DFDL4S library

Developer's Manual

Code	:	S2G-DME-TEC-SUM078
Issue	:	1.C
Date	:	06/03/2015

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

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

This page intentionally left blank

Document Information

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

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

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

Archiving	
Word Processor:	MS Word 2000
File Name:	S2G-DME-TEC-SUM078-1C.doc

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

Table of Contents

List of Tables

List of Figure

1. INTRODUCTION

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

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

1.1. Purpose

The objective of this manual is to provide an operation manual of the use of DFDL4S library to read, parse, inspect or update 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
S2G	Space to Ground
SoW	Statement of Work

This page intentionally left blank

2. RELATED DOCUMENTS

2.1. Applicable Documents

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

Table 1: Applicable documents

Reference	Code	Title	Issue
[AD.1]	S2G-DME-TEC-TNO005	S2G Data Viewer Technical Note: Technical Specification	1.A
[AD.2]	S2G-DME-RCR-ECP032	S2G Data Viewer: Proposal for CCN1 Activities	1.B
[AD.3]	S2G-DME-RCR-ECP056	S2G Data Viewer: Proposal for CCN2 Activities	1.C

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-P-R.207	Data Format Description Language (DFDL) v1.0 Core Specification	1.0
[RD.3]	ECSS E-70-41	Ground systems and operations - Telemetry & telecommand packet utilisation	

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.3]. Based on that standard format, each mission customizes the packets hierarchy to according to its specific needs and instruments.

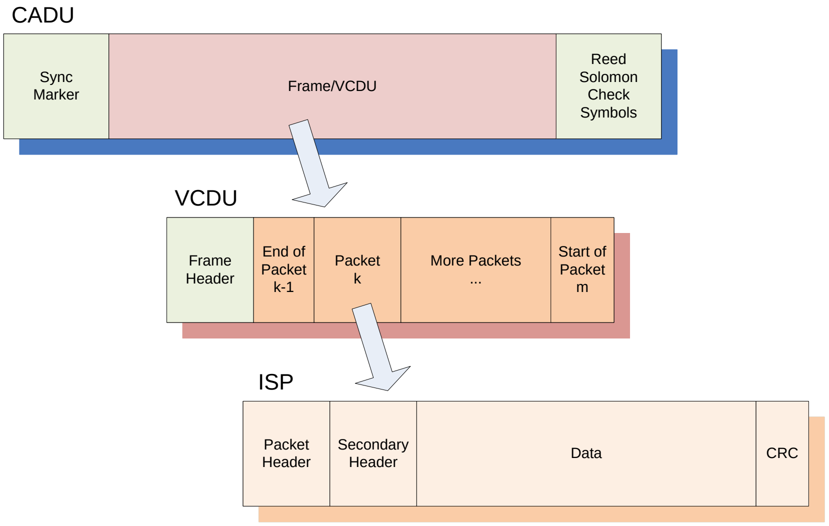

Figure 1: Hierarchy of Data received by the Ground Sensor Stations

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 1.6
Mac OS (64 bit)	RAM:	2 GB
	Disk Space:	50 MB
	Dependencies:	Java 1.6
Windows (32/64 bit)	RAM:	1 GB
	Disk Space:	50 MB
	Dependencies:	Java 1.6

3.3. Installation

To install DFDL4S library simply unzip the distribution archive (dfdl4s_dist.zip) into the installation directory. The folder structure resultant of this action is as follows:

□ DFDL4S: main folder containing the library and test execution scripts:

- dfdl4s.jar: the DFDL4S library itself;
- 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

□ DFDL4S/bin: Compiled code used for the sample tests;

□ DFDL4S/docs: Doxygen generated documentation of the library API in html format;

□ DFDL4S/lib: External libraries used by DFDL4S;

□ DFDL4S/resources/data: Mission Configuration Files (examples);

□ DFDL4S/resources/time: UTC-TAI relation file used for DFDL4S initialization;

□ DFDL4S/src: Source code for the library usage examples;

□ DFDL4S/tests/data: Sample data for running the example tests.

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[1]. 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.

