1 INTRODUCTION

The EO Orbit and Attitude Adapter generates Orbit and Attitude files compliant with EOCFI format (see [RD1] and [RD2], sections 9.3 and 9.10) using data extracted from one or more binary files, for example files containing Telemetry packets including Orbit and Attitude information, such as NAVATT packets.
Orbit and Attitude files can then be ingested by EOCFI functions (see [RD1]) within a user application.

The Adapter can be used as a command line program or as a software library within a user application.

The Adapter receives as input:

- One or more **Input Files** containing orbit and attitude data;
- The **Main Configuration File** used to:
  - set and configure the parser to be used to extract relevant data from the Input Files;
  - set parameters required to write the output files.
- If applicable, the **Data Format Description File**: it describes the input data format and is used by the parser to interpret the binary data.

One of the following parsers can be selected:

- **DFDL4S**: the DFDL4S library (see [RD3]) is used to parse the data. The Data Format Description File has to be written according to the Mission Specification Schema format, see [RD5], that is based on the standard Data Format Description Language (DFDL, see [RD4]). DFDL is a modeling language based on W3C XML schemas (see [RD6]) for describing binary data in a standard way;
- **CUSTOM**: the user can write a customized parser.

When the DFDL4S parser is used, only elements required to write Orbit and Attitude files are extracted from the binary file (e.g. time, position, velocity, quaternions). Other elements are not extracted. To extract other elements, the DFDL4S library can be used directly, an example is provided within the DFDL4S Software distribution package, see [RD7].

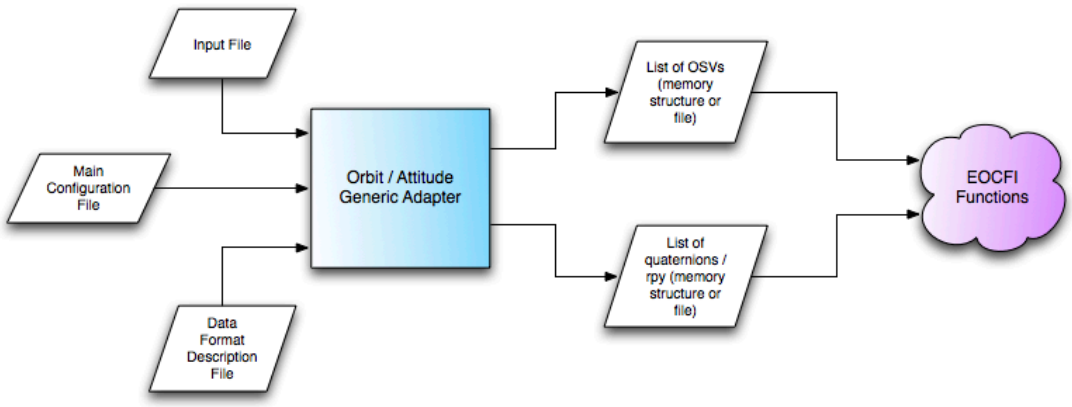

TABLE OF CONTENTS

Change history

Version	Date	Author	Description
1.0	16/12/2016	M. De Bartolomei (ESA/ESTEC, EOP-PE)	First Issue

