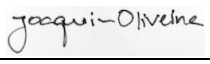
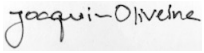
DFDL4S library

Developer's Manual

Code : S2G-DME-TEC-SUM078
Issue : 1.F
Date : 19/09/2017

	Name	Function	Signature
Prepared by	Joaquim Oliveira	Project Manager	
Reviewed by	Antonio Gutiérrez	Technical Consultant	
Approved by	Joaquim Oliveira	Project Manager	
Signatures and approvals on original			

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

This page intentionally left blank

Document Information

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

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

External Distribution		
Name	Organisation	Copies
Free for distribution	-	-

Archiving	
Word Processor:	MS Word 2000
File Name:	S2G-DME-TEC-SUM078-1F.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: 1. Changed Figure 4-3, 4-4 and 4-5 to Figure 3, 4 and 5 2. Updated Figure 3 with updated diagram 3. Reordered Table 4 DFDLLib methods by name 4. Updated contents of Table 4 DFDLLib: a. Changed inputs names: appendElements(1), appendElements(2), initLib, b. Changed description: createDocument(1) c. Added missing method name, input, ouput and description: createDocument(2), getSchemaDefinition, interpretDocument(2), main 5. Updated Figure 4 with updated diagram 6. Updated contents of Table 5 Document: a. Added missing method name, input, ouput and description: childAdd, evaluate	19/09/2017	Chapter 4 Figure 3 Table 4 Table 4 Figure 4 Table 5

	b. Changed inputs names: childAt(1), childAt(2), childAtOffset, childCount, elementContains c. Corrected input, ouput and description: close d. Removed method: getElement 7. Split contents of Table 6 into logical groups and by updated its operations: a. Added new methods: getHexadecimalValue, getIntrisicType, getRangeMaximum, getRangeMinimum, getValueBinary, getValueChar8, getValueFloat32, getValueFloat64, getValueInteger, getValueTime, setData(1), setData(2), setRawData(1), setRawData(2), setValueFloat32, setValueFloat64, setValueInteger, setValueTime, value(3) b. Changed inputs names: absoluteName, name, offset, path, retrieveCleanAlignedData, retrieveCleanData, retrieveRawData(1), retrieveRawData(2), size, type, value(1) 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.		Table 6 Chapter 4.4

Table of Contents

List of Tables

List of Figure

1. INTRODUCTION

The Space-to-Ground Data Viewer application (S2G) (available at http:/eop-cfi.esa.int/) allows to inspect the contents of the binary telemetry files formatted according to CCSDS Standards.

The DFDL for Space (DFDL4S) described in this Developer Manual is the underlying software library that allows to read, parse, interpret, update and create binary telemetry files formatted according to CCSDS Standards (e.g. CADU, TF or ISP data files) by using DFDL schemas [RD.1].

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

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]	CCSDS	CCSDS Standards available at https://public.ccsds.org/	
[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.

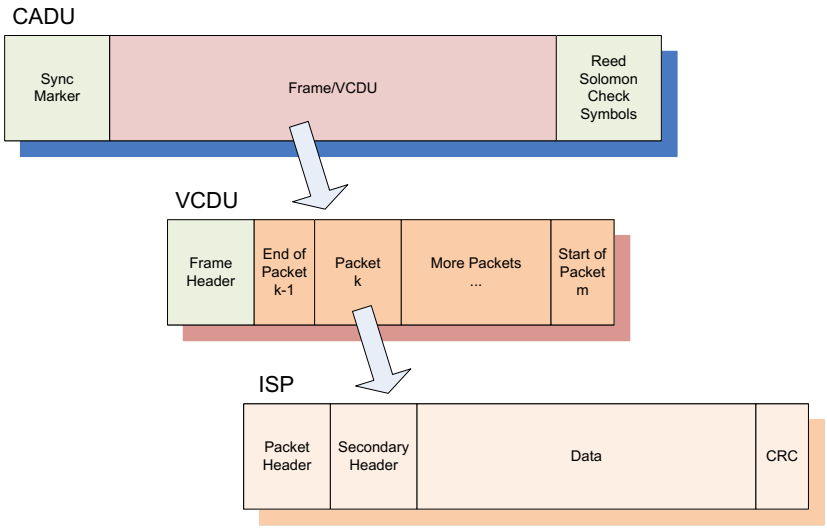


Figure 1: Hierarchy of Data received by the Ground Sensor Stations

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 3. The platforms presented have been used to support testing activities.

Table 3: Minimum System Requirements

Platform	Requirements	
Linux (64 bit)	RAM:	2 GB
	Disk Space:	50 MB
	Dependencies:	Java 1.8
Mac OS (64 bit)	RAM:	2 GB
	Disk Space:	50 MB
	Dependencies:	Java 1.8
Windows (32/64 bit)	RAM:	1 GB
	Disk Space:	50 MB
	Dependencies:	Java 1.8

3.3. Installation

To install DFDL4S library simply unzip the distribution archive (dfdl4s_release_X_X_X.zip) 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;
- DFDL4S/installation_tests: Standard tests exercising basic functionalities (parse, interpret, update and create data) which can be used to validate the proper installation of the library:
  - testRead.sh[1]: script to invoke the DFDL4S reading capabilities;
  - testPrint.sh: script to invoke the DFDL4S library print example;
  - testPerformace.sh: script to assess the DFDL4S library reading performance;
  - testUpdate.sh: script to invoke the DFDL4S library write/update example
  - testCreate.sh: script to invoke the DFDL4S library data creation capabilities.

