Error Generation Libraries for the open Simulation Framework

OSFEG

DEVELOPER'S MANUAL

Code	OPENSF-DMS-OSFEG-DM
Issue	1.4
Date	17/12/2020

	Name	Function	Signature
Prepared by	Enrique del Pozo Gonzalo Vicario Javier Martín	Project Engineers	
Reviewed by	Federico Letterio	Project Manager	
Approved by	Federico Letterio	Project Manager	
Signatures and approvals on original			

DEIMOS Space S.L.
Ronda de Poniente, 19, Edificio Fiteni VI, 2-2ª
28760 Tres Cantos (Madrid), SPAIN
Tel.: +34 91 806 34 50 / Fax: +34 91 806 34 51
E-mail: deimos@deimos-space.com

This page intentionally left blank

Document Information

Contract Data	
Contract Number:	4000120205/17/NL/CT/mg
Contract Issuer:	ESA/ESTEC

Internal Distribution		
Name	Unit	Copies
Antonio Gutierrez	Head of the Ground Segment Business Unit	
Internal Confidentiality Level (DMS-COV-POL05)		
Unclassified ☑	Restricted ☐	Confidential ☐

External Distribution		
Name	Organisation	Copies
Michele Zundo Marcos Bento Montserrat Pinõl Sole	ESA/ESTEC	1

Archiving	
Word Processor:	MS Word 2016
File Name:	OPENSF-DMS-OSFEG-DM-014.doc

Document Status Log

Issue	Change description	Date
1.0	First issue of this document	04/06/09
1.1	Updated for openSF V3	22/11/13
1.2	New build system with CMake	15/12/17
1.3	Updated for OSFEG v1.4	14/12/18
1.4	Updated for OSFEG v.1.4.1 ❑ The value of "float" elements may now be provided as text content. ❑ Updated XML examples for the different perturbation types ❑ Updated environment definition (compiler, C++ standard) to match OSFI ❑ Updated example C++ sources	17/12/20

Table of Contents

1. Introduction....7
1.1. Purpose....7
1.2. Scope....7
1.3. Acronyms and Abbreviations....7
2. RELATED DOCUMENTS....9
2.1. Applicable Documents....9
2.2. Reference Documents....9
2.3. Standards....9
3. GETTING STARTED....10
3.1. Introduction....10
3.2. Conventions used in this Manual....10
3.2.1. <OSFEG_DIR>....10
3.3. Initial Requirements....10
3.3.1. Hardware requirements....11
3.3.2. Software requirements....11
3.4. Installation....12
3.4.1. Build Instructions....12
4. openSF ERROR GENERATION Libraries....14
4.1. Error definition files....14
4.1.1. Definition of perturbations....14
4.1.2. Available error functions....14
4.1.2.1. Deterministic Functions....15
4.1.2.2. Sampling Functions....17
4.1.2.3. Nondeterministic Functions....18
4.1.2.4. Binary and Composite Operations....20
4.1.3. Example file....21
4.2. Process logic....21
4.3. Examples of use....22
4.3.1. C++ Programming Language....22
4.3.2. C++ Compilation and Execution process....23

List of Tables

Table 1: Applicable documents 9
Table 2: Reference documents 9
Table 3: Standards 9
Table 4: Suggested compilers for sources 11
Table 5: System pre-requisites 11
Table 6: Recommended utilities 11

List of Figures

Figure 1: OSFEG distribution 12

1. INTRODUCTION

This project concerns the definition and development of libraries to ease the generation of analytical and stochastic perturbations, or a combination of them, in the models that will be integrated into the open Simulation Framework (openSF) system. It will be applicable to other projects that imply the use of openSF.

1.1. Purpose

The objective of this document is to provide a detailed description and an operation manual of the error generation libraries used during the development and deployment of the models implied in a simulation creation process.

The intended readerships for this document are model developers and scientists that are in charge of integrate those models into the open Simulation Framework.

This document is also useful to software engineers responsible of the testing stage.

1.2. Scope

This document shows a detailed description of the libraries 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.