References

Ref.	Description
[RD1]	Earth Observation Mission CFI Software Main Web page, http://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software
[RD2]	Data Handling Library Software User Manual, http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/CFI/EOCFI/BRANCH_4X/4.12/C-Docs/SUMs/DataHandlingSUM_v4_12.pdf
[RD3]	DFDL4S Developer's Manual, http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/APPLICATIONS/DFDL4S/DFDL4S_Developers_Manual_S2G-DME-TEC-SUM078-1E.pdf
[RD4]	Wikipedia: "Data Format Description Language", https://en.wikipedia.org/wiki/Data_Format_Description_Language
[RD5]	Mission Specification Schemas Developer's Manual, http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/APPLICATIONS/S2G/Mission_Specification_Schemas_S2G-DME-TEC-SUM092-1B.pdf
[RD6]	Wikipedia: "XML Schema (W3C)", https://en.wikipedia.org/wiki/XML_Schema_(W3C)
[RD7]	DFDL4S web page, http://eop-cfi.esa.int/index.php/applications/dfdl4s
[RD8]	Earth Observation Ground Segment File Format Standard (versions 2.0 and 3.0), http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/SYSTEM_SUPPORT_DOCS/PE-TN-ESA_GS-0001_2.0_Signed.pdf http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/SYSTEM_SUPPORT_DOCS/PE-TN-ESA-GS-0001%20EO%20GS%20File%20Format%20Standard%203.0.pdf
[RD9]	EOCFI Software Release Notes, http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/CFI/EOCFI/BRANCH_4X/4.12/EOCFI-4.12-ReleaseNotes.pdf
[RD10]	Eo Adapter Documentation, http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/CFI/EOCFI/ADAPTER/CURRENT/ API HTML documentation: http://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/CFI/EOCFI/ADAPTER/CURRENT/HTML

Applicability

This manual is applicable to version 1.0 of the Eo Adapter Software.

2 INSTALLATION

2.1 System Requirements

Platform Requirements
The Adapter runs in the same platforms supported by the EOCFI SW, see EOCFI SW Release Notes [RD9] for more details.

Software Requirements (Java version)
The following JAVA SDK is required for all platforms: **Java Standard Edition (SE) version 8.**
The following third party libraries are needed:

- **EOCFI Java Libraries v4.12 (see [RD1])**
- **DFDL4S Java Libraries v1.4 (see [RD7])**

2.2 Distribution Package

The Software is distributed via the Mission CFI Software section of the EOP-PE Website:
http://eop-cfi.esa.int/index.php/mission-cfi-software
as a zip package named as follows:

For the Java version:
`EO_ADAPTER-<version>-JAVA-<platform>.zip`
(for example `EO_ADAPTER-1.0-JAVA-LINUX64.zip`)

2.3 Installation

Installation requires to unzip the package in any directory. After unzipping the package, a directory EO_ADAPTER will be created with the following sub-directories:

Java version

Directory	Content
lib	The EO Orbit and Attitude Adapter library: EoOrbAttAdapter.jar.
bin	Adapter program ready to use from a command line interface or a script: • EoOrbAttAdapter_CLI.class: the class implementing the standalone program; • EoOrbAttAdapter_CLI.sh / .bat: script that executes the standalone program.
examples	Ready-to-use examples with standalone Java programs (including scripts to compile and run). • example_1: example for using the DFDL4S parser; • example_2: example for implementing and using a CUSTOM parser.
files	It contains the schema and a template for the configuration file. Examples of the configuration file can be found in the example directories.

3 USAGE

3.1 Overview

The Adapter can be used in different ways:

- **as a command line program:** the user can run the adapter via the command line interface or a script passing as inputs the main configuration file and the input data file(s). See Section 3.2 for more details;
- **as a library with a predefined parser (e.g. DFDL4S):** this option allows the user to call the Adpter methods directly from within his/her Software application. This option has the advantage of having Orbit and Attitude data directly usable within a user program without generating intermediate files. See Section 3.3 for more details;
- **as a library with a CUSTOM parser:** this option allows the user to call the Adapter methods directly from within his/her Software application and to use his/her own customized parser. See Section 3.4 for more details;

3.2 Usage as Command Line Program

3.2.1 Setup

The Command Line Interface (CLI) program runs via the EoOrbAttAdapter_CLI.sh (.bat in Windows platforms) located in the bin directory. This script needs to be edited and adapted according to the user system configuration. In particular the following environment variables need to be changed:

- (Optional) Set the path to the java executable
(only if the directory containing the java executable is not in the PATH and the JAVA_HOME environment variable is not already set):

```
    export JAVA_HOME='path to java executable'
```

- Set the path to the DFDL4S and EOCFI jar files

```
    export DFDL4S_CLASSPATH='directory where the dfdl4s.jar file is installed'
    export EOCFI_CLASSPATH='directory where the EOCFI jar files are installed'
```