[1] Note that both .bat (for Windows) and .sh (for Linux and MacOS) versions of the scripts are available.

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[2]. 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;

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

Utilities: a set of classes that provides additional support to the basic functionality.

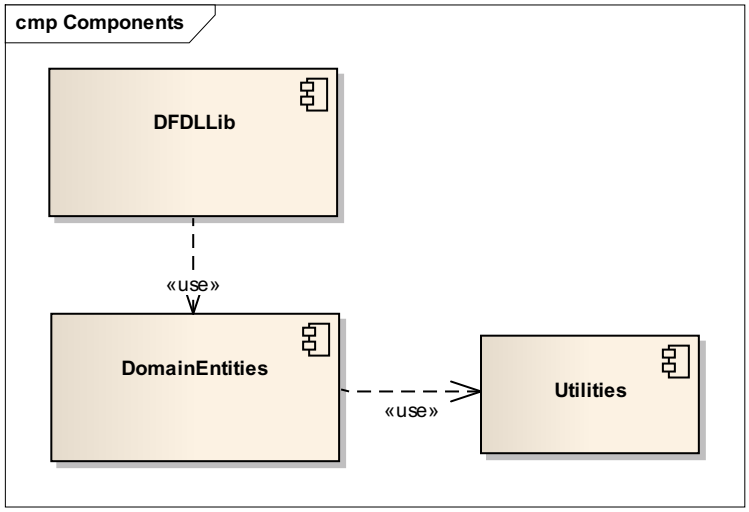


Figure 2: DFDL4S library high-level package diagram

[2] 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.

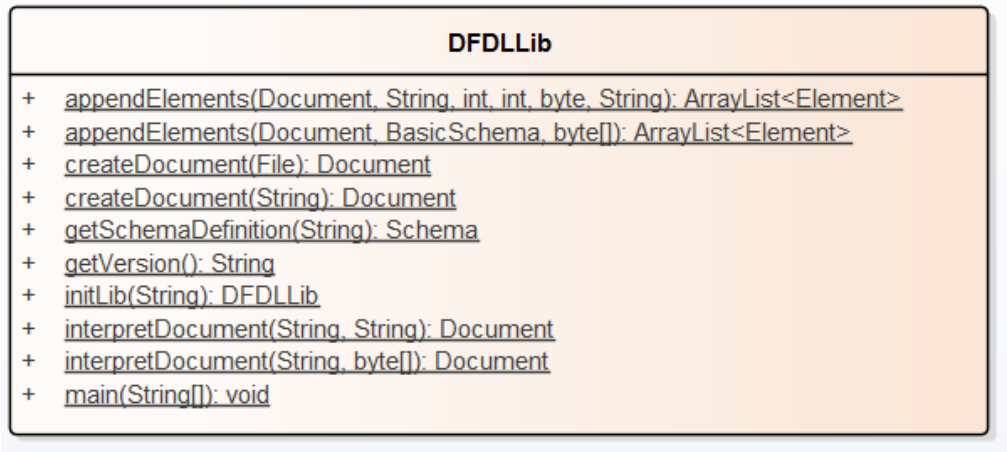


Figure 3: DFDLLib class diagram

Table 4: List of operations of the DFDLLib class

Operation name	Input	Output	Description
`appendElements`	Document String int int byte String	List<Element>	This method adds new elements to a document based on data generation parameters
`appendElements`	Document BasicSchema byte[]	List<Element>	This method adds new elements to a document based on raw data.
`createDocument`	File	Document	This method generates a Document supported by a given file object.

Operation name	Input	Output	Description
createDocument	String	Document	This method generates a Document supported by a given file path.
getSchemaDefinition	String	Schema	Access schema definition from a given file
getVersion	-	String	The version number and release date of the library.
initLib	String	DFDLLib	This method initialises the DFDLLib: sets the UTC TAI conversion data.
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.
main	String[]	-	The class' main: prints the lib version

4.2.2. DomainEntities

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

Figure 4 shows class diagram of the DomainEntities package.