This document contains the following sections:

- An introduction (current section 1) for giving a quick overview of the project;
- A list of related documents to provide a documentary background (section 2)
- An introduction to the libraries, installation and linking instructions (section 3)
- A description of the architecture, the process logic and some examples of use. It also includes the coding guidelines (section 4)

1.3. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

Acronym	Description
AD	Architectural Design Applicable Document
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DMS	DEIMOS Space
FFI	Foreign Function Interface

Acronym	Description
GUI	Graphical User Interface
HW	Hardware
I/O	Input/Output
ICD	Interface Control Document
O/S	Operating System
RD	Reference Document
SOW	Statement Of Work
SPR	Software Problem Report
SUM	System User Manual
SW	Software
TBC	To Be Confirmed
TBD	To Be Defined / Decided
TER	Test Execution Record
TN	Technical Note
TP	Test Plan
TR	Test Report
TS	Technical Specification
UML	Unified Modelling Language
URD	User Requirements Document
V&V	Verification & Validation

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
[OSF-ICD]	openSF-DMS-ICD-001	OpenSF Interface Control Document	3.0
[AD 2]	EOP-SFP/2012-12-1686/PB/ag	Change Request for the openSF V3 activities description.	-

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
[OSFI-DM]	OSFI DM	OpenSF Integration Libraries – Developers Manual	1.19
[OSF-SUM]	openSF-DMS-SUM	OpenSF System User Manual	4.0

2.3. Standards

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

Table 3: Standards

Reference	Code	Title	Issue	Date
[XML]	www.w3.org/TR/xml11	Extensible Markup Language (XML) 1.1	Second Edition	29/09/06
[UML]	www.uml.org/#UML2.0	Unified Model Language (UML)	2.1	06/10/06
[C++]	ISO/IEC 14882:2011	C++ Standard	2011.3	09/11

3. GETTING STARTED

3.1. Introduction

In the frame of concept and feasibility studies for the Earth Observation (EO) activities, mission performance in terms of final data products needs to be predicted by means of so-called end-to-end (E2E) simulators.

A specific mission E2E simulator is able to reproduce all significant processes and steps that impact the mission performance and gets simulated final data products.

The open Simulation Framework (openSF) is a generic simulation framework product aimed to cope with these major goals. It provides end-to-end simulation capabilities that allow assessment of the science and engineering goals with respect to the mission requirements.

This openSF tool lets users to integrate and execute pieces of code, «models» that form the building blocks of a simulation process.

Typically those pieces of code, «models» are handled by openSF as simple executable programs with three interfaces, input, output and configuration.

Under this scenario appears the goal of performing a statistical analysis of the E2E simulator driven by the errors and perturbations present in the parameters involved in a simulation chain.

The Open Simulation Framework Error Generation Libraries (OSFEG from now on) will be used as a tool to ease the mathematical modeling of a perturbation within statistical analysis scenarios.

OSFEG offers to developers a well-documented interface to ease the modeling and generation of a perturbation over desired parameters.

The libraries provide an error-modeling interface based on a XML file definition and its correspondent implementation in C++. A detailed description will be seen in section 4.

3.2. Conventions used in this Manual

This chapter lists all the conventions used throughout this Developer's Manual

3.2.1. <OSFEG_DIR>

All through the contents of this Developer Manual, a "variable" called <OSFEG_DIR> is exhaustively used as a placeholder. The variable value points to the root folder that contains the OSFEG library, normally installed by CMake directly or unpacked from a previous build.

3.3. Initial Requirements

The OSFEG system is prepared to run in a hardware and software platform with the following requirements. These must be fulfilled before installing the distribution.

3.3.1. Hardware requirements

OSFEG is designed to be compatible with any platform that supports a standard C++11 compiler and run-time. In particular, it has been tested with:

- *Operating systems*: Linux, OSX, Windows
- *Architectures*: x86-64 (also known as AMD64 or Intel 64)

3.3.2. Software requirements

This is the list of suggested compilers for the sources.

Table 4: Suggested compilers for sources

Language	Compiler	Licensing	Distribution Site
C/C++	GNU C/C++ compiler v7.3 or superior	GNU General Public License, GNU Lesser General Public License	http://gcc.gnu.org

Nevertheless, developers can use their favorite compilers in each case.

Table 5 shows the system pre-requisites in order to build the OSFEG library.

