::= AdditiveExpr (ValueComp AdditiveExpr)?
AdditiveExpr ::= UnaryExpr ("+" | "-" | " * " ) UnaryExpr)*
UnaryExpr ::= ("-"|"+"|" * ")? ValueExpr
ValueExpr ::= PathExpr
ValueComp ::= "eq"
PathExpr ::= ("/" RelativePathExpr?) | RelativePathExpr | FilterExpr
RelativePathExpr ::= StepExpr ("/" StepExpr)*
StepExpr ::= AxisStep
AxisStep ::= (ReverseStep | ForwardStep)
ForwardStep ::= AbbrevForwardStep
AbbrevForwardStep ::= NodeTest | ContextItemExpr
ReverseStep ::= AbbrevReverseStep
AbbrevReverseStep ::= ".."
NodeTest ::= NameTest | Limiter
Limiter ::= "#"
NameTest ::= QName | WildCard | NameRegex
WildCard ::= "*"
NameRegex ::= StartNameChar ((NameChar)* "*" (NameChar)*)+
FilterExpr ::= PrimaryExpr
PrimaryExpr ::= Literal | ContextItemExpr | FunctionCall
Literal ::= NumericLiteral | StringLiteral
NumericLiteral ::= IntegerLiteral
ContextItemExpr ::= "."
FunctionCall ::= QName "(" (ExprSingle("," ExprSingle)*)? ")"
```

**Figure 12: DFDL4S Expression Language**

Note some important differences between the language specified by DFDL standard and the supported by DFDL4S. Here *if*, *predicate*, *forward axis*, *reverse axis*, *variable reference*, *parenthesised*, *decimal*, *double* expressions are not supported. Moreover, none of the comparison operators *ne*, *lt*, *le*, *gt* and *ge* are supported.

On the other hand DFDL4S supports some extensions. Comparison expressions are extended with the *in* and *in range* expressions. *Path* expressions are extended with *limiters*, *wildcards* and *name regular expressions*.

An *in range* expression is a boolean expression with the form

*numeric* **inrange** [*min*, *max*]

where *numeric* is an expression that evaluates to integer and *min* and *max* are integer literals, and evaluates to *true* if and only if *numeric* evaluates to an integer between *min* and *max* (limits included).

An *in* expression is a boolean expression with the form

*numeric* **in** [*e*<sub>1</sub>, *e*<sub>2</sub>, ..., *e*<sub>*n*</sub>]

where *numeric* is an expression that evaluates to integer and *e*<sub>1</sub> to *e*<sub>*n*</sub> are integer literals, and evaluates to *true* if and only if *numeric* evaluates to one of *e*<sub>1</sub>, *e*<sub>2</sub>, ..., or *e*<sub>*n*</sub>.

A *# node test* is true for the context node if it contains 2 or less elements.

A *\* node test* is true for the first, in document order, element contained in the context node.<sup>4</sup>

A *name regular expression node test* `regex` is true for the first, in the document order, element contained in the context node whose QName match the regular expression `regex`.

With respect to functions:

- `dfdl:contentLength($node,$lengthUnits)`, where *\$node* is a *path* expression (with the described DFDL4S extensions) and *\$lengthUnits* is 'bits' or 'bytes', is supported. This function evaluates to the length of the element given by the evaluation of *\$node*.
- `dfdl:checkRangeInclusive($node,$val1,$val2)`, where *\$node* is a *path* expression (with the described DFDL4S extensions) and *\$val1* and *\$val2*, is supported. This Boolean function evaluates if value given by the evaluation of *\$node* is contained in the numerical set defined between *\$val1* and *\$val2*.

The `dfdl:checkRangeInclusive($node,$val1,$val2)` function covers also the semantics of the *in/inrange* operator:

- To cover semantic of *inrange*:

```
../Node inrange \[0,10]
```

Use

```
{dfdl:checkRangeInclusive(..Node,0,10)}
```

- To cover semantic of *in*:

---

<sup>4</sup> In Xpath 2.0 standard, contrary to DFDL, wildcards are allowed. Although they are also supported in DFDL4S, they do not match all nodes contained in the context node, but only the first one. This modification is needed since DFDL expression must always evaluate to sequences with 0 or 1 items.