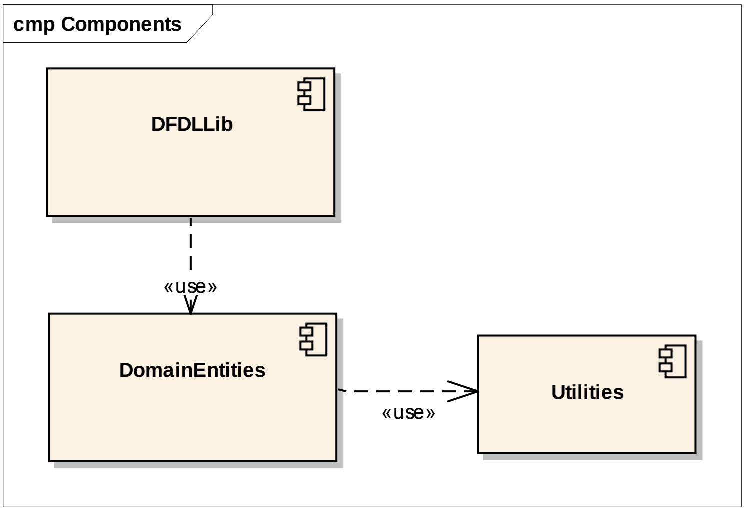

Figure 2: DFDL4S library high-level package diagram

[1] 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 4-3 shows the DFDLLib class diagram, listing interface methods.

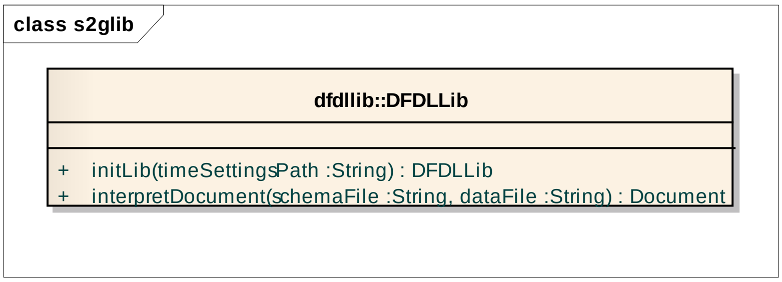
class s2glib
dfdllib::DFDLLib
+ initLib(timeSettingsPath :String) : DFDLLib
+ interpretDocument(schemaFile :String, dataFile :String) : Document

Figure 4-3: DFDLLib class diagram

Table 4: List of operations of the DFDLLib class

Operation name	Input	Output	Description
`initLib`	path	DFDLLib	This method initialises the DFDLLib: sets the UTC TAI conversion data.
`interpretDocument`	schemaFilePath, dataFilePath	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.
`getVersion`	-	String	The version number and release date of the library.

4.2.2. DomainEntities

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

Figure 4-4 shows class diagram of the DomainEntities package.