Table 5: System pre-requisites

Component	Purpose	Licensing	Distribution Site
De-compressor	Extract files from release packaged in a compressed tarball	N/A	N/A
CMake 3.9 or higher	Build, test and pack the OSFI libraries	BSD 3-clause	Linux repository or https://cmake.org/

Table 6 shows a set of utilities that are recommended to build the OSFI libraries. If Xerces-C is not installed in the system, the OSFEG build system can be configured to download and build it automatically.

Table 6: Recommended utilities

Component	Purpose	Licensing	Distribution Site
Doxygen 1.8.13 or higher	Generate OSFI libraries documentation	GNU General Public License	Linux repository or https://www.doxygen.nl
Google Test	Generate and execute C++, C and Fortran tests	BSD 3-clause	Linux repository or https://github.com/ google/googletest
Xerces-C 3.2.0 or higher	Parse XML files	Apache License 2.0	https://xerces.apache.org/

3.4. Installation

OSFEG is distributed as source package. Figure 1 shows a high-level view of the contents of the distribution:

- The folder include contains the header files of the library
- The folder releng (release engineering) contains CMake configuration files
- The folder src contains the source files of the library
- The folder test contains a set of unit and integration test procedures that ensure the proper performance of the library

In addition, the distribution includes the main CMake make file and the license.

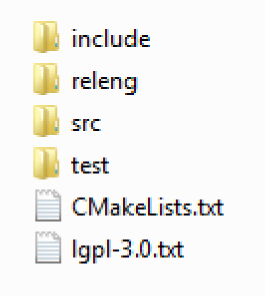

Figure 1: OSFEG distribution

3.4.1. Build Instructions

First, extract the integration libraries into the desired location and enter it:

```
$ tar -xvzf OSFEG_<version>_src.tar.gz
$ cd OSFEG
```

Next, create a folder where the products of the building process will be generated (e.g. build) and enter it:

```
$ mkdir build
$ cd build
```

The command that detects the system properties and creates the build system accepts a set of optional arguments that must be reviewed. First of all, the OSFEG libraries depend on Xerces v3.2.0. The default behavior of the build system is to look for the library in the user's system, but two optional arguments can be used to change the behavior:

- XercesC_DIR: it forces CMake to look for the Xerces library in the directory provided.
- BUILD_XERCES: if this option flag is set to ON, CMake will download and build Xerces 3.2.0 in the directory xerces/ExternalProject created in the build folder.

If the optional flag BUILD_SHARED_LIBS is set to ON (the default is OFF), the build process generates shared libraries. If not, static libraries are created.

If the boolean optional argument BUILD_DOC (which default value is ON) is set to OFF, the Doxygen-based documentation of the OSFEG libraries will not be created. It shall be remarked that the utility Doxygen itself must be installed in order to generate it.

Finally, the boolean argument BUILD_TESTING (which default value is ON) can be set to OFF in order to skip the test building process. Nevertheless, Google Test must be installed in order to build most of the tests.

The following example shows how to configure the OSFEG make files from the build folder created inside the OSFEG directory to generate the static libraries. It can be seen that the Xerces library is downloaded and built. It shall be remarked that the optional arguments are provided starting with "-D".

```
$ cmake -DBUILD_XERCES=ON ../
```

See the documentation for CMake for more configuration options, e.g. for the choice to create projects for different build systems (e.g. Xcode, Eclipse, etc.). Regardless of the choice of build system, once it is are configured, the selected OSFEG libraries can be built with the following command, executed from the build directory:

```
$ cmake --build .
```

The OSFEG tests can be launched with the following command executed from the build directory, or using the "test" target of the build system:

```
$ ctest
```

If the test execution has been successful, the "package" and "package_source" targets can be used to generate distributable versions of the binaries and sources. The first can also be achieved by running:

```
$ cpack
```

If the process has been successful, the package folder structure should be as follows:

- include: header files
- lib: dynamic or static libraries of OSFEG. In addition, the folder cmake/OSFEG/ contains the CMake configuration files.
- share: documentation of the libraries API in html format. This folder is not available if the documentation is not created.