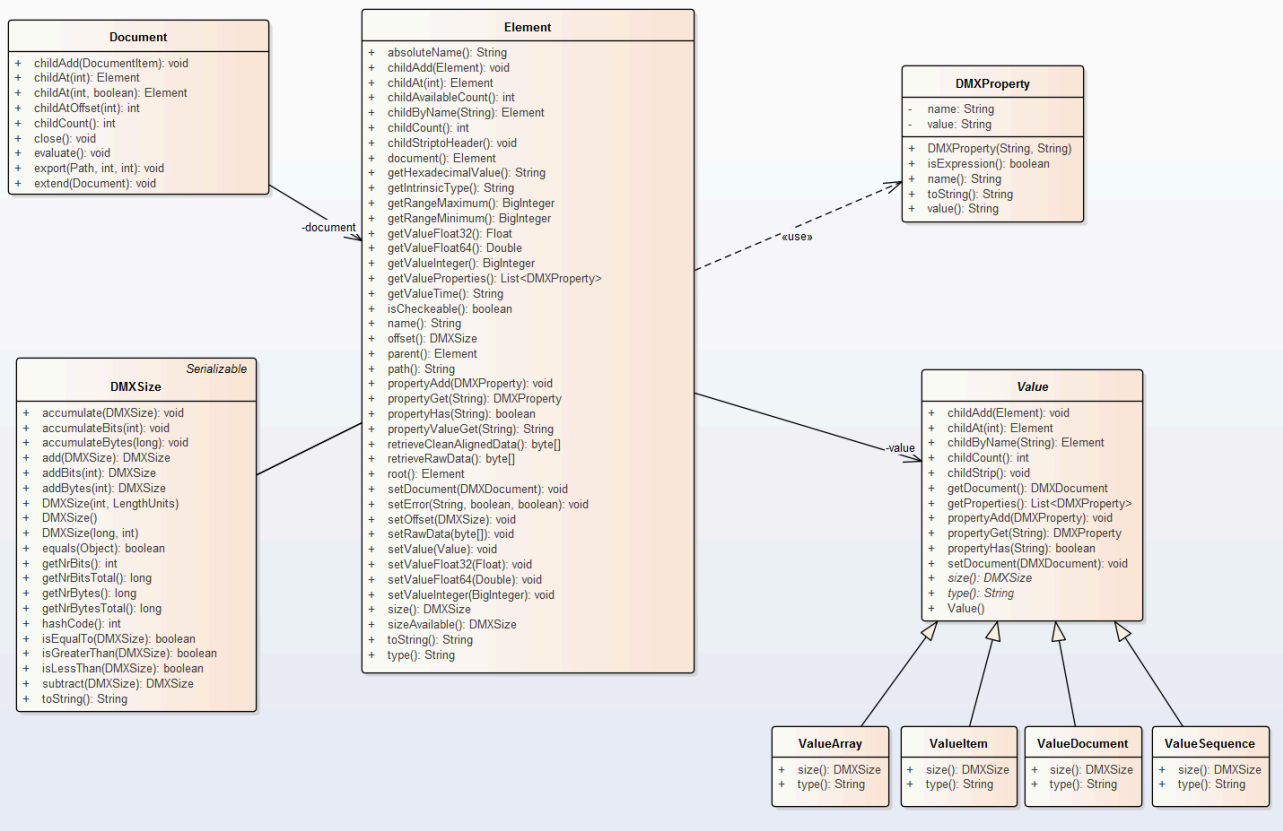


Figure 4: DomainEntities package class diagram

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;
- *Value:* the Value class represents the internals of a domain model element (storing its properties, such as size and representation, and children);

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 5.

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

Table 5: List of operations of the Document class

Operation name	Input	Output	Description
`childAdd`	DocumentItem	-	Add a new child to the set of existing children
`childAt`	int	Element	Access the child at the given index (does not evaluate hooks). Returns the requested child element.
`childAt`	int boolean	Element	Access the child at the given index. Returns the requested child element.
`childAtOffset`	int	int	Access the child at the given file offset. Returns the requested child element index.
`childCount`	-	int	Returns the number of children of the element.
`close`	-	-	Close the document, releasing all associated resources.
`elementContains`	Element long	boolean	Verifies that the element contains the offset. Returns true if element contains the offset; false otherwise.
`evaluate`	Path int int	-	Evaluates all items in the document This method is used for debug purposes, to force the load of every packet into memory.
`extend`	Document	-	Extends the contents of a document based on the contents of another document.

4.2.2.2. *Element* class

The Element class provides:

- Methods to get the value of an element (see Table 6);
- Methods to set the value of an element (see Table 7);
- Methods to get the properties of an element (see Table 8);
- Method to navigate in the data structure (see Table 9);
- Methods implementing application specific functionalities (see Table 10 and Table 11). Note: these methods are deprecated and may be removed. Their usage is not recommended;

Table 6: Element class: List of methods for getting the value of an element

Method name	Input	Output	Description
`getValueFloat32`	-	Float	returns the element value (element shall be of type 32-bit floating point, otherwise an exception is raised)
`getValueFloat64`	-	Double	returns the element value (element shall be of type 64-bit floating point, otherwise an exception is raised)
`getValueInteger`	-	BigInteger	returns the element value (element shall be of type integer, otherwise an exception is raised)
`getValueTime`	-	String	returns the element value (element shall be of type Time, otherwise an exception is raised)
`retrieveCleanAlignedData`	-	Bytes[]	returns the value as an array of bytes. The first (last) byte contains the most (least) significant bits of the element. Not used bits are set to zero.
`getHexadecimalValue`	-	String	returns the element value as an hexadecimal string
`value`	-	String	returns the value of the element as a string

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

Method name	Input	Output	Description
`setValueFloat32`	Float	-	sets the element value (element shall be of type 32-bit floating point, otherwise an exception is raised)
`setValueFloat64`	Double	-	sets the element value (element shall be of type 64-bit floating point, otherwise an exception is raised)
`setValueInteger`	BigInteger	-	sets the element value (element shall be of type integer, otherwise an exception is raised)
`setValueTime`	String	-	sets the element value (element shall be of type Time, otherwise an exception is raised)
`setRawData`	byte[]	-	set the value as an array of bytes. The first (last) byte shall contain the most (least) significant bits of the element. Not used bits shall be set to zero.
`setValue`	Value	void	Setup a new Element Value property (see Value class)

Table 8: Element class: List of methods for getting properties of an element

Operation name	Input	Output	Description
absoluteName	-	String	Access the absolute name of the 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
name	-	String	Access the name of the element
offset	-	DMXSize	Access the offset of the element, according to the offset type
path	-	String	Access the path of the element, starting at the packet level (document level is not considered)
size	-	DMXSize	Access the size of the value represented
type	-	String	Access the type of the element
isCheckeable	void	boolean	Indicates whether this element is checkable from the point of view of Error Control (CRC or RS fields).
propertyAt	index	DMXProperty	Access the property at the given index
propertyCount	void	int	Access the number of properties available
propertyGet	String	DMXProperty	Access the property with a given name
propertyHas	String	boolean	Verify that a property with a given name exists
propertyValueGet	String	String	Access the value of a property with a given name

