DFDL4S++ Library

Developer's Manual

Code : S2G-DME-TEC-SUM113
Issue : 1.N
Date : 12/05/2025

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

DEIMOS Engenharia S.A.
Av. Columbano Bordalo Pinheiro No. 75, 9 A1
1070-061 Lisboa, Portugal
Tel.: +351 21 893 3010
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 2019
File Name:	S2G-DME-TEC-SUM113-1N.doc

Document Change Log

Issue	Change description	Date	Pages Affected
1.A	First issue of the document.	17/02/2017	All
1.B	Update for initial release of DFDL4S++. Clear definition of API for both DFDL4S libraries	16/02/2018	All
1.C	Replaced JDK_HOME by JAVA_HOME in example instructions	04/05/2018	16, 17
1.D	Complete review after total uniformization with Java library, including: - update of the API classes list table - new sections and tables for classes BinaryBuffer, ErrorIndicator, Exception, IOException, InterruptedException, ErrorLoadingException - removal of section 3.5: "Traceability between Java and C++ implementations" - new annex "A" with DFDL4S++ library build instructions	30/08/2018	All
1.E	Add Section 4.1 Testing	17/12/2018	Section 4.1
1.F	Added new methods in DFDLLib class: createNewDocument, storeDocument (1), storeDocument (2) Added new methods in Document class: createElement (1), createElement (2), createElementTime, createElementInteger, createElementFloat32, createElementFloat64, createElementBytes, validate Added new methods in Element class: is() Added new functionality/class CCSDSElementTrait	03/03/2020	1)
1.G	Update the compilers in the Table 4: Minimum System Requirements for MacOS and Linux Added new methods in DFDLLib class: storeDocument (3) Added new methods in Document class: size, getCleanAlignedData, getRawData, childAt. Added new methods in Element class: parent, root, childCount, propertyValueGet, size, sizeAvailable, offset, retrieveRawData (1), retrieveRawData (2), getValueAsRepresentation (1), getValueAsRepresentation (2), getValueBinary (old getValueAsString), evaluate (old ElementFinder.getElement), evaluateBoolean, evaluateInteger. Added new methods in ErrorIndicator class: hasErro() – old	31/08/2020	All

	errorStatus(), isPropagateabel, isSevere, isExpressionMalformed, getPathElementNotFound, hasNonSevereError Added new class DataSize Remove methods in DFDLLib class: createDocument Remove methods in Element class: getChildErrors, hasError, hasSevereError Removed ElementFinder and BinaryBuffer class		
1H	Added section 3.4.11. DFDL4S_JAVA_OPTS with the information regarding the use of the environment variable for C++ applications. Added section 3.4.12. containing the information regarding the different functionalities between Java and C++ DFDL4S libraries.	02/09/2020	3.5.20 3.5.21
1I	Added new methods in Element class: -uniqueName() -absoluteUniqueName()	04/11/2020	3.5.3
1J	Added new classes: - CRCDecoder - Report - ReportProperties - Entry - CRCDecoder - ReedSolomonDecoder - RSDecodeState - RSErrorInfo	21/06/2021	
1K	Added new section and table for new exception class: - DFDL4SJException	07/12/2022	3.5.19
1L	Added new section name "3.4. Guidelines" and new sentence in "3.5. DFDL4S++ Implementation" to enhance the proper usage of DFDLLib. Updated "Minimum System Requirements" table: JDK version from 8 to 11 on all platforms; Clang version from 11 to 12 on macOS; G++ compiler version from 7 to 11 on Linux; Updated references related to these updates. Updated section 3.1.1 to make the last sentence more conditional.	29/05/2023	3.4 3.2 3.1.1
1M	Minimum System requirements updated for Microsoft Visual Studio (versions from 2015 to 2022)	09/10/2023	3.2
1N	Added macOS (arm64) to the updated tables "Installation Archives" and "Minimum System Requirements"	12/05/2025	Table 3 Table 4

Table of Contents

List of Tables

List of Figure

1. INTRODUCTION

The Space to Ground Data Viewer (S2G) [AD.1, AD.2, AD.3, AD.4, AD.5, AD.6, AD.7] is an extensible utility tool to support ground systems engineers during the test campaigns to inspect the contents of the communication channels between the signal-in-space and the ground systems apparatus. The Space to Ground testing comprises 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 DFDL for Space (DFDL4S) is the underlying software library used by S2G. It comprises the capability to use DFDL schemas [RD.1] to read, parse, interpret, update and create CADU, TF or ISP data files. The DFDL for Space C++ (DFDL4S++) is the DFDL4S library implemented in C++.