It must be noted that the OSFEG binaries generated may have dependencies e.g. on Xerces or the C++ runtime of the compiler that was used. If module developers want their modules to be redistributable, they have the responsibility to include any dependencies in the package, especially the Xerces library used during the build process. Note that if the library was built alongside OSFEG (via the BUILD_XERCES flag) the generated products are located in the build directory in the folder xerces/ExternalProject/Install.

4. OPENSF ERROR GENERATION LIBRARIES

In this section, the following is given:

- A detailed description of the functions implemented within the error generation libraries.
- A complete set of examples of how to use the APIs and how to compile and run them.

4.1. Error definition files

In this section will be described the mathematical functions implemented within the error generation libraries. The libraries include the most used analytical and random functions to perturb parameters in E2E simulation modeling scenario.

The parameter perturbation functions are defined through an XML file. An example it is shown at the end of this section.

This section is especially relevant because the error definition file describes the mathematical behavior of the parameters perturbation. It is also included a detailed description of the variables involved in the function definition.

4.1.1. Definition of perturbations

The file is structure around the definition of one or more "error" items, which are introduced by XML elements with a tag name of "parameter". They are defined as follows:

- A "name" attribute, whose value is used to identify the error definition in the API.
- A single child element, which must be one of the error functions in section 4.1.2.

```
<parameter name="ERROR-ID">
  <error-function />
</parameter>
```

4.1.2. Available error functions

Error functions are defined by an element in the XML tree. The element tag name selects the type of function, while the definition parameters (e.g. mu and sigma for a Gaussian distribution) may be passed as attributes or as sub-elements, depending on the specific type.

Normally, parameters that are specified as sub-elements may themselves be of any type of error function, allowing e.g. the frequency of a sinusoidal function to be itself defined as a function growing linearly with time.

```
<error-function attributes…>
  <child-function />
  <child-function />…
</error-function>
```

However, in the case where the parameter is not a function but a constant, the special element "float" provides the value of the constant. It may take one of two forms: one where the value is in the content of the element, and another where it is specified as a "value" attribute. Both forms are recognized and parsed, but the former is preferred for new files as it is slightly shorter.

```
<float>123.456</float> <!-- Preferred -->
<float value="123.456" /> <!-- Legacy -->
```

4.1.2.1. Deterministic Functions

Deterministic functions are those whose value it is known in the entire time domain.

- **Affine**

This function calculates the perturbation as an affine value. An affine transformation consists in a linear transformation and a translation.

- $error = a_1 + a_0 * t$

```
<affine>
  <float>1< /float>  <!--  Linear Transformation Variable a0 -- >
  <float>1</float>    <!--  Translation Variable  a1 -- >
</affine>
```

- **Bias**

This function calculates the perturbation as a constant value.

```
<bias>
  <float>1</float>  <!-- Constant Value -- >
</bias>
```

- **Linear**

Calculates the perturbation as a linear value:

- $p = a * t$

This is a particular case of affine transformation when translation variable is equals to 0.

```
<linear>
  <float>1</float>  <!-- Linear Transformation Variable a -- >
</linear>
```

Parabolic

This function calculates the perturbation as a parabolic value.

- $p = a_0 + a_1 * t + a_2 * t^2$

```
<parabolic>
  <float>1</float>   <!-- a0 -->
  <float>1</float>   <!-- a1 -->
  <float>1</float>   <!-- a2 -->
</parabolic>
```

Polynomial

This function calculates the perturbation as a generic polynomial value. This function has as many float parameters as degrees of the desired polynomial plus one.

```
<polynomial>
  <float>1</float>   <!-- a0 -->
  <float>1</float>   <!-- a1 -->
     …
  <float>1</float>   <!-- a(n-2) -->
  <float>1</float>   <!-- a(n-1) -->
</polynomial>
```

Step

This function calculates the perturbation as step function.