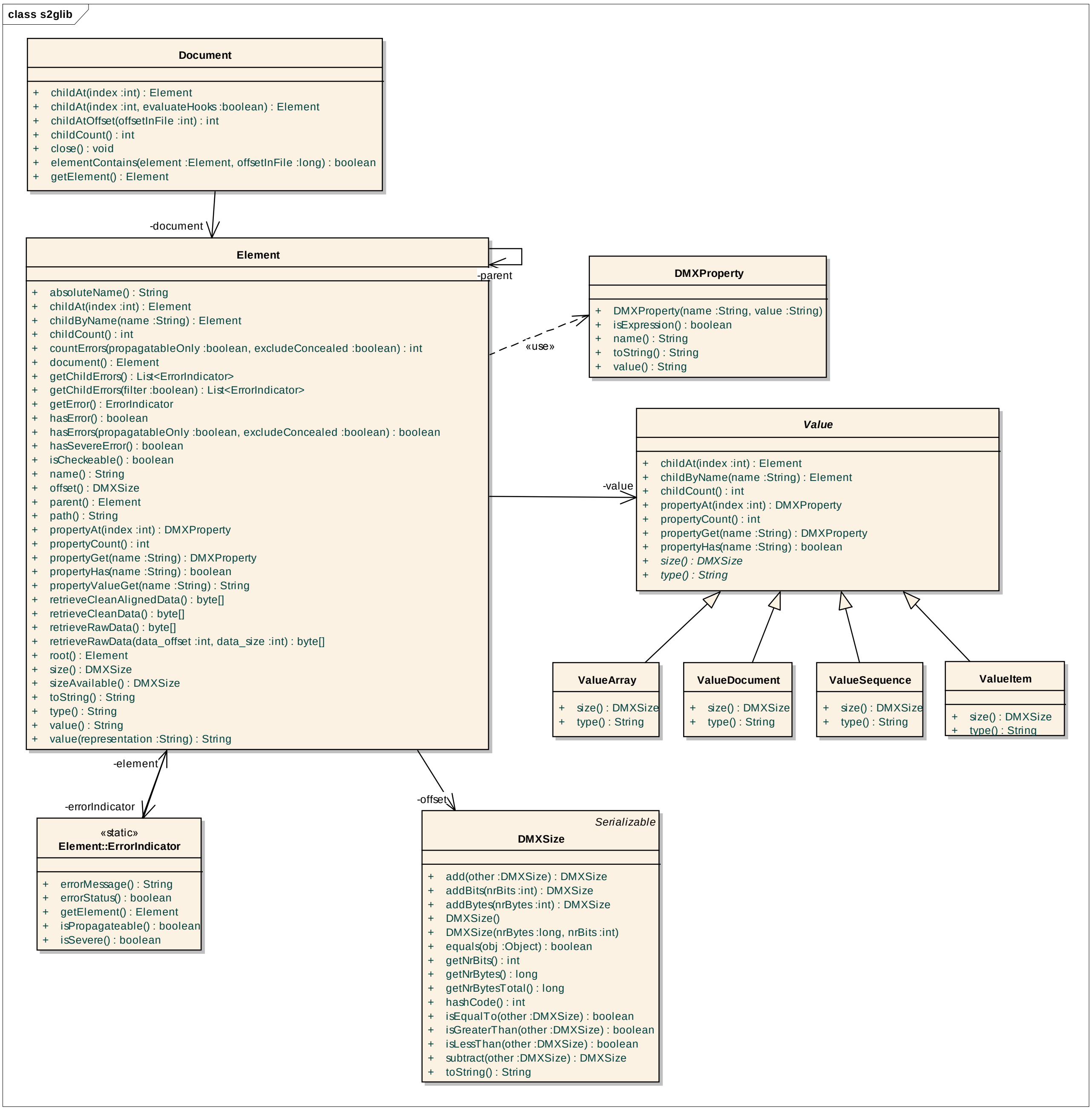
class s2glib
Document
+ childAt(index :int) : Element
+ childAt(index :int, evaluateHooks :boolean) : Element
+ childAtOffset(offsetInFile :int) : int
+ childCount() : int
+ close() : void
+ elementContains(element :Element, offsetInFile :long) : boolean
+ getElement() : Element
-document
Element
-parent
+ absoluteName() : String
+ childAt(index :int) : Element
+ childByName(name :String) : Element
+ childCount() : int
+ countErrors(propagatableOnly :boolean, excludeConcealed :boolean) : int
+ document() : Element
+ getChildErrors() : List<ErrorIndicator>
+ getChildErrors(filter :boolean) : List<ErrorIndicator>
+ getError() : ErrorIndicator
+ hasError() : boolean
+ hasErrors(propagatableOnly :boolean, excludeConcealed :boolean) : boolean
+ hasSevereError() : boolean
+ isCheckeable() : boolean
+ name() : String
+ offset() : DMXSize
+ parent() : Element
+ path() : String
+ propertyAt(index :int) : DMXProperty
+ propertyCount() : int
+ propertyGet(name :String) : DMXProperty
+ propertyHas(name :String) : boolean
+ propertyValueGet(name :String) : String
+ retrieveCleanAlignedData() : byte[]
+ retrieveCleanData() : byte[]
+ retrieveRawData() : byte[]
+ retrieveRawData(data_offset :int, data_size :int) : byte[]
+ root() : Element
+ size() : DMXSize
+ sizeAvailable() : DMXSize
+ toString() : String
+ type() : String
+ value() : String
+ value(representation :String) : String
«use»
DMXProperty
+ DMXProperty(name :String, value :String)
+ isExpression() : boolean
+ name() : String
+ toString() : String
+ value() : String
-value
Value
+ childAt(index :int) : Element
+ childByName(name :String) : Element
+ childCount() : int
+ propertyAt(index :int) : DMXProperty
+ propertyCount() : int
+ propertyGet(name :String) : DMXProperty
+ propertyHas(name :String) : boolean
+ size() : DMXSize
+ type() : String
ValueArray
+ size() : DMXSize
+ type() : String
ValueDocument
+ size() : DMXSize
+ type() : String
ValueSequence
+ size() : DMXSize
+ type() : String
ValueItem
+ size() : DMXSize
+ type() : String
-element
-errorIndicator
«static»
Element::ErrorIndicator
+ errorMessage() : String
+ errorStatus() : boolean
+ getElement() : Element
+ isPropagateable() : boolean
+ isSevere() : boolean
-offset
Serializable
DMXSize
+ add(other :DMXSize) : DMXSize
+ addBits(nrBits :int) : DMXSize
+ addBytes(nrBytes :int) : DMXSize
+ DMXSize()
+ DMXSize(nrBytes :long, nrBits :int)
+ equals(obj :Object) : boolean
+ getNrBits() : int
+ getNrBytes() : long
+ getNrBytesTotal() : long
+ hashCode() : int
+ isEqualTo(other :DMXSize) : boolean
+ isGreaterThan(other :DMXSize) : boolean
+ isLessThan(other :DMXSize) : boolean
+ subtract(other :DMXSize) : DMXSize
+ toString() : String