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 brief description of the DFDL4S++ library and some examples of use that should be used as a reference manual by model developers. An extensive description of the DFDL4S library is available on the Developer's Manual [RD.5].

The following sections of this document are organized as follows:

- Section 2 lists applicable and reference documents
- Section 3 provides instructions to install and launch the application.

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
DFDL4S++	DFDL for Space C++
ISP	Instrument Source Packet
S2G	Space to Ground Data Viewer
TF	Transfer Frame
SoW	Statement of Work

This page intentionally left blank

2. RELATED DOCUMENTS

2.1. Applicable Documents

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

Table 1: Applicable documents

Reference	Code	Title	Issue
[AD.1]	S2G-DME-TEC-TNO005	S2G Data Viewer Technical Note: Technical Specification	1.A
[AD.2]	S2G-DME-RCR-ECP032	S2G Data Viewer: Proposal for CCN1 Activities	1.B
[AD.3]	S2G-DME-RCR-ECP056	S2G Data Viewer: Proposal for CCN2 Activities	1.C
[AD.4]	S2G-DME-RCR-ECP075	S2G Data Viewer: Proposal for CCN3 Activities	1.B
[AD.5]	S2G-DME-RCR-ECP094	S2G Data Viewer: Proposal for CCN5 Activities	1.B
[AD.6]	S2G-DME-RCR-ECP111	S2G Data Viewer: Proposal for CCN7 Activities	1.A
[AD.7]	S2G-DME-RCR-ECP120	S2G Data Viewer: Proposal for CCN9 Activities	1.B

2.2. Reference Documents

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

Table 2: Reference documents

Reference	Code	Title	Issue
[RD.1]	GFD.207	Data Format Description Language (DFDL) v1.0	1.0
[RD.2]	ECSS E-70-41	Ground systems and operations - Telemetry & telecommand packet utilisation	
[RD.3]	REC-xml20081126	Extensible Markup Language (XML) 1.0 (Fifth Edition)	1.0
[RD.4]	REC-xpath20-20101214	XML Path Language (XPath) 2.0 (Second Edition)	2.0
[RD.5]	S2G-DME-TEC-SUM-078	DFDL4S library – Developer's Manual	1.E

Reference	Code	Title	Issue
[RD.6]	CCSDS 130.1-G-3	The Synchronization and Channel Coding – Summary of Concept and Rationale	3.0
[RD.7]	-	Clarification for CCSDS CRC-16 Computation Algorithm	-

This page intentionally left blank

3. GETTING STARTED

3.1. Introduction

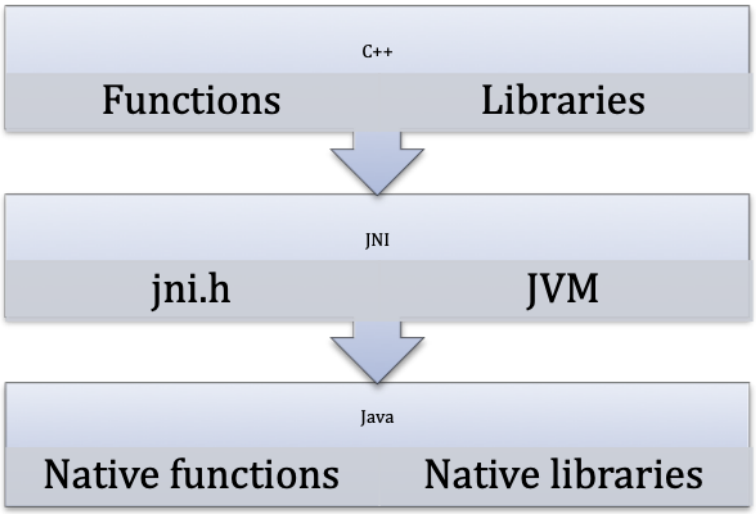
The DFDL4S++ is a library implemented in C++ that interprets the contents of the communication channels between the signal-in-space and the ground systems apparatus. It interprets files containing concatenated CADUs, TFs or ISPs, and lists of available data units and allows reading the fields and associated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

It is a C++ library packaged as a simple to use library 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 or create from scratch a new DFDL tree node value (writing it to the underlying file support).