- if simTime < t => $p = a_0$
- if simTime > t => $p = a_1$

```
<step>
  <float>3</float>    <!-- t -->
  <float>1</float>    <!-- a0 -->
  <float value="-1" />   <!-- a1 -->
</step>
```

Sinusoidal

Calculates the perturbation as sinusoidal function

- p = a * sin(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

```
<sinusoidal>
  <float>10</float>   <!-- Amplitude a -->
  <float>10</float>   <!-- Frequency f in Hz -->
  <float>0</float>     <!-- Angle phi in deg. -->
</sinusoidal>
```

- **Tangent**

Calculates the perturbation as tangent function

- p = a * tan(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

Remember that the tangent function have singularities when the angle evaluated is (+-)*n*pi/2.

```
<tangent>
  <float>10</float>   <!-- Amplitude a -->
  <float>1</float>     <!-- Frequency f in Hz -->
  <float>0</float>     <!-- Angle phi in deg. -->
</tangent>
```

4.1.2.2. Sampling Functions

Error Generation libraries implements three interpolation methods, linear, polynomial and spline sampling. In order to define the points of the interpolation there is a common set of variables that are listed below.

- xMin: Min value of the independent variable
- xMax: Max value of independent variable
- step: Increment between values of the independent variable

The number of points provided for the interpolation (nValues) must be exactly as needed to cover all predefined values of the independent variable, as follows:

$$\frac{xMax - xMin}{step} = nValues$$

- **Linear Sampling**

This function makes an interpolation with the given points assuming it follows a linear rule. In out of range values

```
<linearSampling xMin="1.0" xMax="5.0" step="1">
```

```
   <float>1</float>
   <float>3</float>
   <float value="5" />
   <float>7</float>
   <float>3</float>
</linearSampling>
```

- **Polynomial Sampling**

This interpolation method builds a polynomial grade n, being n the number of specified points. This interpolation minimizes the Least Square Error. Ref: Neville Method.

```
<polynomialSampling xMin="1.0" xMax="12.0" step="1">
   <float>1</float>
   <float>2</float>
   <float>1</float>
   <float>2</float>
   <float>1</float>
   <float>2</float>
   <float>1</float>
   <float>2</float>
   <float>1</float>
   <float>2</float>
   <float>3</float>
   <float>5</float>
</polynomialSampling>
```

- **Spline Sampling**

Interpolate the given “n” points with Cubic Splines Method.

```
<splineSampling xMin="1.0" xMax="20.0" step="2">
   <float>2</float>
   <float>3</float>
   <float>2</float>
   <float>3</float>
   <float>2</float>
   <float>3</float>
   <float>2</float>
   <float>3</float>
   <float>10</float>
   <float>2</float>
</splineSampling>
```

4.1.2.3. Nondeterministic Functions

These functions correspond to common random function implementation with seed management for testing purposes.

- **Beta Distribution**

This function generates random values with Beta function as probability density function.

```
<beta seed="1" v="2" w="5" xMin="0.0" xMax="1.0" />
```

- **Gamma Distribution**

This function generates random values with Gamma function as probability density function.

```
<gamma seed="1" location="0.0" scale="0.5" shape="9" />
```

- **Exponential Distribution**

This function generates random values with Exponential function as probability density function.

```
<exponential seed="1" a="1" b="1.5" />
```

- **Normal Distribution**

This function generates random values with Gaussian function as probability density function.

```
<normal seed="1" mu="100.0" sigma="10.0" />
```

- **Uniform Distribution**

This function generates random values following a Uniform Distribution.

```
<uniform seed="1" xMin="0" xMax="1" />
```

- **Poisson Distribution**

This function returns the perturbation as a generated random value with Poisson function as probability density function.

```
<poisson seed="1" mu="10" />
```

- **Truncated Gaussian Distribution**

This function returns the perturbation as a generated random value with Truncated Gaussian function as probability density function.

```
<truncatedGaussian seed="1" mu="0.5" sigma="0.2" xMin="0.4"
xMax="0.6" />
```

- **Uniform Discrete Distribution**

This function returns the perturbation as a generated random value with Uniform Discrete function as probability density function.

```
<uniformDiscrete seed="1" i="0" j="1" />
```

- **Distribution with custom Probability Density Function**

Returns the value of a random variable generated with a custom probability density function given. It is only recommended to use it by expert developers/scientists.

```
<customPDF seed="24" xMin="0.0" xMax="12.0" step="1">
  <float>7</float>
  <float>43</float>
  <float>21</float>
  <float>10</float>
  <float>2</float>
  <float value="6" />
  <float value="23" />
  <float value="31" />
  <float>7</float>
  <float>2</float>
  <float>7</float>
  <float>43</float>
  <float>21</float>
</customPDF>
```

4.1.2.4. Binary and Composite Operations

Error Generation Libraries implements the basics mathematical operations in binary mode. The operations implemented are:

- **Addition**
- **Subtraction**
- **Multiplication**
- **Division**
- **Exponentiation**
- **Root**

Composite operations consist of a deterministic function with one or more of its parameters following another function or binary operation.

