DFDL4S++ Library

Developer's Manual

Code :	S2G-DME-TEC-SUM113
Issue :	1.C
Date :	04/05/2018

	Name	Function	Signature
Prepared by	Joaquim Oliveira	Technical Manager	Joaquim Oliveira
Reviewed by	Rui Mestre	Project Manager	Rui Mestre
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-SUM113-1B.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

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] 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

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

This page intentionally left blank

3. GETTING STARTED

3.1. Introduction

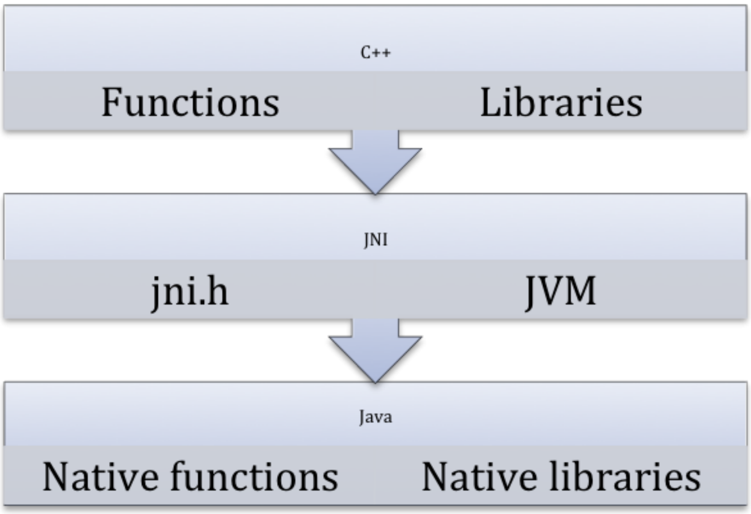
The DFDL4S++ is a library implemented in C++ that interprets the contents of the communication channels between the signal-in-space and the ground systems apparatus. It interprets files containing concatenated CADUs, TFs or ISPs, and lists of available data units and allows reading the fields and associated 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 when a new version of the library is developed using native C++, the C++ layer is added to take the place of the JNI and Java layers (which will be removed).

It is important for the reader to notice that on the current DFDL4S++ version not all of the DFDL4S API is available. Check the section 3.4 to see which of the methods are available on the DFDL4S++.

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-CPP-1.0-linux64.zip	Linux (64 bit)
DFDL4S-CPP-1.0-mac64.zip	Mac OS (64 bit)
DFDL4S-CPP-1.0-win64.zip	Windows (64 bit)

Table 4: Minimum System Requirements

Platform	Requirements	
Linux (64 bit)	RAM:	2 GB
	Disk Space:	10 MB
	Dependencies:	G++ compiler 64bit (v4.8+) Oracle JDK 1.8 64 bit
Mac OS (64 bit)	RAM:	2 GB
	Disk Space:	10 MB
	Dependencies:	Apple LLVM v8.1.0 (clang-802.0.42) 64 bit Oracle JDK 1.8 64 bit
Windows (64 bit)	RAM:	2 GB
	Disk Space:	10 MB
	Dependencies:	Microsoft Visual Studio 14.0+ Express 64 bit Oracle JDK 1.8 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 3.4 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. Set an environment variable pointing to the home of the Oracle JDK installation - JAVA_HOME (see below examples for each of the platforms)
   a. On Linux open the console and type:

   ```
   > export JAVA_HOME=/usr/lib/jvm/java-8-oracle
   ```

   b. On Mac open the terminal and type:

   ```
   > export JAVA_HOME=
   /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home
   ```

   c. On Windows open the Command Prompt and type:

   ```
   > set JAVA_HOME=C:\Program Files\jdk1.8.0_91
   ```

Hint: If you don't know where is your Oracle 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-8-oracle/bin/javac
```

The correct path for JAVA_HOME is:

```
/usr/lib/jvm/java-8-oracle
```

b) One Mac open the terminal and type:

```
> /usr/libexec/java_home -V
Matching Java Virtual Machines (1):
1.8.0_91, x86_64:  "Java SE 8"
/Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home
```

The correct path for JAVA_HOME is:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home
```

c) On Windows open the Command Prompt and type:

```
> for %i in (javac.exe) do @echo.   %~$PATH:i
C:\Program Files\jdk1.8.0_91\bin\javac.exe
```

The correct path for JAVA_HOME is:

```
C:\Program Files\jdk1.8.0_91
```

2. On Windows, you can previously select the version of Visual Studio in `runExample.bat`, as explained in the script; otherwise the default will be used. Then run the script that compiles and runs the example code:

   a. On Linux or Mac on the same console, type:

   ```
   sh runExample.sh
   ```

   b. On Windows on the same Command Prompt, type:

   ```
   runExample.bat
   ```

3.3. Example

With the DFDL4S++ package we include an example (see the Example.cpp file) to demonstrate some usages of the read and write functionalities provided by the DFDL4S++ lib.