3.1.1. DFDL4S++ architectural overview

The current implementation consists of a C++ library that wraps the native DFDL4S Java library through a JNI layer. The JNI layer allows Java code that runs inside a Java Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly[2].

Figure 1 - C++ to Java top-level architecture



[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.

[2] http://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/intro.html

For the sake of simplicity of the DFDL4S++ library, and also easing the future evolution of the library, the JNI details and implementation are hidden within inner classes. This assures a clean interface and whenever a new version of the library would be developed using native C++, the C++ layer would be added to take the place of the JNI and Java layers (which would be removed).

3.2. Installation

The DFDL4S++ is available for several platforms. Please use the version supporting your platform (according to Table 3). The installation should consider the minimum requirements presented in Table 4. The platforms presented have been used to support testing activities.

Table 3: Installation Archives

Archive	Supported Platform
dfdl4s++-X.Y.Z-linux-x64.tar.gz	Linux (64 bit)
dfdl4s++- X.Y.Z-mac-x64.tar.gz	macOS (x86_64)
dfdl4s++- X.Y.Z-mac-arm64.tar.gz	macOS (arm64)
dfdl4s++- X.Y.Z-win-x64.tar.gz	Windows (64 bit)

Table 4: Minimum System Requirements

Platform	Requirements	
Linux (64 bit)	RAM: Disk Space: Dependencies:	2 GB 10 MB G++ compiler 64bit (v11+) JDK 11 64 bit
macOS (x86_64)	RAM: Disk Space: Dependencies:	2 GB 10 MB Apple clang version 12.0.0 - 64 bit JDK 11 64 bit
macOS (arm64)	RAM: Disk Space: Dependencies:	2 GB 10 MB Apple clang version 17.0.0 - 64 bit JDK 23 64 bit
Windows (64 bit)	RAM: Disk Space: Dependencies:	2 GB 10 MB Microsoft Visual Studio versions from 2015 to 2022 64 bit JDK 11 64 bit

Each package contains the following:

- README: a read me file for quick reference
- LICENSE: the DFDL library licensing schema
- docs: folder containing the doxygen generated documentation of the library source code
- examples: folder containing the code with ready-to-use examples, i.e. a standalone C++ program (including a script to compile and build it)
- include: folder containing the header files for the DFDL4S++ library
- lib: folder containing the DFDL4S library + external libraries used by DFDL4S

The DFDL4S++ library should be installed on your library folder. For the sake of simplicity you should set an environment variable for it (e.g. DFDL4S) and use it to build your application.

To build your application you should refer to the:

- Developer's Manual [RD.5] from the Section 5 to Appendix A for information on how DFDL is implemented on DFDL4S
- Example on section 3.3
- Section 3.5 for the available API provided by the DFDL4S++ (in contrast to DFDL4S)

To check if the installation was successful, go to the examples folder on the DFDL4S++ library root folder and follow the bellow procedure:

1. Open the run_examples file (Linux and Mac: run_examples.sh | Windows: run_examples.bat) file and set an environment variable pointing to the home of the JDK installation - JAVA_HOME (see below examples for each of the platforms)
    a. On Linux:
    ```
    JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
    ```
    b. On Mac:
    ```
    JAVA_HOME=
    /Library/Java/JavaVirtualMachines/jdk-11.0.13.jdk/Contents/Home
    or
    JAVA_HOME=$(/usr/libexec/java_home)- to use the most recent version of JDK installed
    ```
    c. On Windows:
    ```
    JAVA_HOME= C:\Program Files\Java\jdk-11.0.13
    ```

Hint: If you don't know where is your JDK installation, if it was installed correctly you can find it:

a) On Linux open the terminal and type:

```
> find / -name javac
```

```
/usr/lib/jvm/java-11-<distribution>/bin/javac
```

The correct path for JAVA_HOME is:

```
/usr/lib/jvm/java-11-<distribution>
```

b) On Mac open the terminal and type:

```
> /usr/libexec/java_home -V

Matching Java Virtual Machines (1):

11.0_13, x86_64: "Java SE 11"
/Library/Java/JavaVirtualMachines/jdk-11.0.13.jdk/Contents/Home
```