```
../Node in \[0,1,3,4]
```

#### Use

```
{dfdl:checkRangeInclusive(../Node,0,1) or  
dfdl:checkRangeInclusive(../Node,3,4)}
```

DFDL4S accepts expressions in the described language as value of the DFDL properties `test`, in *discriminators*, and `length`; and DMX property `assertExpression`. The value of a `test` or `assertExpression` property must evaluate to a *boolean*, being in the language defined by `OrExpr`. The value of a `length` property must evaluate to a positive *integer*, being in the language defined by `AdditiveExpr`.

DFDL4S accepts explicit declaration of all default values in a `dfdl:format` construct. By defining a default value this way, all elements will share this default value and override it as needed.

As an extension of `dfdl:format`, DFDL4S also accepts declaration of `dfdl:defineFormat` with a name (declared only once, in dedicated schema file).

See below example for `dfdl:defineFormat`.

On `main.xsd`:

```
<xs:include schemaLocation="xsd/built-in-formats.xsd"/>  
  
<xs:annotation>  
  <xs:documentation/>  
  <xs:appinfo source="http://www.ogf.org/dfdl/">  
    <dfdl:format ref="myFormat"/>  
    <dfdl:format x="a"/>  
  </xs:appinfo>  
</xs:annotation>
```

On `xsd/built-in-formats.xsd`:

```
<xs:annotation>  
  <xs:appinfo source="http://www.ogf.org/dfdl/">  
  
    <dfdl:defineFormat name="myFormat">  
  
      <dfdl:format x="b" y="a" ref="myFormatExt"/>  
    </dfdl:defineFormat>  
  
    <dfdl:defineFormat name="myFormatExt">  
      <dfdl:format y="b"/>  
    </dfdl:defineFormat>  
  
  </xs:appinfo>  
</xs:annotation>
```

After loading the `main.xsd` schema, the following default property/value would be defined as below:

```
dfdl:x="a"  
dfdl:y="b"
```

But one could still override the default properties defined by the `dfdl:format`, by setting them in the element attributes as below:

```
<xs:element name="TypeX" type="xs:int" dfdl:x="c"/>
```





## APPENDIX A - COMPATIBILITY WITH DFDL CORE SET

This section describes the compliance of DFDL4S implementation with respect to DFDL standard specification. Regarding the DFDL simple types depicted in Figure 1 (see Section 5. of [RD.1]), all of them are supported.

### DFDL built-in types

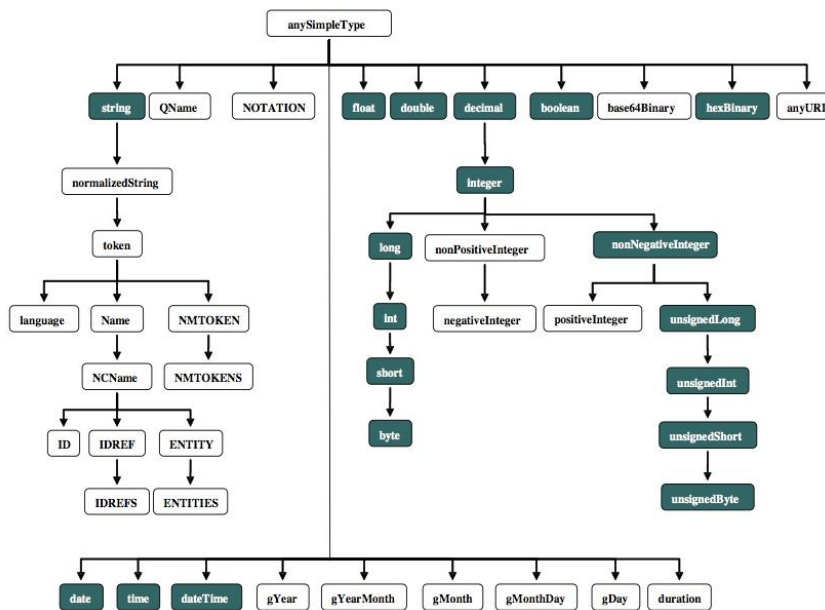


Figure 13: DFDL Simple Types

For every DFDL numeric simple type DFDL4S validates the associated explicit length restrictions, specified for base-2 binary number elements listed in Table 17 (see Section. 12.3.7.2.1 of [RD.1]).

Table 17 Allowable Specified Lengths in Bits for Base-2 Binary Number Elements

Type	Minimum value of length	Maximum value of length
xs:byte	2	8
xs:short	2	16
xs:int	2	32
xs:long	2	64
xs:unsignedByte	1	8
xs:unsignedShort	1	16
xs:unsignedInt	1	32
xs:unsignedLong	1	64
xs:nonNegativeInteger	1	Implementation-dependent (but not less than 64)
xs:integer	2	Implementation-dependent (but not less than 64)
xs:decimal	2	Implementation-dependent (but not less than 64)

With respect to the DFDL core properties, the Table 18 describes which of them are implemented - completely or partially - in the DFDL4S library. Column “DFDL4S Compliance” contains values: “C” – implemented in DFDL4S; “PC” – partially implemented in DFDL4S; “NC” – not implemented in DFDL4S and; “N/A” - not available in DFDL4S (DFDL supports both binary and text data, but due to the intended use of DFDL4S the support for text data has not been considered and is not covered by the current implementation; hence N/A is the classification given to properties related only to text data encoding).

In the scope of DSDL4S an element in the information set (also applicable to simple type definition) can be assigned with the attributes specified in Table 18. In the following description the concept of ‘DFDL expression’ is used. This corresponds to an expression defined by a path to access a given element in the data schema. DFDL expressions are not limited to the defining type. Instead they can be used to traverse the schema and reach any element. This can be done thru the use of the following path elements:

- ".": the element itself;
- "..": the parent element;
- "\*": the element's first child;
- "\*<name>": the elements child matching the given <name>.

**Table 18 DFDL core properties compatibility**

Property name	Short Description	DFDL4S Compliance	Supported values
<i>Properties Common to both Content and Framing</i>			
byteOrder	This property applies to all types with representation binary. Valid values 'bigEndian', 'littleEndian'.	C	bigEndian littleEndian
bitOrder	The bit order is the correspondence of a bit's numeric significance to the bit position (1 to 8) within the byte. Valid values 'mostSignificantBitFirst', 'leastSignificantBitFirst'.	PC	mostSignificant BitFirst
encoding	Values are one of: IANA charset name; CCSID; DFDL standard encoding name; implementation-specific encoding name.	PC	UTF-8 UTF-16 ISO-8859-1
utf16Width	Specifies whether the encoding 'UTF-16' should be treated as a fixed or variable width encoding.	PC	variable



Property name	Short Description	DFDL4S Compliance	Supported values
ignoreCase	Whether mixed case data is accepted when matching delimiters and data values on input.	NC	
encodingErrorPolicy	This property provides control of how decoding and encoding errors are handled when converting the data to text, or text to data.	PC	error
<i>Common Framing, Position, and Length</i>			
alignment	A non-negative number that gives the alignment required for the beginning of the item	PC	implicit
alignmentUnits	Scales the alignment so alignment can be specified in either units of bits or units of bytes.	NC	
fillByte	Used on unparsing to fill empty space such as between two aligned elements.	PC	0
leadingSkip	A non-negative number of bytes or bits to skip before alignment is applied.	PC	0
trailingSkip	A non-negative number of bytes or bits to skip after the element, but before considering the alignment of the next element.	PC	0
initiator	Specifies a whitespace separated list of alternative literal strings one of which marks the beginning of the element or group of elements.	PC	""