In particular, the example 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.

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.

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

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

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

   ➢ `BinaryBuffer elementData = BinaryBuffer(dfdl_lib, ANNOTATION_SIZE + HEADER_SIZE);`

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. 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.

3.4. 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 5 - 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.
`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.
`ElementFinder`	The ElementFinder class provides the means to search the Element tree for specific values.
`Schema`	The Schema class represents a schema from a mission.
`DFDL4SException`	The DFDL4SException is the class that represents an exception of type DFDL4SException.

3.4.1. DFDLLib

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

Table 6 - List of operations of the DFDLLib class

Operation name	Input	Output	Description
`DFDLLib (constructor)`	`string` `string`	`DFDLLib`	This method initialises the DFDLLib: sets the UTC TAI conversion data.
`interpretDocument`	`string` `string`	`Document`	This method interprets the contents of a binary file according to the specifications of a schema file. Returns the element (document) containing all element items available in the binary file.

Operation name	Input	Output	Description

`interpretDocument`	`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.

`createDocument`	`string`	`Document`	This method generates a Document supported by a given file.
`getSchemaDefinition`	`string`	`Schema`	Access schema definition from a given mission definition file.

`appendElements`	`Document *`
	`string`
	`int`
	`int`
	`unsigned char`
	`string`
		`void`	This method adds new elements to a document based on data generation parameters

`appendElements`	`Document *`
	`string`
	`size_t`
		`void`	This method adds new elements to a document based on raw data.

`getVersion`		`string`	The version number and release date of the library.

3.4.2. Document

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

Table 7 - List of operations of the Document class

Operation name	Input	Output	Description
`childCount`		`int`	Returns the number of children of the element.
`childAt`	`int`	`Element`	Access the child at the given index. Returns the requested child element.
`close`		`void`	Close the document, releasing all associated resources.

3.4.3. Element

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

Table 8 - List of operations of the Element class

Operation name	Input	Output	Description
`absoluteName`		`std::string`	Access the absolute name of the element.
`getValueHexadecimal`		`std::string`	Access the value of the element (according to the 'HEXADECIMAL' representation)
`getIntrinsicType`		`std::string`	Gets the element intrinsic type (xsd type)
`getRangeMaximum`		`long long`	For XSD_TYPES BYTE, SHORT, INT or LONG, return the maximum type value based on size
`getRangeMinimum`		`long long`	For XSD_TYPES BYTE, SHORT, INT or LONG, return the minimum type value based on size
`getValueFloat32`		`float`	Access the value of the element (according to the 'FLOAT_32' representation)
`getValueFloat64`		`double`	Access the value of the element (according to the 'FLOAT_64' representation)
`getValueInteger`		`long long`	Access the integer value of the element
`getValueTime`		`std::string`	Access the value of the element (according to the 'TIME' representation)
`name`		`std::string`	The name of the element

Operation name	Input	Output	Description
`getValueBytes`		`std::vector <unsigned char>`	Access the clean and aligned data of the element
`setValueBytes`	`std::vector <unsigned char>`		Update the raw data of the element
`setValueFloat32`	`float`		Set the value of the element (according to the 'FLOAT_32' representation)
`setValueFloat64`	`double`		Set the value of the element (according to the 'FLOAT_64' representation)
`setValueInteger`	`long long`		Set the value of the element (according to the 'INTEGER' representation)
`getValueAsString`		`std::string`	Access the value of the element (according to the representation specified in the binary definition)
`setValueTime`	`std::string`		Set the value of the element (according to the 'TIME' representation)

3.4.4. ElementFinder

The ElementFinder class provides the means to search the Element tree for specific values.

Table 9 - List of operations of the ElementFinder class

Operation name	Input	Output	Description

`getElement`	`DFDLLib*`
	`const Element *`
	`const std::string&`
		`Element`	Gets the element for a given packet and expression

3.4.5. Schema

The Schema class represents a schema from a mission.

