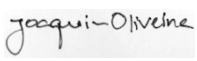
DFDL4S++ Library

Developer's Manual

Code : S2G-DME-TEC-SUM113
Issue : 1.E
Date : 17/12/2018

	Name	Function	Signature
Prepared by	Joaquim Oliveira	Project Manager	
Reviewed by	Joaquim Oliveira	Project Manager	
Approved by	Antonio Gutiérrez	Technical Consultant	
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

RESTRICTED

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-SUM113-1E.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

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

This page intentionally left blank

3. GETTING STARTED

3.1. Introduction

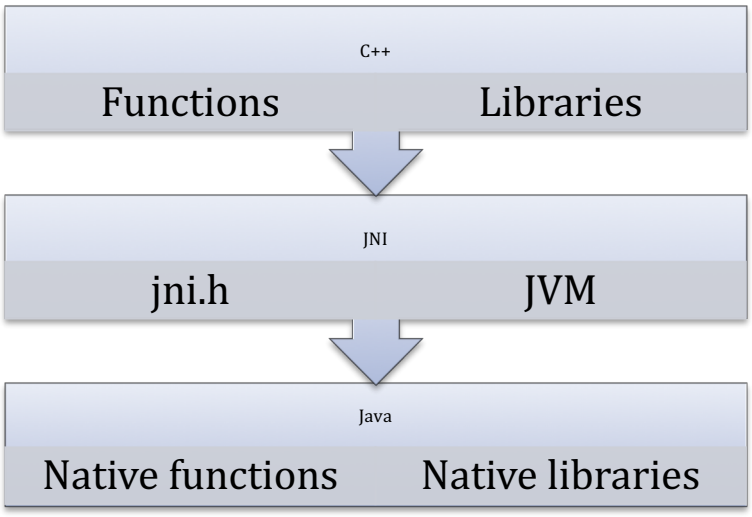
The DFDL4S++ is a library implemented in C++ that interprets the contents of the communication channels between the signal-in-space and the ground systems apparatus. It interprets files containing concatenated CADUs, TFs or ISPs, and lists of available data units and allows reading the fields and associated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

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-linux64.tar.gz	Linux (64 bit)
dfdl4s++- X.Y.Z-mac64.tar.gz	macOS (64 bit)
dfdl4s++- X.Y.Z-win64.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 (v4.8+) Oracle JDK 1.8 64 bit
macOS (64 bit)	RAM: Disk Space: Dependencies:	2 GB 10 MB Apple LLVM v8.1.0 (clang-802.0.42) 64 bit Oracle JDK 1.8 64 bit
Windows (64 bit)	RAM: Disk Space: Dependencies:	2 GB 10 MB 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) On 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:

   ➢ `Document document_1 = dfdl_lib.createDocument( filename );`

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.
`BinaryBuffer`	A binary buffer is a linear finite sequence of raw data elements.
`ErrorIndicator`	The ErrorIndicator class stores error information related to an instance of Element
`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

3.4.1. 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)	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.
createDocument	std::string	Document	This method generates a Document supported by a given 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.

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	void	int	Returns the number of children of the element.

Operation name	Input	Output	Description
childAt	int	Element	Access the child at the given index. Returns the requested child element.
close	void	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	void	std::string	Access the absolute name of the element.
childAt	int	Element	Access the child at the given index
childAvailableCount	void	int	Access the number of children available
getChildErrors	void	std::vector<ErrorIndicator>	Access the list of all child errors
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)
`hasError`	`void`	`bool`	Indicates the presence of an error in the item
`hasSevereError`	`void`	`bool`	Indicates the presence of a severe error in the item or any of its children
`name`	`void`	`std::string`	The name of the element
`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.
`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)
`setValueInteger`	`long long`	`void`	Set the value of the element (according to the 'INTEGER' representation)

Operation name	Input	Output	Description
getValueAsString	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)

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	Element * std::string	Element	Gets the element for a given packet and expression

3.4.5. BinaryBuffer

A binary buffer is a linear, finite sequence of raw data elements, with different sizes and representing primitive types. Aside from its content, the essential properties of a buffer are its capacity and position:

- A buffer's *capacity* is the number of bytes it contains, i.e that could be written to. The capacity of a buffer is never negative and never changes.
- A buffer's *position* is the index of the next bit to be written.

A buffer's position is never negative and is never greater than its capacity in bits.

Table 10 - List of operations of the BinaryBuffer class

Operation name	Input	Output	Description
putDecimalNumber	long long int	void	Writes the N-bit two's complement representation of a given base 10 number into this buffer at the current position, and then increments the position by N
fill	int unsigned char	void	Fills a specified number of bytes with a given byte value, starting at current position.

Operation name	Input	Output	Description
asHexadecimal	void	std::string	Returns the hexadecimal representation of this buffer's content

3.4.6. 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
errorStatus	void	bool	Access the error status

3.4.7. 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 3.4.8).

Table 12 - 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

3.4.8. 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 3.4.7.