Property name	Short Description	DFDL4S Compliance	Supported values
terminator	Specifies a whitespace separated list of alternative text strings that one of which marks the end of an element or group of elements.	PC	""
emptyValueDelimiterPolicy	Indicates that when an element in the data stream is empty, an initiator (if one is defined), a terminator (if one is defined), both an initiator and a terminator (if defined) or neither must be present. Valid values are 'none', 'initiator', 'terminator' or 'both'	NC	
documentFinalTerminator CanBeMissing	When this property is true, then when an element is the last element in the data stream, then on parsing, it is not an error if the terminator is not found. Valid values are 'yes', 'no'.	NC	
lengthKind	Controls how the content length of the component is determined. Valid values are: 'explicit', 'delimited', 'prefixed', 'implicit', 'pattern', 'endOfParent'	PC	explicit
lengthUnits	Specifies the units to be used whenever a length is being used to extract or write data. Applicable when dfdl:lengthKind is 'explicit', 'implicit' (for xs:string and xs:hexBinary) or 'prefixed'. Valid values 'bytes', 'characters', 'bits'.	PC	bits bytes

Property name	Short Description	DFDL4S Compliance	Supported values
length	Specifies the length of this element in units that are specified by the <code>dfdl:lengthUnits</code> property. This property can be computed by way of an expression which returns a non-negative integer. The expression must not contain forward references to elements which have not yet been processed. Only used when <code>lengthKind</code> is 'explicit'.	C	Non-negative Integer or DFDL Expression
<code>prefixIncludesPrefixLength</code>	Whether the length given by a prefix includes the length of the prefix as well as the length of the content region. Valid values are 'yes', 'no'.	NC	
<code>prefixLengthType</code>	This type specifies the representation of the length prefix, which is in the <code>PrefixLength</code> region.	NC	
<code>lengthPattern</code>	Specifies a regular expression that, on parsing, is executed against the datastream to determine the length of the element. Only used when <code>lengthKind</code> is 'pattern'.	NC	
<i>Simple Type Content</i>			
<code>representation</code>	The permitted representation properties for each logical type. Valid values are 'text' and 'binary' and are dependent on logical type.	PC	binary

Property name	Short Description	DFDL4S Compliance	Supported values
textPadKind	Indicates whether to pad the data value on unparsing.	N/A	
textTrimKind	Indicates whether to trim data on parsing.	N/A	
textOutputMinLength	Specifies the minimum content length during unparsing for simple types	N/A	
escapeSchemeRef	A named, reusable, escape scheme is used by referring to its name from a <code>dfdl:escapeSchemeRef</code> property on an element.	PC	""
escapeKind	The type of escape mechanism defined in the escape scheme. Valid values 'escapeCharacter', 'escapeBlock'.	N/A	
escapeCharacter	DFDL String Literal or DFDL Expression. Specifies one character that escapes the subsequent character.	N/A	
escapeBlockStart	The string of characters that denotes the beginning of a sequence of characters escaped by a pair of escape strings.	N/A	
escapeBlockEnd	The string of characters that denotes the end of a sequence of characters escaped by a pair of escape strings.	N/A	
escapeEscapeCharacter	Specifies one character that escapes an immediately following <code>dfdl:escapeCharacter</code> or first character of <code>dfdl:escapeBlockEnd</code> .	N/A	

Property name	Short Description	DFDL4S Compliance	Supported values
<code>extraEscapedCharacters</code>	A whitespace separated list of single characters that must be escaped in addition to the in-scope delimiters	N/A	
<code>generateEscapeBlock</code>	Controls when escaping is used on unparsing. Valid values 'always', 'whenNeeded'.	N/A	
<code>textBidi</code>	Indicates the text content of the element is bidirectional.	N/A	
<code>textBidiOrdering</code>	Defines how bidirectional text is stored in memory.	N/A	
<code>textBidiOrientation</code>	Indicates how the text should be displayed.	N/A	
<code>textBidiSymmetric</code>	Defines whether characters such as < ( [ { that have a symmetric character with an opposite directional meaning: > ) ] } should be swapped	N/A	