No operations are supported for this class.

3.4.6. DFDL4SException

The DFDL4SException is the class that represents an exception of type DFDL4SException.

Table 10 - List of operations of the DFDL4SException class

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

3.5. Traceability between Java and C++ implementations

The following tables provide the traceability matrixes between the Java and the C++ implementation of the DFDL4S library.

Table 11 - Traceability matrix for DFDLLib class

DFDLLib	DFDL4S Java	DFDL4S C++
`initLib(String)`	X	X[3]
`interpretDocument(String, String)`	X	X
`interpretDocument(String, byte[])`	X	X[4]
`createDocument(String)`	X	X
`createDocument(File)`	X	
`appendElements(Document, String, int, int, byte, String)`	X	X
`appendElements(Document, BasicSchema, byte[])`	X	X
`getSchemaDefinition(String)`	X	X
`getVersion()`	X	X

Table 12 - Traceability matrix for Document class

Document	DFDL4S Java	DFDL4S C++
`childAdd(DocumentItem)`	X	
`childAt(int)`	X	X
`childAt(int, boolean)`	X	
`childAtOffset(int)`	X	
`childCount()`	X	X
`close()`	X	X
`elementContains(Element, offsetInFile)`	X	
`evaluate()`	X	
`export(Path, int, int)`	X	
`extend(Document)`	X	
`getElement()`	X	
`getProvider()`	X	

[3] The initLib is implied on the DFDLLib constructor.

[4] One additional argument - the array length – but the functionality is the same.

Document	DFDL4S Java	DFDL4S C++
`getSyncHistory`	X	
`setProvider(DMXDataProvider)`	X	
`setSyncHistory(SyncErrorContainer)`	X	

Table 13 - Traceability matrix for Element class

Element	DFDL4S Java	DFDL4S C++
`absoluteName()`	X	X
`childAdd(Element)`	X	
`childAt(int)`	X	
`childAvailableCount()`	X	
`childByName(String)`	X	
`childCount()`	X	
`childStriptoHeader()`	X	
`countErrors(boolean, boolean)`	X	
`document()`	X	
`getChildErrors()`	X	
`getChildErrors(boolean)`	X	
`getError()`	X	
`getHexadecimalValue()`	X	X[5]
`getIntrinsicType()`	X	X
`getRangeMaximum()`	X	X
`getRangeMinimum()`	X	X
`getValueBytes ()`	X	X
`getValueFloat32()`	X	X
`getValueFloat64()`	X	X
`getValueInteger()`	X	X
`getValueTime()`	X	X
`getValueProperties()`	X	
`hasError()`	X	
`hasErrors(boolean, boolean)`	X	
`hasNonSevereError()`	X	
`hasSevereError()`	X	

[5] Renamed getValueHexadecimal in DFDL4S C++.

Element	DFDL4S Java	DFDL4S C++
`isCheckeable()`	X	
`name()`	X	X
`offset()`	X	
`parent()`	X	
`path()`	X	
`propertyAdd(DMXProperty)`	X	
`propertyGet(String)`	X	
`propertyHas(String)`	X	
`propertyValueGet(String)`	X	
`retrieveCleanData()`	X	
`retrieveRawData()`	X	
`retrieveRawData(int, int)`	X	
`root()`	X	
`setchildCount(int)`	X	
`setData(String)`	X	
`setData(String, String)`	X	
`setDocument(DMXDocument)`	X	
`setError(String, boolean, boolean)`	X	
`setOffset(DMXSize)`	X	
`setValueBytes(byte[])`	X	X
`setValueFloat32(Float)`	X	X
`setValueFloat64(Double)`	X	X
`setValueInteger(BigInteger)`	X	X
`setValueTime(String)`	X	X
`value()`	X	X[6]

Table 14 - Traceability matrix for ElementFinder class

ElementFinder	DFDL4S Java	DFDL4S C++
`elementAtOffset(Document, int, int, boolean)`	X	
`findNext(Document, int, String, String, SearchMonitor, boolean)`	X	
`findPrevious(Document, int, String, String, SearchMonitor, boolean)`	X	
`getElement(Element, String)`	X	X

[6] Renamed getValueAsString in DFDL4S C++.

ElementFinder	**DFDL4S Java**	**DFDL4S C++**
`getValueBoolean(Element, String)`	X	

End of Document