The correct path for JAVA_HOME is:

```
/Library/Java/JavaVirtualMachines/jdk-11.0.13.jdk/Contents/Home
```

c) On Windows open the Command Prompt and type:

```
> for %i in (javac.exe) do @echo.   %~$PATH:i

C:\Program Files\Java\jdk-11.0.13\bin\javac.exe
```

The correct path for JAVA_HOME is:

```
C:\Program Files\Java\jdk-11.0.13
```

2. On Windows, you can previously select the version of Visual Studio in `run_examples.bat`, as explained in the script; otherwise, the default will be used. As an alternative, you can also use the "x64 Native Tools Command Prompt for VS <VS version>", whose link can be found in the Start Menu and will work from VS 2015 to VS 2022. Then, after changing to the respective directory, run the script that compiles and runs the Example.cpp and CreateNewDocument.cpp code:

   a. On Linux or Mac, type:

   ```
   ./run_examples.sh
   ```

   b. On Windows, type:

   ```
   run_examples.bat
   ```

3.2.1. Analysis to the variables defined in run_examples

As explained in previous section, the user has only to set the environment variable JAVA_HOME to compile and run the *Example.cpp* and *CreateNewDocument.cpp* using the *run_examples* batch/script file. However, multiple variables are defined during the execution of the *run_examples*. The table below shows the defined variables and a brief description about each one.

Table 5 - Variables defined to compile and run the Example.cpp

Variable name	Description	When they are used?
DFDL4S_INCLUDE	Path to the include folder of DFDL4S (contains the headers files)	**Compilation**
DFDL4S_LIB	Path to the lib folder where is the libraries dependencies necessaries to the dfdl4s++	

Variable name	Description	When they are used?
JVM_LIB	Path to the folder containing the libjvm.dylib	
JLI_LIB (macOS exclusive)	Path to the folder containing the libjli.dylib	
CFLAG	Specify additional switches to be passed to a compiler	
GXX (Unix Systems)	Specify the compiler to be used according to the unix system used	
(DY)LD_LIBRARY_PATH	Variable with the paths to the libraries needed to execute the compiled files	**Execution**
DYLD_INSERT_LIBRARIES (macOS exclusive)	Variable with the path to the libjli.dylib file inside the JLI_LIB folder	

3.3. Examples

With the DFDL4S++ package we include two examples (see the Example.cpp and CreateDocument.cpp file) to demonstrate some usages of the read, write and create document functionalities provided by the DFDL4S++ lib.

In particular, the examples shows how to use DFDL4S++ to:

- generate a binary file composed by a sequence of packets with a given structure;

- read / write elements of such binary file.

- create and store a document from scratch.

The packet structure is defined by a schema.

Run the example, (check section 3.2 as reference to run the example) and observe how the example implements the above use cases and processes the data.

3.4. Guidelines

To use the DFDL4S++ library you should follow a few guidelines (DFDLLib object lifecycle):

1. Initialise the DFDLLib object before using with:
    a. Path to the Orekit UTC TAI Initialisation file
    b. Path to the DFDL4S lib jar files
    c. Specify JVM options

    ➢ `DFDLLib dfdl_lib = DFDLLib("resources/time", "../lib", "-Xmx1024M");`

2. Re-use the DFDLLib instance on other classes:

- `Document document_1 = dfdl_lib.createNewDocument( schemaFile );`

3. Destroy the DFDLLib instance when it is no longer needed to release the allocated resources. This is automatically done when `dfdl_lib` goes out of scope, if not created with new.

It's important to notice that only one instance of the DFDLLib object is created and used. DFDLLib object instance must be created first of all to allow other DFDL4S++ classes proper usage. Due to the JNI nature on how native objects are stored in memory, if a new DFDLLib object is instantiated those objects are lost when a new JVM context is created.

3.5. DFDL4S++ Implementation

The available classes and methods for the DFDL4S C++ library are presented on the following sections. A complete reference for the C++ implementation is also distributed with the package as doxygen documentation.