<code>textBidiShaped</code>	Defines whether characters should be shaped on unparsing.	N/A	
<code>textBidiNumeralShapes</code>	Defines on unparsing whether logical numbers with text representation should have Arabic shapes.	N/A	
<code>textStringJustification</code>	Valid values 'left', 'right', 'center'	N/A	
<code>textStringPadCharacter</code>	The value that is used when padding or trimming string elements.	N/A	
<code>truncateSpecifiedLengthString</code>	Used on unparsing only	N/A	
<code>decimalSigned</code>	Indicates whether an xs:decimal element is signed.	N/A	

Property name	Short Description	DFDL4S Compliance	Supported values
textNumberRep	Valid values are 'standard', 'zoned'	N/A	
textNumberJustification	Controls how the data is padded or trimmed on parsing and unparsing.	N/A	
textNumberPadCharacter	The value that is used when padding or trimming number elements.	N/A	
textNumberPattern	Defines the ICU-like pattern that describes the format of the text number.	N/A	
textNumberRounding	Specifies how rounding is controlled during unparsing.	N/A	
textNumberRoundingMode	Specifies how rounding occurs during unparsing.	N/A	
textNumberRoundingIncrement	Specifies the rounding increment to use during unparsing.	N/A	
textNumberCheckPolicy	Indicates how lenient to be when parsing against the pattern.	N/A	
textStandardDecimalSeparator	Defines the whitespace separated list of single characters that will appear (individually) in the data as the decimal separator.	N/A	
textStandardGroupingSeparator	Defines the single character that will appear in the data as the grouping separator.	N/A	
textStandardExponentRep	Defines the actual character(s) that will appear in the data as the exponent indicator.	N/A	
textStandardInfinityRep	The value used to represent infinity.	N/A	

Property name	Short Description	DFDL4S Compliance	Supported values
<code>textStandardNaNRep</code>	The value used to represent NaN.	N/A	
<code>textStandardZeroRep</code>	The whitespace separated list of alternative literal strings that are equivalent to zero.	N/A	
<code>textStandardBase</code>	Indicates the number base.	N/A	
<code>textZonedSignStyle</code>	Specifies the code points that are used to overpunch the sign nibble	N/A	
<code>binaryNumberRep</code>	Allowable values for each number type.	PC	binary
<code>binaryDecimalVirtualPoint</code>	An integer that represents the position of an implied decimal point within a number	PC	0
<code>binaryPackedSignCodes</code>	A whitespace separated string giving the hex sign nibbles to use for a positive value, a negative value, an unsigned value, and zero.	NC	
<code>binaryNumberCheckPolicy</code>	Indicates how lenient to be when parsing binary numbers.	NC	
<code>binaryFloatRep</code>	This specifies the encoding method for the float and double.	PC	ieee
<code>textBooleanTrueRep</code>	A whitespace separated list of representations to be used for 'true'.	N/A	
<code>textBooleanFalseRep</code>	A whitespace separated list of representations to be used for 'false'.	N/A	
<code>textBooleanJustification</code>	Controls how the data is padded or trimmed on parsing and unpadding.	N/A	

Property name	Short Description	DFDL4S Compliance	Supported values
<code>textBooleanPadCharacter</code>	The value that is used when padding or trimming boolean elements.	N/A	
<code>binaryBooleanTrueRep</code>	This value gives the representation to be used for 'true'	PC	1
<code>binaryBooleanFalseRep</code>	This value gives the representation to be used for 'false'	PC	0
<code>calendarPattern</code>	Defines the ICU pattern that describes the format of the calendar.	NC	
<code>calendarPatternKind</code>	Valid values 'explicit', 'implicit'	NC	
<code>calendarCheckPolicy</code>	Indicates how lenient to be when parsing against the pattern.	NC	
<code>calendarTimeZone</code>	This property provides the time zone that will be assumed if no time zone explicitly occurs in the data.	NC	
<code>calendarObserveDST</code>	Whether the time zone given in <code>dfdl:calendarTimeZone</code> observes daylight savings time.	NC	
<code>calendarFirstDayOfWeek</code>	The day of the week upon which a new week is considered to start.	NC	