Operation name	Input	Output	Description
toString	void	String	Access the string representation of the element

Table 9: Element class: List of methods for navigation

Operation name	Input	Output	Description
childAt	index	Element	Access the child at the given index
childByName	String	Element	Access the child with a given name. Returns the requested child element; null if not available.
childCount	void	int	Access the number of children of the element
document	void	Element	Access the top element (document) that contains this element.
parent	void	Element	Access the parent of the element
root	void	Element	Access the packet element (below document) that contains this element
sizeAvailable	void	DMXSize	Access the actual size of data available in file

Table 10: Element class: List of methods for error handling (deprecated)

Operation name	Input	Output	Description
countErrors	propagateableOnly: boolean, excludeConcealed: boolean	int	Access the number of errors (including children errors) associated with this item.
getChildErrors	void	List ErrorIndicator	Access the list of all child errors
getChildErrors	filter	List ErrorIndicator	Access the list of child errors (optionally filtering only the errors to be propagated).
getError	void	ErrorIndicator	Access error indicator related to this element
hasError	void	boolean	Indicates the presence of an error in the item
hasErrors	propagateableOnly excludeConcealed	boolean	Indicates the presence of an error in the item or any of its children
hasSevereError	void	boolean	Indicates the presence of a severe error in the item or any of its children

Table 11: Element class: List of methods implementing application specific functionalities (deprecated)

Operation name	Input	Output	Description
getValueBinary	-	String	Access the value of the element (according to the 'BINARY' representation)
getValueChar8	-	String	Access the value of the element (according to the 'CHARACTER_8' representation)
retrieveCleanData	-	byte[]	Access the clean (with cleared insignificant bits) data of the element. Notice that data is not word aligned. Leading and trailing bits have been zeroed.

Operation name	Input	Output	Description
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.
setData	String	-	Set the data of the element based on a typed value
setData	String String	-	Set the data of the element based on given representation (see chapter 6 for representations)
setRawData	byte[] int int	-	Update the raw data of the element at a given offset with a given data_size
value	String	String	Access the value of the element (according to the specified representation)
value	String String	String	Access the value of the element (according to the specified representation)

4.2.2.3. Value class

The Value class contains the internals of the domain model element, such as its properties, size, representation, etc. This class contains methods for manipulate those internals.

Table 12: List of operations of the Value class

Operation name	Input	Output	Description
childAt	int	Element	Access the child at the given index.
childByName	String	Element	Access the child with a given name. Returns the requested child element; null if not available.
childCount	void	int	Access the number of children available.
propertyAt	int	DMXProperty	Access the property at the given index.
propertyCount	void	int	Access the number of properties available.

Operation name	Input	Output	Description
propertyGet	String	DMXProperty	Access the property with a given name. Returns the requested property if it exists; null if property not available.
propertyHas	String	boolean	Verify that a property with a given name exists.
size	void	DMXSize	Access the size of the value represented.
type	void	String	Access the type of the value represented.

The following table lists the supported properties used in the definition of a "Value" type.

Table 13: List of properties supported by the Value class

Property name	Short Description	dfdl or dmx
representation	The permitted representation properties for each logical type.	dfdl
byteOrder	This property applies to all types with representation binary. Valid values 'bigEndian', 'littleEndian'.	dfdl
encoding	Values are one of: IANA charset name; CCSID; DFDL standard encoding name; implementation-specific encoding name.	dfdl
concealable	Conceal data if requested and applicable.	dmx
assertExpression	Expression evaluated to check for errors in the data field value.	dmx
assertMessage	Error message associated to a given assert error.	dmx
assertPropagate	Determine whether to propagate an assert error to upper data levels.	dmx
hook	Identifies the evaluation of hooks (such as ISP inside buffer interpretation).	dmx
checkable	Indicates whether an element is checkable from the point of view of Error Control.	dmx

There are several sub-classes to Value each one covering a specific typed value:

- *ValueDocument:* represents the value of a complete binary file;
- *ValueArray:* represents a value that is composed of a set of elements of the same type;
- *ValueSequence:* represents a value that is composed of a set of elements of the different types (struct);

- *ValueItem:* represents a value item (leaf) of the domain model.

The following table lists the supported properties used in the definition of a "ValueArray" type (in addition to the generic ones for "Value" type).

Table 14: List of properties supported by the ValueArray class

Property name	Short Description	dfdl or dmx
occursCountKind	Specifies how the actual number of occurrences is to be established. Supported values are 'fixed' and 'expression'.	dfdl
occursCount	Specifies the number of occurrences of the element.	dfdl

The following table lists the supported properties used in the definition of a "ValueItem" type (in addition to the generic ones for "Value" type):

Table 15: List of properties supported by the ValueItem class

Property name	Short Description	dfdl or dmx
occursCountKind	Specifies how the actual number of occurrences is to be established. Supported values are 'fixed' and 'expression'.	dfdl
occursCount	Specifies the number of occurrences of the element.	dfdl

DMXProperty: the DMXProperty class represents an element property (or attribute). This is simply a pair of string values: a name and a value.

Table 16: List of operations of the DMXProperty class

Operation name	Input	Output	Description
DMXProperty	String, String		The property constructor.
isExpression	void	boolean	Checks if the value of this property is an expression to be evaluated (i.e., has the format { expression }).
name	void	String	Access the name of the property.
toString	void	String	Generate the textual representation of the property.
value	void	String	Access the value of the property.