Table 6 - Classes available and short description

Class name	Description
`DFDLLib`	The DFDLLib class provides the capability to interpret the contents of a binary file according to the specifications of a schema. DFDLLib object instantiation is mandatory in order to make use of DFDL4S++.
`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.
`ErrorIndicator`	The ErrorIndicator class stores error information related to an instance of Element
`CCSDSTimeTrait`	CCSDSTimeTrait implements a generic trait that can be use to check for traits on Elements. This concrete implementation allows checking for CCSDS Time Traits (as also storing its properties)
`DataSize`	The class DataSize represents the size of a data block. It provided byte and bit reference
`DFDL4SException`	The DFDL4SException is the class that represents an exception of type DFDL4SException. All other types of exceptions inherit this class, which provides a common API for all other exception types.
`Exception`	Exception is the base class of all types of exceptions thrown by the core Java library.
`IOException`	Class that represents an exception of type IOException
`InterruptedException`	Class that represents an exception of type InterruptedException
`ErrorLoadingException`	Class that represents an exception of type ErrorLoadingException

3.5.1. DFDLLib

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

Table 7 - List of operations of the DFDLLib class

Operation name	Input	Output	Description
DFDLLib (constructor)	std::string std::string	DFDLLib	This method initialises the DFDLLib: sets the UTC TAI conversion data.
interpretDocument	std::string std::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	std::string unsigned char* size_t	Document	Interprets the contents of a binary file according to the specifications of a schema file, memory block containing the data to be accessed and the number of elements of the memory block. Returns the element (document) containing all element items available in the binary file.
appendElements	Document * std::string int int unsigned char std::string	void	This method adds new elements to a document based on data generation parameters
appendElements	Document * std::string size_t	void	This method adds new elements to a document based on raw data.
getVersion	void	std::string	The version number and release date of the library.
createNewDocument	std::string	Document	Create new Document containing the specific schema file
storeDocument	Document * std::string	void	Stores the document in the specific path file.
storeDocument	Document *	std::vector<unsigned char>	Stores the document in a byte[] and return the byte[] with the document data.

Operation name	Input	Output	Description
storeDocument	Document * std::string int int	void	Export a given range of packets to a given file.

3.5.2. Document

The Document class represents the root of the domain element that is used to structure the binary data.

Table 8 - List of operations of the Document class

Operation name	Input	Output	Description
size	-	long	Access the size of the stored data.
getCleanAlignedData	DataSize* DataSize*	std::vector <unsigned char>	Retrieve the array of bytes storing information of the given element. The content of the byte array is shifted to guarantee that significant bits begin at the least significant bit of the array. Returns the byte array containing data (in big endian) for the element
getRawData	DataSize* DataSize*	std::vector <unsigned char>	Retrieve the array of bytes storing information of the given element. According to the element size/offset, this byte array may contain left and/or right bits that are not relevant. Returns the byte array containing data (in big endian) for the element
childCount	void	int	Returns the number of children of the element.
childAt	int	Element	Access the child at the given index. Returns the requested child element.

Operation name	Input	Output	Description
childAt	int std::string std::string	Element	Access the child at the given index, with the possibility of evaluated internal ISP in TF files or internal TF in CADU files.
close	void	void	Close the document, releasing all associated resources.
createElement	std::string	Element	Create a root element
createElement	std::string Element*	Element	Create a child element of the given element
createElementTime	std::string Element* std::string	Element	Create a child element of the given element with the specific String Time value
createElementInteger	std::string Element* Long long	Element	Create a child element of the given element with the specific BigInteger type value
createElementFloat32	std::string Element* float	Element	Create a child element of the given element with the specific Float type value
createElementFloat64	std::string Element* double	Element	Create a child element of the given element with the specific Double type value
createElementBytes	std::string Element* std::vector<unsigned char>	Element	Create a child element of the given element with the specific byte[] type value
validate	-	void	Validate "handcrafted" document

3.5.3. Element

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

Table 9 - List of operations of the Element class

Operation name	Input	Output	Description
absoluteName	void	std::string	Access the absolute name of the element.
name	void	std::string	The name of the element

Operation name	Input	Output	Description
uniqueName	void	std::string	Get the Element name, including the suffix
absoluteUniqueName	void	std::string	Get the absolute path to the Element, containing the root node.
parent	void	Element	Get the parent of the element
root	void	Element	Get the packet element (below document) that contains this element.
size	void	DataSize	Get the DataSize object containing the expected size (number of bytes and bits) of the Element according to the schema.
sizeAvailable	void	DataSize	Get the DataSize object containing the size available (number of bytes and bits) of the Element according to the schema.
offset	void	DataSize	Get the DataSize object containing the number of bytes and bits corresponding to the offset of the element
getValueAsRepresentation	void	std::string	Access the value of the element according the representation (if dmx:representation is present in schema) or intrinsic type.