Figure 4-4: 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;

Table 5: List of operations of the Document class

Operation name	Input	Output	Description
`childAt`	index	Element	Access the child at the given index (does not evaluate hooks). Returns the requested child element.
`childAt`	index, evaluateHooks	Element	Access the child at the given index. Returns the requested child element.
`childAtOffset`	offsetInFile	int	Access the child at the given file offset. Returns the requested child element index.
`childCount`	void	int	Returns the number of children of the element.
`close`	void	int	Returns the number of children of the element.
`elementContains`	Element, offset	boolean	Verifies that the element contains the offset. Returns true if element contains the offset; false otherwise.
`getElement`	void	Element	Returns the root element.

- ❑ *Element:* the Element class represents a domain element that is used to structure the binary data;

Table 6: List of operations of the Element class

Operation name	Input	Output	Description
`absoluteName`	void	String	Access the absolute name of the element
`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
`countErrors`	propagateableOnly: boolean, excludeConcealed: boolean	int	Access the number of errors (including children errors) associated with this item.
`document`	void	Element	Access the top element (document) that contains this element.

Operation name	Input	Output	Description
`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: boolean, excludeConcealed: boolean	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
`isCheckeable`	void	boolean	Indicates whether this element is checkable from the point of view of Error Control (CRC or RS fields).
`name`	void	String	Access the name of the element
`offset`	void	DMXSize	Access the offset of the element, according to the offset type
`parent`	void	Element	Access the parent of the element
`path`	void	String	Access the path of the element, starting at the packet level (document level is not considered)
`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

Operation name	Input	Output	Description
propertyValueGet	String	String	Access the value of a property with a given name
retrieveCleanAlignedData	void	byte[]	Access the clean and aligned data of the element
retrieveCleanData	void	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.
retrieveRawData	void	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	offset, size	byte[]	Access the raw data "inside" the element
root	void	Element	Access the packet element (below document) that contains this element
setValue	Value	void	Setup a new Element value
size	void	DMXSize	Access the size of the value represented
sizeAvailable	void	DMXSize	Access the actual size of data available in file
toString	void	String	Access the string representation of the element
type	void	String	Access the type of the element
value	void	String	Access the value of the element (according to the representation specified in the binary definition)
value	String	String	Access the value of the element (according to the specified representation)

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

Table 7: 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.
`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 8: 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

Property name	Short Description	dfdl or 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 9: 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 10: 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 11: 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 12: 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.

Operation name	Input	Output	Description
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
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 13: 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 4-5 shows class diagram of the Utilities classes' package.