This package contains also extra classes that support some additional functionality:

DMXSize: the class DMXSize represents the size of a data block. It provides byte and bit reference;

Table 17: List of operations of the DMXSize class

Operation name	Input	Output	Description
add	DMXSize	DMXSize	Add this object with another size object Returns a new size object, containing the value of this after adding the other size's number of bits and bytes.
addBits	int	DMXSize	Add the given number of bits. Returns a new size object, containing the value of this after adding the requested number of bits.
addBytes	int	DMXSize	Add the given number of bytes. Returns a new size object, containing the value of this after adding the requested number of bytes.
DMXSize	void	void	The size constructor This constructor builds a 'zero' object (0 nrBytes and 0 nrBits)
DMXSize	int, long	DMXSize	The size constructor
equals	Object	boolean	Indicates whether some other object is "equal to" this one.
getNrBits	void	int	Access the number of bits.
getNrBytes	void	long	Access the number of bytes.
getNrBytesTotal	void	long	Access the number of bytes necessary to store a block of data of size this.
hashCode	void	int	Returns a hash code value for the object. This method is supported for the benefit of hash tables.
isEqualTo	DMXSize	boolean	Checks if this object equals another size instance Returns true if both instances represent the same size value; false otherwise
isGreaterThan	DMXSize	boolean	Checks if this object is greater than another size instance. Returns true if this represents a greater size value; false otherwise.
isLessThan	DMXSize	boolean	Checks if this object is less than another size instance. Returns true if this represents a lesser size value; false otherwise

Operation name	Input	Output	Description
subtract	DMXSize	DMXSize	Subtract another size object from this instance. Return a new size object, containing the value of this after subtracting the other size's number of bits and bytes
toString	Void	String	Generate the textual representation of size.

ErrorIndicator: The ErrorIndicator class stores the error information related to an instance of Element (obtained when parsing the binary file).

Table 18: List of operations of the ErrorIndicator class

Operation name	Input	Output	Description
errorMessage	void	String	Access the error message
errorStatus	void	boolean	Access the error status
getElement	void	Element	Access the element associated with this error
isPropagateable	void	boolean	Checks is this error should be propagated
isSevere	void	boolean	Checks is this error is severe

4.2.3. Utilities

The classes contained in this package provide functionalities for advanced access to the contents of a binary file.

Figure 5 shows class diagram of the Utilities classes' package.

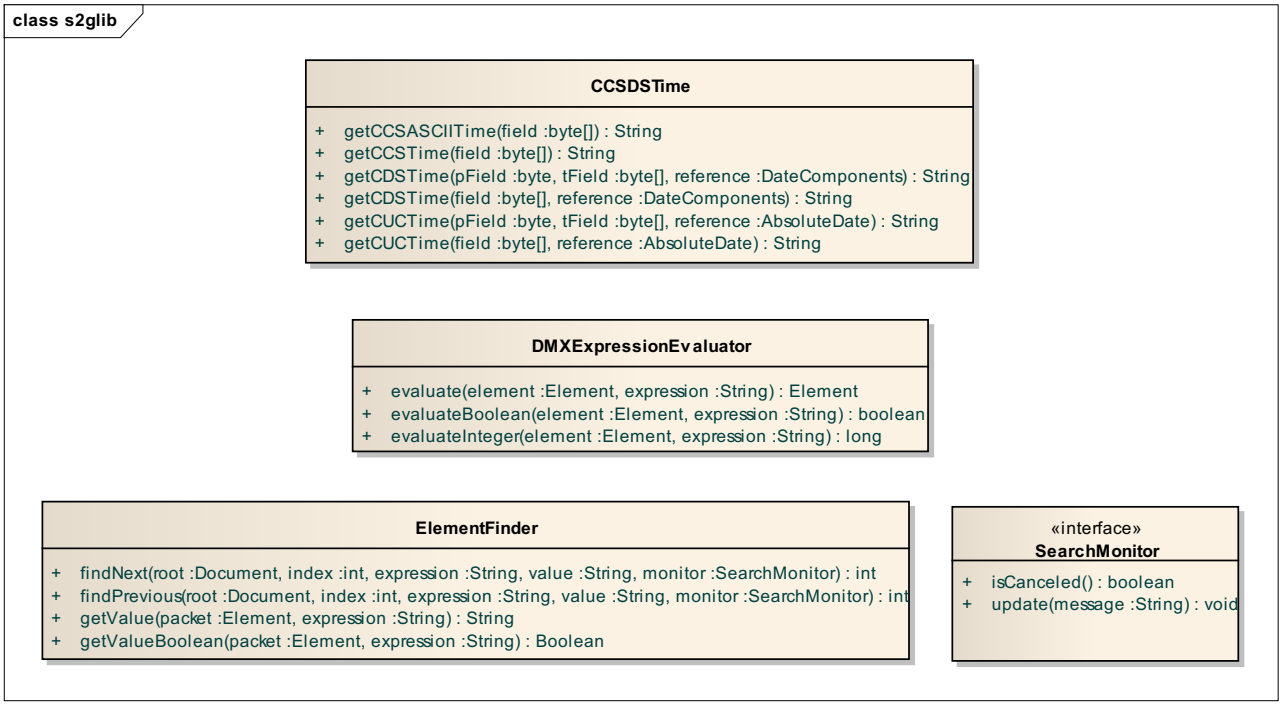


Figure 5: Utilities classes' package class diagram

Classes contained in this package are:

CCSDSTime: the CCSDSTime class is a utilities class that allows to convert binary data into time considering the CCDSD Time Standards;

Table 19: List of operations of the CCSDSTime class

Operation name	Input	Output	Description
`getCCSASCIITime`	field: byte[]	String	Convert from Calendar Segmented (ASCII) Time Code. Returns the string time.
`getCCSTime`	field: byte[]	String	Convert from Calendar Segmented Time Code. Returns the string time in TAI.
`getCDSTime`	preamble: byte, time: byte[], reference: DateComponents[3]	String	Convert from Day Segmented Time Code. Returns the string time in TAI.
`getCDSTime`	timeField: byte[], reference: DateComponents[2]	String	Convert from Day Segmented Time Code. Returns the string time in TAI.