4.1.3. Example file

An example of an error definition file with several parameters using both random and deterministic functions is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <parameter name="Affine and sinusoidal">
    <affine>
      <sinusoidal>
        <float>10<!-- A --></float>
        <float value="90" /> <!-- f -->
        <float><!-- phi -->0</float>
      </sinusoidal>
      <float value="5" />
    </affine>
  </parameter>
  <parameter name="Sinusoidal and beta">
    <sinusoidal>
      <beta seed="1" v="1.0" w="2.0" xMin="10.0" xMax="15.0" />
      <float>10</float>
      <float>0</float>
    </sinusoidal>
  </parameter>
  <parameter name="Composition ">
    <addition>
      <exponentiation>
        <sinusoidal>
          <float>10</float>
          <float value="90" />
          <float>0</float>
        </sinusoidal>
        <float>2</float>
      </exponentiation>
      <subtraction>
        <sinusoidal>
          <float value="40" />
          <float value="90" />
          <float value="5" />
        </sinusoidal>
        <bias>
          <float>1</float>
        </bias>
      </subtraction>
    </addition>
  </parameter>
</errorsFile>
```

4.2. Process logic

In this section, the process logic of using the libraries in models source code is shown.

Steps for using the Error Generation Libraries:

1. Include the OSFEG.h header file in your code

```
#include "OSFEG.h"
```

2. Create an instance of the ErrorSources class passing the name of the XML error definition file. The constructor throws an exception in case of error, so remember to handle it. Note that exceptions thrown from ErrorSources may have additional information about the cause of the problem in more specific exceptions nested in the outermost one[1].

```
ErrorSources reader = ErrorSources(errorDefinitionFile);
```

3. Access the perturbation values by the complete name of the parameter and a double specifying the simulation step.

```
reader.getError(paramName, step);
```

4.3. Examples of use

4.3.1. C++ Programming Language

Here is an example of C++ code that uses the error generation libraries.

```
#include "OSFEG.h"
#include <iostream>
#include <string>
#include <stdexcept>

using namespace std;

int main(int argc, char *argv[])
try {
   string config(argv[1]);
   cout << "Reading file " << config << endl;
   // Create an ErrorSources instance to read the file
   ErrorSources reader{config};
   double t = 1.25;
   string paramName = "Example Param Name";
   reader.getError(paramName, t);
   return 0;
} catch (const exception &e) {
   cerr << e.what() << endl;
   return 1;
}
```

[1] See for example the "print_exception" function in the example for nested exception functions in https://en.cppreference.com/w/cpp/error/throw_with_nested

4.3.2. C++ Compilation and Execution process

This section provides instructions for building the modules using CMake, the suggested build system. It assumes that OSFEG and Xerces are already built.

In order to provide the Xerces and OSFEG libraries to the building system, the user should use the CMake command *find_package*. Firstly, the developer shall add the Xercesc package with the commands shown below. It can be seen that function *find_package* allows the user to input the location of the library to be added. The package Threads refers to the threading library of the system and it is usually needed by Xerces.

```
find_package(Threads REQUIRED)
find_package(XercesC REQUIRED CONFIG HINTS "${XercesC_DIR}")
```

The OSFEG library is added using the same command.

```
find_package(OSFEG REQUIRED CONFIG HINTS "<OSFEG_DIR>")
```

Where <OSFEG_DIR> is as defined in section 3.2.1. After these commands, Xerces and OSFEG are available for the building process, which shall be performed with the proper CMake commands.

Integration libraries come in two distribution types, shared or static libraries.

If you have linked the shared libraries you can execute the binary files after specifying the location of those shared libraries like this:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<OSFEG_DIR>/lib
```

Linking with static libraries does not require specifying the location of the linked libraries since the executable already includes all the code.

The command for executing the example binary is:

```
./cppExample <arguments>
```

End of Document