- (Optional) Set the path to EoOrbAttAdapter_CLI.sh
(in order to call the script from any directory, add the directory in which is located to the PATH environment variable):

```
    export PATH=$PATH:'directory where EoOrbAttAdapter_CLI.sh is located'
```

3.2.2 Usage

The script can be executed as follows:

```
EoOrbAttAdapter_CLI.sh  [ARGUMENTS]
```

The script accepts the following arguments:

Argument	Description
-help (or no arguments)	Shows the help screen
-cfg “configuration_file“	Input configuration file (Mandatory)
-df “input_data_files“	Input data files. (Optional) Several files can be provided separated by a blank space. If not provided, the data file will be taken from the "configuration file"
-of “output_orbit_file“	Output orbit file. (Optional) If not provided, the output orbit file name will be taken, if available, from the "configuration file", otherwise the name will be generated according to the EO GS File Format Standard.
-af "output_attitude_file"	Output attitude file. (Optional) If not provided, the output attitude file name will be taken, if available, from the "configuration file", otherwise the name will be generated according to the EO GS File Format Standard.
-disableOrbitParser	Orbit parser disabled (Optional). Orbit file will not be created.
-disableAttitudeParser	Attitude parser disabled (Optional). Attitude file will not be created.

3.2.3 Example (Java)

Parse the data file used in example_1 and generate orbit and attitude files with name compliant with EO GS File Format Standard [RD8]:

```
> cd <path to Adapter installation root>
> cd examples/example_1
> ../../bin/EoOrbAttAdapter_CLI.sh -cfg ./eoadp_configuration_file.xml -df data/ISPData.dat
```

or, if the PATH has been set:

```
> EoOrbAttAdapter_CLI.sh -cfg ./eoadp_configuration_file.xml -df data/ISPData.dat
```

The data file to be parsed is dat/ISPData.dat. It contains a sequence of NAVATT packets originated by the Sentinel-3A satellite.

The configuration file is ./eoadp_configuration_file.xml. It defines the parser type (DFDL4S) and filename (with relative path) of the Mission Specification Schema (./data/schema/Sentinel3X-bandTM/Sentinel3X-bandTMISP.xsd):

```
<Data_Block type="xml">
   <Orbit_Attitude_Adapter_Configuration>
      <Input_Configuration>
…
         <Parser_Configuration type="DFDL4S" version="1.4">
            <Schema_Filename>./data/schema/Sentinel3X-bandTM/Sentinel3X-bandTMISP.xsd</Schema_Filename>
```

Since the schema filename is given with a relative path, the Adapter has to be called from the example/example_1 directory. In order to call the Adapter from any directory, the path has to be changed from relative to absolute.

The programs logs on standard output information related to the file(s) being processed, the progress and the orbit and attitude file generated:

```
[INFO    ] Earth Observation Orbit and Attitude Adapter. Version 1.0 (01-Dec-2016)
[INFO    ] Parsing File  1/ 1 : data/ISPData.dat
[INFO    ] Reading Input File
[INFO    ] Reading Input File
[INFO    ] Reading Input File
[INFO    ] Processed    526 /   6136 packets (  8% Completed)
[INFO    ] Processed   2557 /   6136 packets ( 41% Completed)
[INFO    ] Processed   4512 /   6136 packets ( 73% Completed)
[INFO    ] Processed   6136 /   6136 packets (100% Completed)
[INFO    ] Written Orbit File     : S3A_OPER_AUX_ORBRES_20160407T041544_20160407T055759_0001.EOF
[INFO    ] Written Attitude File  : S3A_OPER_AUX_ATTRES_20160407T041544_20160407T055759_0001.EOF
[INFO    ] Completed.
```

Orbit and Attitude files are generated with names according to EO GS File Format Standard [RD8].
More details about the configuration file and the schema file can be found in section 4.