[3] Data type defined in orekit library (dependency package for s2glib).

Operation name	Input	Output	Description
`getCUCTime`	preamble: byte, time: byte[], reference: AbsoluteDate[2]	String	Convert from Unsegmented Time Code. Returns the string time in TAI.
`getCUCTime`	timeField: byte[], reference: AbsoluteDate[2]	String	Convert from Unsegmented Time Code. Returns the string time in TAI.

DMXExpressionEvaluator: the DMXExpressionEvaluator class represents an evaluation engine for path expressions. Given an element and an expression, it provides the element "pointed" by the expression in the following way:

```
DMXElement targetElement = DMXExpressionEvaluator.evaluate(initialElement, expressionString);
```

Table 20: List of operations of the DMXExpressionEvaluator class

Operation name	Input	Output	Description
`evaluate`	Element, String	Element	Evaluate a path expression. Return the element "pointed" by the expression; null if unable to evaluate the expression correctly.
`evaluateBoolean`	Element, String	boolean	Evaluate an expression known to be a boolean value. Returns the boolean value obtained by evaluating the expression.
`evaluateInteger`	Element, String	long	Evaluate an expression known to be an integer value. Returns the long value obtained by evaluating the expression.

ElementFinder: the ElementFinder class provides the means to search the Element tree for specific values. Value class represents the internals of a domain model element (storing its properties, such as size and representation, and children).

Table 21: List of operations of the ElementFinder class

Operation name	Input	Output	Description
`findNext`	Document, index, expression, value, monitor	int	Find the next element stored below root element, after the index that has an element with the given value at the element retrieved by evaluating the expression. Search shall run until the number of elements below root is exhausted.

Operation name	Input	Output	Description
`findPrevious`	Document, index, expression, value, monitor	int	Find the previous element stored below root element, after the index that has an element with the given value at the element retrieved by evaluating the expression. Search shall run until the number of elements below root is exhausted.
`getValue`	Element, String	String	Access the value of the packet element accessible from the given element by evaluating the expression. Returns the value of the element.
`getValueBoolean`	Element, String	Boolean	Evaluate the expression in the packet element given by the parameter. Returns the boolean result of evaluating the expression.

This package contains also extra interface classes to support the defined functionalities:

SearchMonitor: the SearchMonitor interface represents a cancellable monitor to be used in support of ElementFinder;

Table 22: List of operations of the SearchMonitor interface

Operation name	Input	Output	Description
`isCanceled`	void	boolean	Checks if the monitor has been cancelled. Returns true if requested for cancel; false otherwise.
`update`	String	void	Update the information provided by the monitor.

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.dfdllib.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 pt.com.deimos.s2g.lib.* 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.childAt(0)); //output the entire value of the first data unit in the file
```

4. Close the document once not needed.

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

4.3.1.3. Utilities

Steps for using the Utilities module.

5. Import package pt.com.deimos.s2g.lib.* in your code;
6. Use the static methods using the methods explained in section 4.2.3 as a library of functionalities. No instance creation is needed

```
System.out.pritln(ElementFinder.getValue(document.childAt(0), "/ISP");
```

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 schema a setting integer values.

```
import org.esa.s2g.dfdllib.DFDLLib;
import pt.com.deimos.s2g.lib.Document;
import pt.com.deimos.s2g.lib.Element;
import pt.com.deimos.s2g.lib.ElementFinder;
import pt.com.deimos.s2g.lib.loader.ErrorLoadingException;

import java.io.File;
import java.io.IOException;
import java.math.BigInteger;

public class DFDLLibTest {

    public static void main(String[] args) throws InterruptedException, ErrorLoadingException, IOException {

        String timeSettingsPath = "A_TIME_SETTINGS_PATH";
        String filenamePath = "A_FILENAME_PATH";
        String schemaPath = "A_SCHEMA_PATH";
        int nrOfBytes = 1; // the size of the document in bytes

        Document document = createDocumentWith(timeSettingsPath, filenamePath, schemaPath, nrOfBytes);

        String integerOnePath = "./Integer_1";
        String integerTenPath = "./Integer_10";

        setValueIntegerOn(document, integerOnePath, BigInteger.ONE);
        setValueIntegerOn(document, integerTenPath, BigInteger.TEN);

        getValueIntegerFrom(document, integerOnePath);
        getValueIntegerFrom(document, integerTenPath);

        document.close();
    }

    /**
     * Create a document with the given parameters
     *
     * @param timeSettingsPath the time settings path
     * @param filenamePath the file path where data is going to be written
     * @param schemaPath the schema path corresponding to this file
     * @param nrOfBytes the number of bytes according to this schema
     *
     * @return the dfdl4s document
     *
     * @throws InterruptedException
     * @throws ErrorLoadingException
     * @throws IOException
     */
    private static Document createDocumentWith(String timeSettingsPath, String filenamePath, String schemaPath, int nrOfBytes) throws
InterruptedException, ErrorLoadingException, IOException {