Operation name	Input	Output	Description
getValueAsRepresentation	REPRESENTATION_TYPE*	std::string	Access the value of the element according to the given representation type (more information about representation type can be found in section 6).
childAt	int	Element	Access the child at the given index
childAvailableCount	void	int	Access the number of children available
getError	void	ErrorIndicator	Access error indicator related to this element
getValueHexadecimal	void	std::string	Access the value of the element (according to the 'HEXADECIMAL' representation)
getIntrinsicType	void	std::string	Gets the element intrinsic type (xsd type)
getRangeMaximum	void	long long	For XSD_TYPES BYTE, SHORT, INT or LONG, return the maximum type value based on size
getRangeMinimum	void	long long	For XSD_TYPES BYTE, SHORT, INT or LONG, return the minimum type value based on size
getValueFloat32	void	float	Access the value of the element (according to the 'FLOAT_32' representation)

Operation name	Input	Output	Description
getValueFloat64	void	double	Access the value of the element (according to the 'FLOAT_64' representation)
getValueInteger	void	long long	Access the integer value of the element
getValueTime	void	std::string	Access the value of the element (according to the 'TIME' representation)
retrieveRawData	void	std::vector <unsigned char>	Access the raw data of the element. Notice that data is not word aligned and may require cleaning of leading and trailing bits.
retrieveRawData	int int	std::vector <unsigned char>	Access the raw data of the element according to the desired size and offset. Notice that data is not word aligned and may require cleaning of leading and trailing bits.
getValueBytes	void	std::vector <unsigned char>	Access the clean and aligned data of the element
setValueBytes	std::vector <unsigned char>	void	Update the raw data of the element
setValueFloat32	float	void	Set the value of the element (according to the 'FLOAT_32' representation)
setValueFloat64	double	void	Set the value of the element (according to the 'FLOAT_64' representation)

Operation name	Input	Output	Description
setValueInteger	long long	void	Set the value of the element (according to the 'INTEGER' representation)
getValueBinary	void	std::string	Access the value of the element (according to the representation specified in the binary definition)
setValueTime	std::string	void	Set the value of the element (according to the 'TIME' representation)
propertyValueGet	String	std::string	Access the value of a property with a given name
is	Element*	bool	Check if this element is an ElementTrait instance
evaluate	Element*	std::string	Gets the element for a given path expression
evaluateBoolean	Element*	std::string	Evaluate an expression known to be a boolean value
evaluateInteger	Element*	std::string	Evaluate an expression known to be an integer value

3.5.4. CCSDSTimeTrait

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

Table 10 - List of operations of the CCSDSTimeTrait class

Operation name	Input	Output	Description
check	Element*	bool	Check for traits on Element
getType	void	std::string	Get the CCSDSTime type
getSize	std::string	long	Get the size in bits associated with the given property

3.5.5. ErrorIndicator

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

Table 11 - List of operations of the ErrorIndicator class

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

3.5.6. DataSize

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

Table 12 - List of operations of the DataSize class

Operation name	Input	Output	Description
DataSize	long int	DataSize	Constructor that allow the user create a new instance of DataSize object containing the given number of bytes and bits
getNrBytesTotal	-	long	Get the number of bytes necessary to store a block of data of size this.
getNrFullReservedBytes	-	long	Get the number of full reserved bytes of the element
getNrBitsOfTheLastByte	-	int	Access the number of bits of the last byte reserved for the element
getNrBitsTotal	-	long	Access the total number of bits covering the data of this size

Operation name	Input	Output	Description
isEqualTo	DataSize*	Boolean	Checks if this object equals another size instance
isGreaterThan	DataSize*	Boolean	Checks if this object is greater than another size instance
isLessThan	DataSize*	Boolean	Checks if this object is less than another size instance
accumulate	long int	Void	Accumulate this size object with another size object

3.5.7. Report

The Report interface defines the interaction methods related to reporting. The Report is divided in 2 sections:

1. Summary Entries
    a. The user can add summary into the report (For example: number of isp analysed, the number o CRC errors found in the isp file, ...)
2. Event Entries
    a. The user can add entry events into the report. These events should report specific properties of a given element. (For example: reporting the presence of CRC error in a specific element)

Table 13 - List of operations of the DataSize class

Operation name	Input	Output	Description
getEvents	-	std::vector<Entry>	Return Entry events contained in the report.
getSummary	-	Entry	Return the Entry summary contained in the report.
report	Document* ReportProperties*	Report	Generate the report of the given Document using the ReportProperties.
report	Document* ReportProperties* Int Int	Report	Generate the report of the given Document between the begin and end element, using the Report Properties.