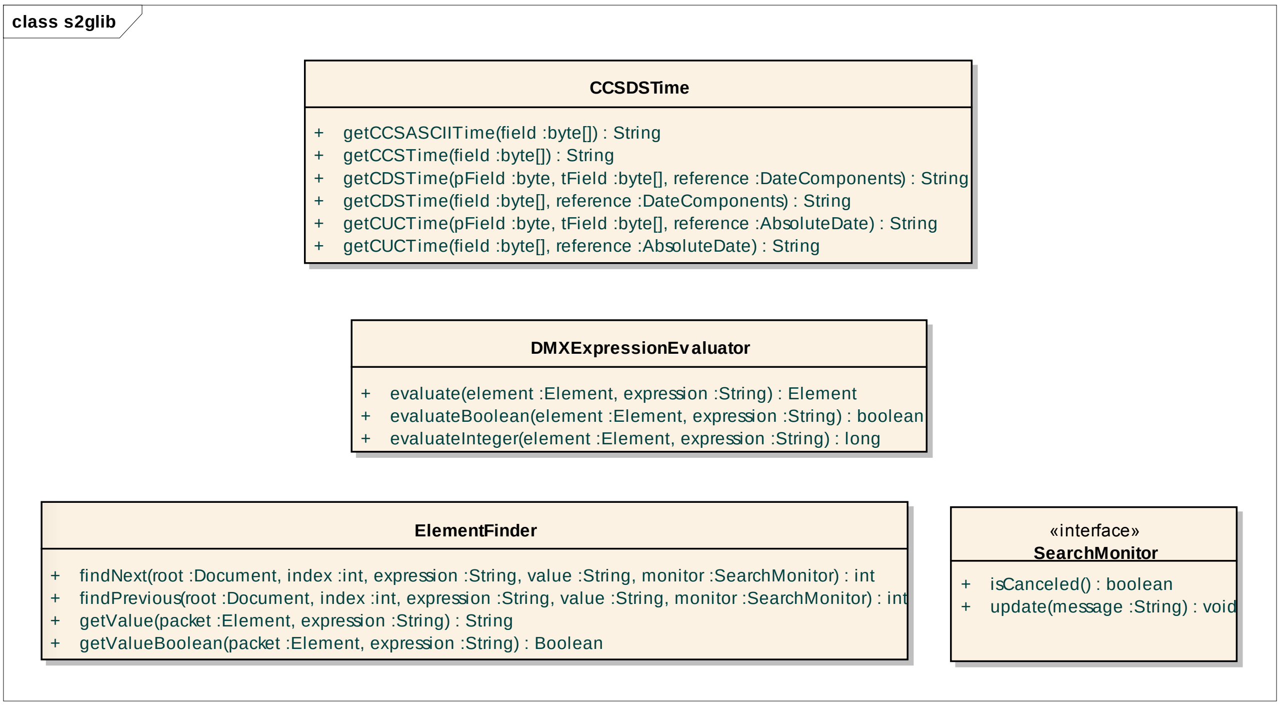
class s2glib
CCSDSTime
+ getCCSASCIITime(field :byte[]) : String
+ getCCSTime(field :byte[]) : String
+ getCDSTime(pField :byte, tField :byte[], reference :DateComponents) : String
+ getCDSTime(field :byte[], reference :DateComponents) : String
+ getCUCTime(pField :byte, tField :byte[], reference :AbsoluteDate) : String
+ getCUCTime(field :byte[], reference :AbsoluteDate) : String
DMXExpressionEvaluator
+ evaluate(element :Element, expression :String) : Element
+ evaluateBoolean(element :Element, expression :String) : boolean
+ evaluateInteger(element :Element, expression :String) : long
ElementFinder
+ findNext(root :Document, index :int, expression :String, value :String, monitor :SearchMonitor) : int
+ findPrevious(root :Document, index :int, expression :String, value :String, monitor :SearchMonitor) : int
+ getValue(packet :Element, expression :String) : String
+ getValueBoolean(packet :Element, expression :String) : Boolean
«interface»
SearchMonitor
+ isCanceled() : boolean
+ update(message :String) : void

Figure 4-5: 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 14: 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.

Operation name	Input	Output	Description
getCDSTime	preamble: byte, time: byte[], reference: DateComponents[2]	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.
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 15: 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).

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

Table 16: 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.
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 17: 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.

```
import org.esa.s2g.dfdllib.DFDLLib;

import pt.com.deimos.s2g.lib.Document;
import pt.com.deimos.s2g.lib.ElementFinder;
import pt.com.deimos.s2g.lib.loader.ErrorLoadingException;

public class DFDLLibTest {

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

        int element = 0;
        String path = "/Primary_Header/Virtual_Channel_Id_V2";
        String value = "0";

        DFDLLib.initLib("resources/time");

        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");

        System.out.println(document.childAt(0));
        System.out.println("Changing " + path + " of element " + element + " from " +
ElementFinder.getValue(document.childAt(element), path) + " to " + value);

        ElementFinder.setValue(value, document.childAt(element), path);

        System.out.println("Final value = " + ElementFinder.getValue(document.childAt(element),
path));

        document.close();

    } // End of main
}
```

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 ./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 ./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].