If not provided via the –df option, the input file name will be taken from the main configuration file:

```
<Input_Configuration>
   <Default_Values>
      <Input_Data_Filename>data/DefaultFileName.dat</Input_Data_Filename>
   </Default_Values>
…
```

The following command will write orbit and attitude data respectively to orbit_file.xml and attitude_file.xml respectively:

```
> ../../bin/EoOrbAttAdapter_CLI.sh -cfg ./eoadp_configuration_file.xml -df data/ISPData.dat –of
orbit_file.xml –af attitude_file.xml
```

NOTE: EoOrbAttAdapter_CLI.sh is a wrapper around the call to the java interpreter. The java interpreter can also be invoked directly from the command line or another scripts provided the -classpath switch is set properly. Assuming that DFDL4S and EOCFI jar files are located respectively in the lib/cots/DFDL4S and lib/cots/EOCFI directories, the java interpreter can be invoked from the example/example_1 directory as follows (/*:../../bin EoOrbAttAdapter_CLI.class is the class implementing the command line program).

```
> java -classpath :../../lib/cots/DFDL4S/*:../../lib/cots/EOCFI/*:../../lib/*:../../bin
EoOrbAttAdapter_CLI -cfg eoadp_configuration_file.xml -df data/ISPData.dat
```

3.3 Usage as Library with a predefined Parser (DFDL4S)

3.3.1 Overview

The user can run the Adapter from within their own program thanks to the EoOrbAttAdapter class (EoOrbAttAdapter.jar in the lib directory). Once the input files are parsed, orbit and attitude data can be used to write files or can be passed directly to EOCFI functions.
These are the steps to be followed to build an application using the EoOrbAttAdapter class (see also the example program EoAdapterDfdl4sExample provided in example/example_1 directory):

1 - Import the package:

```
import EoOrbAttAdapter.*;
```

2 - Create the adapter and initialise the logger with the `initLogger` method (log messages will be sent to standard output)

```
EoOrbAttAdapter myAdapter = new EoOrbAttAdapter();
Logger logger = myAdapter.initLogger();
```

(Use the following call instead to log also to file)

```
logger = myAdapter.initLogger(logFile);
```

(`logger` is a standard Java Logger object, several methods are available such as `info, severe`)

```
// Log some information
logger.info( "Eo Adapter example START" );
logger.info("Parser Type    = " + adpConfig.getParserType());
logger.info("Parser Version = " + adpConfig.getParserVersion());
```

3 - Set the main configuration
- option 1: create a AdpMainConfiguration object and load it with the `loadMainConfiguration` method using eoadp_configuration_file.xml file. Set the configuration in the adapter with the `setMainConfiguration` method.

```
AdpMainConfiguration adpConfig = new AdpMainConfiguration();
adpConfig.loadMainConfiguration(configurationFile);
myAdapter.setMainConfiguration(adpConfig);
```

- option 2: initialise the adapter with the `setMainConfiguration` method using directly the configuration file.

```
myAdapter.setMainConfiguration(configurationFile);
```

3 – parse the input data file

```
    myAdapter.parse(inputDataFile);
```

(to parse only Orbit Data)

```
    myAdapter.disableAttitudeParser();
    myAdapter.parse(inputDataFile);
```

(if more than one file has needs to be parsed)

```
    myAdapter.parse(inputDataFile1);
    myAdapter.parse(inputDataFile2);
    myAdapter.parse(inputDataFile3);
```

(if the parse method is used without inputs, the input file will be taken from the configuration file)

```
    myAdapter.parse();
```

(a `EoAdapterProgressLogger` object can be created and used during parsing)

```
    EoAdapterProgressLogger progressLogger = new EoAdapterProgressLogger(myAdapter);
    Thread loggerThread = new Thread(progressLogger);
    loggerThread.start();
    myAdapter.parse(inputDataFile);
    progressLogger.stop();
```

4 – Use the data
- option 1: write data to file.
Filenames provided by the user:

```
    String outFileName;
    outFileName = myAdapter.writeOrbitDataToFile(orbitFileName);
```

```
    outFileName = myAdapter.writeAttitudeDataToFile(attitudeFileName);
```