Table 13 - List of operations of the Exception class

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

3.4.9. IOException

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

Table 14 - List of operations of the IOException class

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

3.4.10. InterruptedException

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

Table 15 - List of operations of the InterruptedException class

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

3.4.11. ErrorLoadingException

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

Table 16 - List of operations of the ErrorLoadingException class

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

4. ANNEX A: DFDL4S++ Library Build Instructions

The DFDL4S++ library binary packages are released with everything necessary to compile and link executables that wish to use the library. This includes, besides the library binary file, C++ header and documentation files, and also an example that the user can easily build and run, namely through the scripts also provided. Sections 3.2 and 3.3 list the binary packages distributed for each operating system and describe both the library and the example usages.

Although not necessary nor recommended for most purposes, the DFDL4S++ library itself can also be built for any of the supported operating systems. The source package dfdl4s++-X.Y.Z-src.tar.gz, common for all systems, contains the source and configuration files needed to build the library with CMake. Additional dependencies are required:

- CMake 3.2+
- doxygen 1.8.13+

As in the case of the example, the following dependencies must also be installed:

- Oracle JDK 1.8 64 bit
- G++ compiler 64bit (v4.8+) (Linux)
- Apple LLVM v8.1.0+ (clang-802.0.42+) 64 bit (mac OS)
- Visual Studio 14.0+ 64 bit (Windows)

These dependencies must be installed in a way such that CMake is able to find them. Standard installations are usually found, but please see the CMake documentation for any particular or unusual configuration of the dependencies. CMake displays informative messages about missing or not found packages. CMake itself should be found in the PATH, otherwise the provided scripts (see below) should be manually changed to specify the CMake binary full path.

An out-of-source build is performed, meaning that auxiliary and target build outputs are placed into dedicated directories (respectively "/build" and "/bin"), both of them out of the directory containing the sources. The "/build" directory can be safely deleted after a successful build, all the files necessary to use the library being inside the "/bin" directory. The source directory remains untouched.

Decompressing the source package in any directory created by the user (this is referred to as the "root" directory) a "/source" sub-directory is created and 2 scripts, build.bat (to be used in Windows) and build.sh (to be used in Linux and Mac OS) will be found in the root directory.

After opening a terminal, change to the root directory and run the script for the respective operating system:

$ ./build.sh (Linux, macOS)

$ build.bat (Windows)

In Windows an additional step is necessary before invoking the script, to select one of the supported Visual Studio versions. As can be seen in the instructions at the top of build.bat, the value of the script's variable "vcvarsall_DIR" should be changed to the corresponding directory of the required Visual Studio installation (the directory where vcvarsall.bat is located). Both scripts contain at the top a small set of build instructions that can be reviewed for quick details.

The build outputs will be found in one of the following sub-directories of the root directory:

- \bin\x64\Release\bin (Windows)
- /bin/x86_64/Release/bin (Linux)
- /bin/x86_64/Release/dfdl4s++.app (Mac).

4.1. Testing

The DFDL4S++ build lifecycle includes a test phase to run unit tests using a library plugin on the CMake configuration:

- Goggle Test 1.8.1

The configuration of unit tests with CMake and Google Test is not automatic and multiple changes have to be made on the following locations:

- Tests source code location: src-test/
- Tests resources location: src-test/resources/
- CMake configuration to add unit test: CMakeLists.txt

Steps:

1. Add a unit test to tests source code location
2. Add test resources to tests resources location
3. Add test build configuration to CMake configuration

See below example on CMake configuration for the provided Tests.cpp:

1. Add executable
2. Link executable to library gtest and gtest_main (Google Test dependency)
3. Link executable to DFDL4S++ library and JVM library
4. On Windows, link executable to delayimp library
5. Add unit test to build (make test)
6. Copy test resources to build path
7. Copy lib folder to build path

```
###############
# Unit Tests
###############
add_executable(Tests src-test/Tests.cpp)

# Standard linking to gtest
target_link_libraries(Tests gtest gtest_main)
```

```
# Extra linking for the project
target_link_libraries(Tests dfdl4s++ ${JAVA_JVM_LIBRARY} )

if (WIN32)
    target_link_libraries( Tests delayimp )
endif()

# Run test with 'make test'
add_test(NAME DFDL4S_Test COMMAND Tests)

# add resources needed for Tests test case
install(
    DIRECTORY
    src-test/resources
    lib
    DESTINATION ${CMAKE_CURRENT_BINARY_DIR}
)
```

See below example on Tests.cpp to produce a test report in JUnit XML format:

1. Set test_report_output to produce JUnit XML report (xml:test_report_for_Tests.xml)

```
// catch google test init
int main(int argc, char **argv) {

  // set flag to output test report
  ::testing::GTEST_FLAG(output) = test_report_output;

  // continue google test init
  ::testing::InitGoogleTest(&argc, argv);

  // run tests
  return RUN_ALL_TESTS();
}
```

The unit test output will create a report on the following path

- Windows: \build\x64\Release\$test_report_output
- Linux: /build/x86_64/Release/$test_report_output
- Mac: /build/x86_64/Release/$test_report_output

  Where $test_report_output is the value of the variable defined in the test (e.g. "xml:test_report_for_Tests.xml")

End of Document