3.5.8. ReportProperties

The ReportProperties class allows the user to create the properties needed to generate the report for each type of document (ISP, TF and CADU).

Table 14: List of operations of the ReportProperties class

Operation name	Input	Output	Description
buildPropertiesISP	std::string std::string std::string std::string	ReportProperties	Build properties necessary to generate a complete report for ISP products.
buildPropertiesTF	std::string std::string std::string std::string std::string std::string std::string	ReportProperties	Build properties necessary to generate report for TF products.
buildPropertiesCADU	std::string std::string std::string	ReportProperties	Build properties necessary to generate report for CADU products.
with	std::string std::string	ReportProperties	Add additional properties to be used in the report generation.

3.5.9. Entry

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

Table 15: List of operations of the Entry class

Operation name	Input	Output	Description
addField	std::string std::string	void	Add field to detailed / describe the entry in the report.
getNumberOfFields	-	int	Gets the number of fields contained by the entry.
getNameOfFieldAt	Int	std::string	Get the name of the field at the given index. If the name of the field starts with "*" the field is mandatory to process/generate the report.
getValueOfFieldAt	Int	std::string	Get the value of the field at the given index.

3.5.10. CRCDecoder

The CRCDecoder class provided functionality for validating the CRC of a data frame.

Table 16: List of operations of the CRCDecode class

Operation name	Input	Output	Description
`decode`	`std::vector <unsigned char>`	`int`	Decode the data frame using the algorithm CRC16 CCITT false (with initial value 0xFFFF and polynomial 0x1021) following the description provided in [RD.6] - section 9.4 - and [RD.7].

3.5.11. ReedSolomonDecoder

The ReedSolomonDecoder class provided functionality for validating the ReedSolomon of a data frame

Table 17: List of operations of the ReedSolomonDecoder class

Operation name	Input	Output	Description
`decode`	`std::vector <unsigned char>` `bool`	`RSDecodeState`	Do Reed Solomon decoding on the given data frame, returning a RSCodeState. • Calculate the syndrome vector from the received RS codeword. • Calculate the coefficients of the error locator polynomial. • Calculate the roots of the error locator polynomial. • Calculate the error magnitudes. • Correct the symbols in error with the previously calculated information. The algorithm used is described in [RD.6] – section 5.

3.5.12. RSDecodeState

The RSDecodeState class contains the detailed result of Reed-Solomon decode.

Table 18: List of operations of the RSDecodeState class

Operation name	Input	Output	Description
`getState`	-	`int`	Get the state of the Reed Solomon decode: OK=0, CORRECTED=-1 or UNCORRECTABLE=-2

Operation name	Input	Output	Description
getErrorInfo	-	std::vector<RSError Info>	Get the list containing information about the errors detect during the decode (empty list is returned if no error is detected).

3.5.13. RSErrorInfo

The RSErrorInfo contains the details of a Reed-Solomon errors.

Table 19: List of operations of the RSErrorInfo class

Operation name	Input	Output	Description
getLevel	-	Int	Gets the level of interleaving of the error.
getLocation	-	Int	Gets the location of the error within the frame (symbol offset).
getErrorValue	-	Byte	Gets the uncorrected value of the error.
getCorrectValue	-	Byte	Gets the expected/corrected value.

3.5.14. DFDL4SException

DFDL4SException is a C++ only base class for all types of exceptions, including those thrown by the Java library, providing an interface common not only to all library specific exception types but also to the standard std::exception, which it inherits.

Besides this generic role as common interface (base class), it is also reserved for C++ specific errors, that is, not originated in the core Java library. It is the only exception class that provides a public constructor, because the instances of all other types, which inherit Exception, are created internally by the library and meant only to represent their corresponding Java exception types.

To handle the C++ layer specific exceptions, use DFDL4SException in the exception handlers (catch clauses); to handle Java exceptions passed from the Java layer to the C++ layer, use any of the Exception hierarchy of classes (see 3.5.15).