        DFDLLib.initLib(timeSettingsPath);

        Document document = DFDLLib.createDocument(new File(filenamePath));

        byte data[] = new byte[nrOfBytes];
        DFDLLib.appendElements(document, DFDLLib.getSchemaDefinition(schemaPath), data);

        return document;
```

```
    }

    private static void setValueIntegerOn(Document document, String path, BigInteger integerValue) {

        Element root = document.childAt(0);
        Element element = ElementFinder.getElement(root, path);
        element.setValueInteger(integerValue);
    }

    private static BigInteger getValueIntegerFrom(Document document, String path) {

        Element root = document.childAt(0);
        Element element = ElementFinder.getElement(root, path);
        return element.getValueInteger();
    }
}
```

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].

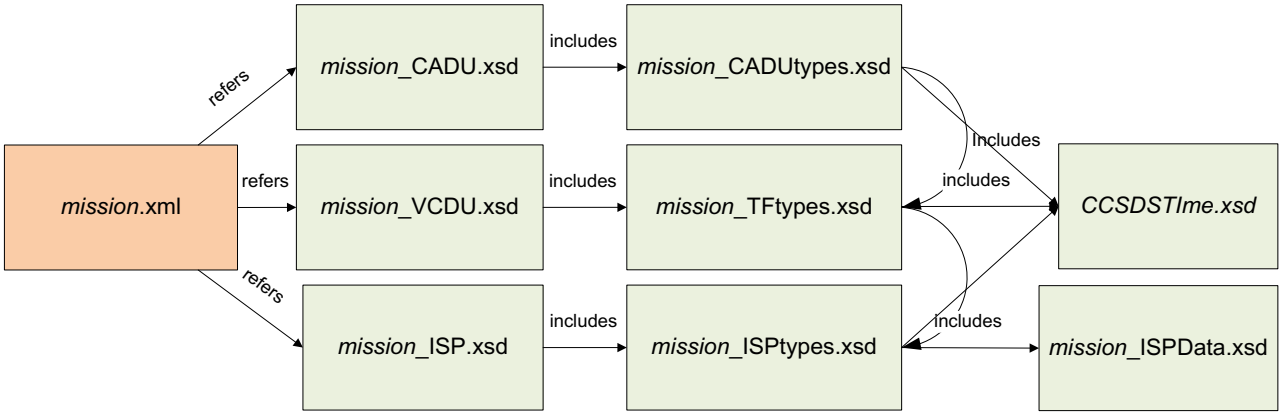


Figure 6: Mission Configuration files structure

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="Sentinel3X-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>
[...snip...]
    </xs:complexType>

    <xs:complexType name="TypeData">
        <xs:sequence>
            <xs:element name="data"
              type="xs:byte" dfdl:lengthKind="expression" 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, DLFL does not provide a way to specify the representation of the nodes in the information model, meaning that although DFDL 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, DFDL4S must extend DFDL with a mechanism to interpret the raw data and transform it into a human readable format.

To accomplish this goal, DFDL4S 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 DFDL extends the XML Schema definition by adding attributes that describe properties of the represented binary data, DFDL4S 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>
...
```

Representation specifi
is presented in the GUI

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 be 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 9,223,372,036,854,775,808, 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.

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[4]:

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 Figure 10 [5].

[4] In general, DFDL expressions are used in more situations (see [RD.1]).

[5] 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 10: DFDL4S Expression Language

Note some important differences between the language specified by DFDL standard and the supported by DFDL4S. Here *if, multiplicative, 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_1`,` e_2`,...,` e_n`]`

where *numeric* is an expression that evaluates to integer and e_1 to e_n are integer literals, and evaluates to *true* if and only if *numeric* evaluates to one of e_1, e_2,..., or e_n.

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.[6]

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, only `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`.

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`.

[6] 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.

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.

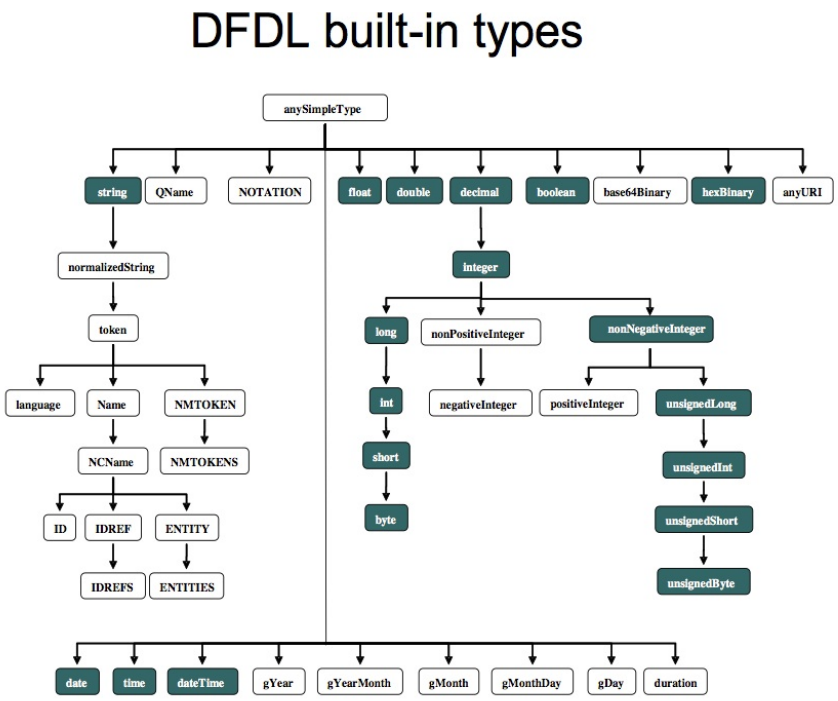


Figure 11: DFDL Simple Types

For every DFDL numeric simple type DFDL4S validates the associated explicit length restrictions, specified for base-2 binary number elements listed in Table 23 (see Section. 12.3.7.2.1 of [RD.1]).