Filenames generated according to [RD8] (note that `writeOrbitDataToFile` and `writeAttitudeDataToFile` methods return the filename that is not known in this case):

```
    outFileName = myAdapter.writeOrbitDataToFile();
    logger.info("Created Orbit File     : "+ outFileName);
    outFileName = myAdapter.writeAttitudeDataToFile();
    logger.info("Created Attitude File : "+ outFileName);
```

- option 2: get orbit and attitude data as `OrbitFile` and `AttFile` EOCFI objects.

```
    OrbitFile myOrbitFile = myAdapter.getOrbitData();
    AttFile   myAttFile   = myAdapter.getAttitudeData();
    logger.info("Number of AttRec elements parsed = " + myAttFile.attRec.size());
    logger.info("Number of OsvRec elements parsed = " + myOrbitFile.osvRec.size());
```

3.3.2 Example (Java)

The directory example/example_1 contains the EoAdapterDfdl4sExample.java program that can be built and excuted via the example_compile_and_run.sh script that needs to be adapted according to system configuration (see also section 3.2.1 and README file witihin the directory).
The program performs the following operations:
- Initialise the adapter using the main configuration file eoadp_configuration_file.xml;
- Initialise the logger and progress logger;
- parse the input data file data/ISPData.dat;
- write orbit and attitude data to files in several ways;
- retrieve and use orbit and attitude data as EOCFI SW data structures;
- (this step is just an example how to use Orbit and Attitude data directly from the application) use Orbit data to calculate a few Orbit State Vectors.

When executing the example_compile_and_run.sh script, the following output will be shown:

```
--------------------
compiling EoAdapterDfdl4sExample.java
--------------------
--------------------
executing
--------------------
[INFO    ] Eo Adapter example START
[INFO    ] Parser Type    = DFDL4S
[INFO    ] Parser Version = 1.4
[INFO    ] Parsing File : data/ISPData.dat
[INFO    ] Reading Input File
[INFO    ] Reading Input File
[INFO    ] Reading Input File
[INFO    ] Processed    457 /   6136 packets (  7% Completed)
[INFO    ] Processed   2526 /   6136 packets ( 41% Completed)
[INFO    ] Processed   4430 /   6136 packets ( 72% Completed)
[INFO    ] Processed   6136 /   6136 packets (100% Completed)
[INFO    ] Writing Orbit and Attitude Files with user provided file names
[INFO    ] Created Orbit File    : output_orbit_file.xml
[INFO    ] Created Attitude File : output_attitude_file.xml
[INFO    ] Writing Orbit and Attitude Files according to file format standard
[INFO    ] Created Orbit File    : S3A_OPER_AUX_ORBRES_20160407T041544_20160407T055759_0001.EOF
[INFO    ] Created Attitude File : S3A_OPER_AUX_ATTRES_20160407T041544_20160407T055759_0001.EOF
[INFO    ] Number of AttRec elements parsed = 6136
[INFO    ] Number of OsvRec elements parsed = 6136
[INFO    ] Calculate and print some OSVs
[INFO    ] TIME=5941.177591 X=    -1130750.045 Y=     -896561.605 Z=     7026142.694
[INFO    ] TIME=5941.184692 X=     -545617.995 Y=     3533382.657 Z=     6219995.306
[INFO    ] TIME=5941.191793 X=      583632.330 Y=     6508964.458 Z=     2973565.824
[INFO    ] TIME=5941.198893 X=     1633472.899 Y=     6847188.240 Z=    -1440737.196
[INFO    ] TIME=5941.205994 X=     1955024.372 Y=     4455814.262 Z=    -5290886.784
[INFO    ] TIME=5941.213095 X=     1226921.428 Y=      352256.260 Z=    -7074680.292
[INFO    ] TIME=5941.220195 X=     -345064.521 Y=    -3784896.547 Z=    -6101676.844
[INFO    ] TIME=5941.227296 X=    -2080935.573 Y=    -6305791.298 Z=    -2747952.969
[INFO    ] TIME=5941.234397 X=    -3116732.618 Y=    -6247729.507 Z=     1681412.881
[INFO    ] TIME=5941.241497 X=    -2820627.867 Y=    -3718888.480 Z=     5450173.576
[INFO    ] TIME=5941.248598 X=    -1137111.723 Y=      171847.876 Z=     7079836.235
[INFO    ] Eo Adapter example END
```