Table 20 - List of operations of the DFDL4SException class

Operation name	Input	Output	Description
DFDL4SException (constructor)	void	DFDL4SException	DFDL4SException default constructor
DFDL4SException (constructor)	char *	DFDL4SException	DFDL4SException constructor with a given message
what	void	char *	Access for the exception message

3.5.15. Exception

Exception is the wrapper class of all native java Exception, and base class (superclass) of all wrapper classes of the respective exception types thrown by the Java library: IOException, InterruptedException and ErrorLoadingException. It inherits DFDL4SException: see 3.5.7.

Table 21 - List of operations of the Exception class

Operation name	Input	Output	Description
what	void	char *	Access for the exception message

3.5.16. IOException

Class that represents a Java exception of type IOException. It inherits Exception.

Table 22 - List of operations of the IOException class

Operation name	Input	Output	Description
what	void	char *	Access for the exception message

3.5.17. InterruptedException

Class that represents a Java exception of type InterruptedException. It inherits Exception.

Table 23 - List of operations of the InterruptedException class

Operation name	Input	Output	Description
what	void	char *	Access for the exception message

3.5.18. ErrorLoadingException

Class that represents a Java exception of type ErrorLoadingException. It inherits Exception.

Table 24 - List of operations of the ErrorLoadingException class

Operation name	Input	Output	Description
what	void	char *	Access for the exception message
getOptions	void	std::vector<std::string>	Get the data definition options offered to the user when there is ambiguity
getSource	void	std::string	Returns the source of the exception (usually the file path)

3.5.19. DFDL4SJException

Class that represents a Java exception of type DFDL4SJException. It inherits Exception.

Table 25 - List of operations of the DFDL4SJException class

Operation name	Input	Output	Description
what	void	char *	Access for the exception message
getOptions	void	std::vector<std::string>	Get the data definition options offered to the user when there is ambiguity
getSource	void	std::string	Returns the source of the exception (usually the file path)

3.5.20. DFDL4S_JAVA_OPTS

DFDL4S++ allows customizing the options passed to the underlying JVM by defining the environment variable: `DFDL4S_JAVA_OPTS`. The environment variable is automatically extracted by the DFDL4S++-based application, and the options passed as the parameters for the internal JVM.

As an example, the user can define the variable as shown in the following table:

Table 26 - Example of DFDL4S_JAVA_OPTS definition

DFDL4S_JAVA_OPTS Environment Variable
`DFDL4S_JAVA_OPTS="-Xms128m -Xmx512m -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5055"`

In particular, it allows the user to define the heap and stack size of the JNI virtual machine. Also, the last option, allows the user to enable debugging of the Java code.

3.5.21. Different functionalities between Java and C++ version

The user can find some different functionalities between C++ and Java version of DFDL4S library. The list below presents the existent differences between the two version.

1. Implement/extend existent interfaces/classes
    - DFDL4S++ does not handle extensions based on inheritance from existing abstract classes (e.g. ElementTrait), as it is possible in DFDL4S
2. Callback functionality
    - DFDL4S++ does not support Callback mechanism
    - The following DFDL4S method overloads are not available in DFDL4S++:

Operation/Method	Remark
DFDLLib::interpretDocument (*const std::string* schemaFile,* *const std::string* dataFile,* *InterpreterMonitor interpreterMonitor)*	*InterpreterMonitor* implementation requires callback capabilities
DFDLLib::interpretDocument (*const std::string* schemaFile,* *const unsigned char *data, const size_t length,* *InterpreterMonitor interpreterMonitor)*	*InterpreterMonitor* implementation requires callback capabilities
DFDLLib::interpretDocument (*const std::string* schemaFile,* *const std::string* dataFile,* *const unsigned char **referenceData,* *const unsigned char **maskData,* *const std::map<DataSize, DataSize> offsetCache,* *const std::string* ambiguityChoice,* *InterpreterMonitor interpreterMonitor)*	*InterpreterMonitor* implementation requires callback capabilities
DFDLLib::interpretDocument (*const std::string* schemaFile,* *const unsigned char *data, const size_t length,* *const unsigned char **referenceData,* *const unsigned char **maskData,* *const std::map<DataSize, DataSize> offsetCache,* *const std::string* ambiguityChoice,* *InterpreterMonitor interpreterMonitor)*	*InterpreterMonitor* implementation requires callback capabilities

End of Document