Table 23 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 24 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 24. 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 24 DFDL core properties compatibility

Property name	Short Description	DFDL4S Compliance	Supported values
Properties Common to both Content and Framing			
`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`

Property name	Short Description	DFDL4S Compliance	Supported values
utf16Width	Specifies whether the encoding 'UTF-16' should be treated as a fixed or variable width encoding.	PC	variable
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
Common Framing, Position, and Length			
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

Property name	Short Description	DFDL4S Compliance	Supported values
initiator	Specifies a whitespace separated list of alternative literal strings one of which marks the beginning of the element or group of elements.	PC	""
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

Property name	Short Description	DFDL4S Compliance	Supported values
`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`
`length`	Specifies the length of this element in units that are specified by the dfdl:lengthUnits 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 lengthKind is 'explicit'.	C	Non-negative Integer or DFDL Expression
`prefixIncludesPrefixLength`	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	
`prefixLengthType`	This type specifies the representation of the length prefix, which is in the PrefixLength region.	NC	
`lengthPattern`	Specifies a regular expression that, on parsing, is executed against the datastream to determine the length of the element. Only used when lengthKind is 'pattern'.	NC	
Simple Type Content			

Property name	Short Description	DFDL4S Compliance	Supported values
representation	The permitted representation properties for each logical type. Valid values 'text' and 'binary' and are dependent on logical type.	PC	binary
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 dfdl:escapeSchemeRef 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	

Property name	Short Description	DFDL4S Compliance	Supported values
escapeEscapeCharacter	Specifies one character that escapes an immediately following dfdl:escapeCharacter or first character of dfdl:escapeBlockEnd.	N/A	
extraEscapedCharacters	A whitespace separated list of single characters that must be escaped in addition to the in-scope delimiters	N/A	
generateEscapeBlock	Controls when escaping is used on unparsing. Valid values 'always', 'whenNeeded'.	N/A	
textBidi	Indicates the text content of the element is bidirectional.	N/A	
textBidiOrdering	Defines how bidirectional text is stored in memory.	N/A	
textBidiOrientation	Indicates how the text should be displayed.	N/A	
textBidiSymmetric	Defines whether characters such as < ([{ that have a symmetric character with an opposite directional meaning: >)] } should be swapped	N/A	
textBidiShaped	Defines whether characters should be shaped on unparsing.	N/A	
textBidiNumeralShapes	Defines on unparsing whether logical numbers with text representation should have Arabic shapes.	N/A	
textStringJustification	Valid values 'left', 'right', 'center'	N/A	

Property name	Short Description	DFDL4S Compliance	Supported values
textStringPadCharacter	The value that is used when padding or trimming string elements.	N/A	
truncateSpecifiedLengthString	Used on unparsing only	N/A	
decimalSigned	Indicates whether an xs:decimal element is signed.	N/A	
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	

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

Property name	Short Description	DFDL4S Compliance	Supported values
textBooleanTrueRep	A whitespace separated list of representations to be used for 'true'.	N/A	
textBooleanFalseRep	A whitespace separated list of representations to be used for 'false'.	N/A	
textBooleanJustification	Controls how the data is padded or trimmed on parsing and unparsing.	N/A	
textBooleanPadCharacter	The value that is used when padding or trimming boolean elements.	N/A	
binaryBooleanTrueRep	This value gives the representation to be used for 'true'	PC	1
binaryBooleanFalseRep	This value gives the representation to be used for 'false'	PC	0
calendarPattern	Defines the ICU pattern that describes the format of the calendar.	NC	
calendarPatternKind	Valid values 'explicit', 'implicit'	NC	
calendarCheckPolicy	Indicates how lenient to be when parsing against the pattern.	NC	
calendarTimeZone	This property provides the time zone that will be assumed if no time zone explicitly occurs in the data.	NC	
calendarObserveDST	Whether the time zone given in dfdl:calendarTimeZone observes daylight savings time.	NC	
calendarFirstDayOfWeek	The day of the week upon which a new week is considered to start.	NC	

Property name	Short Description	DFDL4S Compliance	Supported values
calendarDaysInFirstWeek	Specify the number of days of the new year that must fall within the first week.	NC	
calendarCenturyStart	This property determines on parsing how two-digit years are interpreted.	NC	
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	
Sequence Groups			
sequenceKind	Valid values are 'ordered', 'unordered'	PC	ordered

Property name	Short Description	DFDL4S Compliance	Supported values
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	""
separatorPosition	Valid values 'infix', 'prefix', 'postfix'	NC	
separatorSuppressionPolicy	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	
floating	Whether the occurrences of an element in an ordered sequence can appear out-of-order in the representation.	PC	no
hiddenGroupRef	Elements within this model group will not be added to the Infoset, and are called hidden elements.	NC	
discriminator	This property allows the resolution of a point of uncertainty by choosing the specific alternatives based on the evaluation of a boolean expression denominated *test*. The test attribute of dfdl:discriminator 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

Property name	Short Description	DFDL4S Compliance	Supported values
Choice Groups			
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	
Array elements and optional elements			
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	
Calculated Values			
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