3.4 Usage as Library with a CUSTOM parser

3.4.1 Overview

In order to define a customized parser, the user has to implement the following Java classes:

- A Class extending the **`AdpDefaultParser`** class. This class has to implement the **`parse`** method that is in charge of reading the data file and filling the output objects. For an example of a custom parser, see the class **`MyParser`** in example/example_2 directory.
- A Class implementing the **`AdpParserConfiguration`** interface. This class has to implement the method **`loadFromFile`**. This method is used to read custom elements in the main configuration file. For an example of a custom parser configuration, see the class **`MyParserConfiguration`** in example/example_2 directory.

The custom parser can then be used as follows from within the user application (see **`EoAdapterCustomExample`** class in example/example_2 directory):

```
// Create the custom parser myTextParser and configuration handler parserConfig
MyParserConfiguration parserConfig = new MyParserConfiguration();
MyParser              myTextParser = new MyParser();
// set the configuration handler
myTextParser.setParserConfiguration(parserConfig);

// Create the adapter, set the custom parser and cofiguration
EoOrbAttAdapter myAdapter = new EoOrbAttAdapter();
myAdapter.setParser(myTextParser);
myAdapter.setMainConfiguration(configurationFile);

// Init the logger (log messages will be sent to standard output)
logger = myAdapter.initLogger();
// Use the following call instead to log also to file
//logger = myAdapter.initLogger(logFile);

// Log some information
logger.info( "Eo Adapter example (Custom Parser) START" );

// Parse the input data file.
logger.info("Parsing File : "+ inputDataFile);
myAdapter.parse(inputDataFile);
```

3.4.2 Example (Java)

The directory example/example_2 contains the EoAdapterCustomExample.java program that can be built and excuted via the example_compile_and_run.sh script that needs to be adapted according to system configuration (see also section 3.2.1 and README file witihin the directory).

In this example, the custom parser is used to read a file not readable using the DFDL4S library, text file data/my_data_file.txt containing one OSV per row.

The following classes are provided:

- **`MyParserConfiguration:`** it provides the **`loadFromFile`** method that reads user defined parameters from the main configuration file. When a custom parser is used, the parser configuration section in the configuration file can be filled with user defined variables values. In this example the Field_Separator and Quality variables are defined (see configuration file eoadp_configuration_file.xml). In this example the Field_Separator element "," defines the separator in the input text file and the Quality element is used to fill the Quality element in the output Orbit File.

```
<Parser_Configuration type="CUSTOM" version="1.0">
   <Schema_Filename></Schema_Filename>
   <Orbit status="enabled">
      <Field_Separator>, </Field_Separator>
      <Quality>1</Quality>
   </Orbit>
   <Attitude status="disabled">
   </Attitude>
</Parser_Configuration>
```

- **`MyParser:`** it implements the custom parser. It provides the **`parse`** method that is in charge of reading Orbit and Attitude data from the input file and fill the **`orbitData`** private variable.
- **`EoAdapterCustomExample:`** it implements a standalone program that uses the custom parser. It shows how to:
    - Initialise and set the custom parser and configuration handler;
    - Initialise the adapter using the main configuration file;
    - Initialise the logger;parse the input data file;
    - write orbit data to file;