<code>calendarDaysInFirstWeek</code>	Specify the number of days of the new year that must fall within the first week.	NC	
<code>calendarCenturyStart</code>	This property determines on parsing how two-digit years are interpreted.	NC	



Property name	Short Description	DFDL4S Compliance	Supported values
calendarLanguage	The language that is used when the pattern produces a presentation in text.	NC	
textCalendarJustification	Controls how the data is padded or trimmed on parsing and unparsing.	N/A	
textCalendarPadCharacter	The value that is used when padding or trimming calendar elements.	N/A	
binaryCalendarRep	Categorization of the encoding used for dates.	NC	
binaryCalendarEpoch	The epoch from which to calculate dates and times.	NC	
nilKind	Used when XSDL nillable is 'true'	N/A	
nilValue	Specifies the text strings that are the possible literal or logical nil values of the element.	N/A	
nilValueDelimiterPolicy	Indicates that when the value nil is represented, an initiator (if one is defined), a terminator (if one is defined), both an initiator and a terminator (if defined) or neither must be present.	N/A	
useNilForDefault	Valid values are 'yes', 'no'	N/A	
<i>Sequence Groups</i>			
sequenceKind	Valid values are 'ordered', 'unordered'	PC	ordered
initiatedContent	Valid values are 'yes', 'no'	PC	no
separator	Specifies a whitespace separated list of alternative literal strings that are the possible separators for the sequence.	PC	""

Property name	Short Description	DFDL4S Compliance	Supported values
<code>separatorPosition</code>	Valid values 'infix', 'prefix', 'postfix'	NC	
<code>separatorSuppressionPolicy</code>	Controls the circumstances when separators are expected in the data when parsing, or generated when unparsing, if an optional element occurrence or a group has a zero-length representation.	NC	
<code>floating</code>	Whether the occurrences of an element in an ordered sequence can appear out-of-order in the representation.	PC	no
<code>hiddenGroupRef</code>	Elements within this model group will not be added to the Infoset, and are called hidden elements.	NC	
<code>discriminator</code>	This property allows the resolution of a point of uncertainty by choosing the specific alternatives based on the evaluation of a boolean expression denominated <i>test</i> . The <i>test</i> attribute of <code>dfdl:discriminator</code> defines an expression that is evaluated as boolean. If it succeeds the element where the discriminator is defined is taken as part of the DFDL model.	PC (only a subset of DFDL expression language is supported)	DFDL Expression
Choice Groups			

Property name	Short Description	DFDL4S Compliance	Supported values
choiceLengthKind	Valid values are 'implicit' and 'explicit'.  'implicit' means the branches of the choice are not filled, so the ChoiceContent region is variable length depending on which branch appears.  'explicit' means that the branches of the choice are always filled to the fixed length specified by dfdl:choiceLength, so the ChoiceContent region is fixed length regardless of which branch appears.	C	implicit
choiceLength	Specifies the length of the choice in bytes.	NC	
initiatedContent	When 'yes' indicates that all the branches of the choice are initiated.	NC	
choiceDispatchKey	A DFDL Expression discriminating one of the branches of a choice. The parser then goes straight to that branch, ignoring consideration of any other choice branches.	NC	
choiceBranchKey	This literal provides an alternate way to discriminate a choice to a branch.	NC	
<i>Array elements and optional elements</i>			
occursCountKind	Specifies how the actual number of occurrences is to be established. Valid values 'fixed', 'expression', 'parsed', 'implicit' and 'stopValue'.	PC	fixed expression

Property name	Short Description	DFDL4S Compliance	Supported values
occursCount	Specifies the number of occurrences of the element.	NC	
occursStopValue	A whitespace separated list of logical values that specify the alternative logical stop values for the element.	NC	
<i>Calculated Values</i>			
inputValueCalc	An expression that calculates the value of the element when parsing.	NC	
outputValueCalc	An expression that calculates the value of the current element when unparsing.	NC	

End of Document