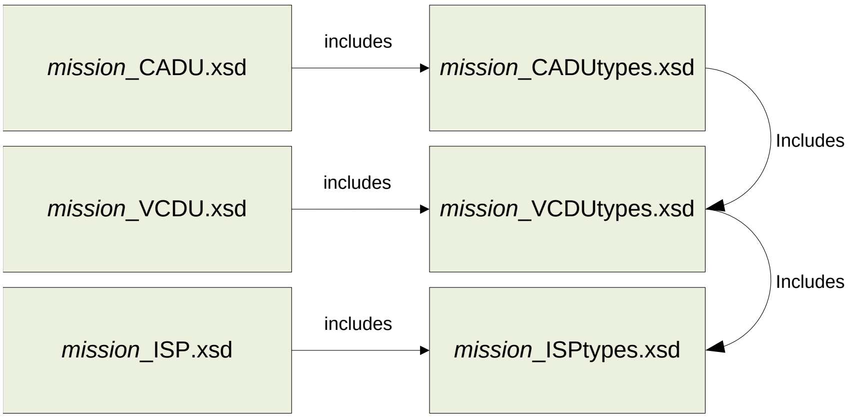

Figure 6: Mission Configuration files structure

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. 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.

```
<?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:element name="ISP" dfdl:encoding="utf-8" dfdl:byteOrder="bigEndian">
      <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)

APPENDIX A – COMPATIBILITY WITH DFDL CORE SET

This section presents a compliance table verifying which of the DFDL core set properties are implemented in the DFDL4S library. Column "DFDL4S Compliance" contains values: "C" – 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).

Property name	Short Description	DFDL4S Compliance
Properties Common to both Content and Framing		
`byteOrder`	This property applies to all types with representation binary. Valid values 'bigEndian', 'littleEndian'.	C
`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'.	NC
`encoding`	Values are one of: IANA charset name; CCSID; DFDL standard encoding name; implementation-specific encoding name.	C
`utf16Width`	Specifies whether the encoding 'UTF-16' should be treated as a fixed or variable width encoding.	NC
`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.	NC
Common Framing, Position, and Length		
`alignment`	A non-negative number that gives the alignment required for the beginning of the item	NC
`alignmentUnits`	Scales the alignment so alignment can be specified in either units of bits or units of bytes.	NC

Property name	Short Description	DFDL4S Compliance
`fillByte`	Used on unparsing to fill empty space such as between two aligned elements.	NC
`leadingSkip`	A non-negative number of bytes or bits to skip before alignment is applied.	NC
`trailingSkip`	A non-negative number of bytes or bits to skip after the element, but before considering the alignment of the next element.	NC
`initiator`	Specifies a whitespace separated list of alternative literal strings one of which marks the beginning of the element or group of elements.	NC
`terminator`	Specifies a whitespace separated list of alternative text strings that one of which marks the end of an element or group of elements.	NC
`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’, ‘implicit’ and ‘pattern’)
`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 (‘bytes’ and ‘bits’)

Property name	Short Description	DFDL4S Compliance
`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
`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'.	PC (property not implemented but the semantics are supported in dfdl:length)
Simple Type Content		
`representation`	The permitted representation properties for each logical type. Valid values are 'text' and 'binary' and are dependent on logical type.	PC (only 'binary' is allowed)
`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.	N/A
`escapeKind`	The type of escape mechanism defined in the escape scheme. Valid values 'escapeCharacter', 'escapeBlock'.	N/A

Property name	Short Description	DFDL4S Compliance
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 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
textStringPadCharacter	The value that is used when padding or trimming string elements.	N/A
truncateSpecifiedLengthString	Used on unparsing only	N/A

Property name	Short Description	DFDL4S Compliance
`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
`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.	NC

Property name	Short Description	DFDL4S Compliance
`binaryDecimalVirtualPoint`	An integer that represents the position of an implied decimal point within a number	NC
`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.	NC
`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'	NC
`binaryBooleanFalseRep`	This value gives the representation to be used for 'false'	NC
`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
`calendarDaysInFirstWeek`	Specify the number of days of the new year that must fall within the first week.	NC

Property name	Short Description	DFDL4S Compliance
`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'	NC
`initiatedContent`	Valid values are 'yes', 'no'	NC
`separator`	Specifies a whitespace separated list of alternative literal strings that are the possible separators for the sequence.	NC
`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.	NC

Property name	Short Description	DFDL4S Compliance
`hiddenGroupRef`	Elements within this model group will not be added to the Infoset, and are called hidden elements.	NC
	Choice Groups	
`choiceLengthKind`	Valid values are 'implicit', 'explicit'	C
`choiceLength`	Specifies the length of the choice in bytes.	C
`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 and expression)
`occursCount`	Specifies the number of occurrences of the element.	C
`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