When executing the example_compile_and_run.sh script, the following output will be shown:

```
--------------------
compiling MyParser.java EoAdapterCustomExample.java
--------------------
--------------------
executing
--------------------
[INFO    ] Eo Adapter example (Custom Parser) START
[INFO    ] Parsing File : data/my_data_file.txt
[INFO    ] Writing Orbit and Attitude Files according to file format standard
[INFO    ] Created Orbit File    : S3A_OPER_AUX_ORBRES_20030507T185536_20030507T235723_0001.EOF
[INFO    ] Number of OsvRec elements parsed = 4
[INFO    ] Eo Adapter example END
```

4 REFERENCE MANUAL

4.1 The Main Configuration File

4.1.1 Format Description

Operations of the adapter are controlled by a main configuration file. This configuration file has to be a file compliant with Earth Observation file format standard [RD8]. The following defines the sections enclosed in the Data Block relevant to the adapter configuration (i.e. header information is omitted).

```
<Data_Block type="xml">
  <Orbit_Attitude_Adapter_Configuration>
    <Input_Configuration>
      <Default_Values>
        <Input_Data_Filename>…</Input_Data_Filename>
      </Default_Values>
      <Parser_Configuration type="…" version="…">
      ...
      </Parser_Configuration>
      <Common>
      ...
      </Common>
    </Input_Configuration>
    <Output_Configuration>
      <Orbit status="…">
      ...
      </Orbit>
      <Attitude status="…">
      ...
      </Attitude>
      <Common>
      ...
      </Common>
    </Output_Configuration>
  </Orbit_Attitude_Adapter_Configuration>
</Data_Block>
```

The file is composed by a node named Orbit_Attitude_Adapter_Configuration that contains Input_Configuration and Output_Configuration nodes.

The Input_Configuration node contains:

- the Default_Values node: contains the Input_Data_Filename element. The value of this element is used as default input file name if this is not provided via the Adapter interface (i.e. when -df option in the command line program is omitted or the parse method is called without input arguments;
- the Parser_Configuration node: contains parameters needed to configure the parser. It has two attributes:
  - type: the parser type, either **DFDL4S** or **CUSTOM**;
  - version: the parser version, e.g. 1.4;
- the Common node: contains additional parameters needed to properly parse the data, e.g. time correlations.

```
<Parser_Configuration type="DFDL4S" version="1.4">
    <Schema_Filename>… </Schema_Filename>
    <Orbit status="…">
…
    </Orbit>

    <Attitude status="…">
…
    </Attitude>
</Parser_Configuration>
```

When the parser type is **CUSTOM**, the Parser_Configuration node can contain any user defined parameter, provided that the user implements the `AdpParserConfiguration` interface. In the example_2, the `MyParserConfiguration` class implements the `AdpParserConfiguration` interface and in particular the `loadFromFile` method that reads the parser specific parameters.

When the parser type is **DFDL4S**, the Parser_Configuration node ontains the following items:

- Schema_Filename: It gives the Mission Specification Schema file that defines the structure of the input data file;
- Orbit: contains parameters needed to extract Orbit data. It has the status attribute that can be set to either **enabled** or **disabled**. When the status is disabled, Orbit data is not extracted. Note that this value can be overridden by the Adapter main interface (i.e. –disableOrbitParser switch);
- Attitude: contains parameters needed to extract Attitude data. It has the status attribute that can be set to either **enabled** or **disabled**. When the status is disabled, Attitude data is not extracted. Note that this value can be overridden by the Adapter main interface (i.e. –disableAttitudeParser switch);

```
<Orbit status="enabled">
  <Mapping type="OSV">
    <Time_Reference>…</Time_Reference>
    <Time>… </Time>
    <Reference_Frame>…</Reference_Frame>
    <X correction_factor="…">… </X>
    <Y correction_factor="…">… </Y>
    <Z correction_factor="…">… </Z>
    <VX correction_factor="…">… </VX>
    <VY correction_factor="…">… </VY>
    <VZ correction_factor="…">… </VZ>
    <Orbit_Number default="…"/>
    <Quality default="…"/>
    <Default_Values>
      <Quality>…</Quality>
      <Orbit_Number type="…">
        <OSF_Filename>… </OSF_Filename>
      </Orbit_Number>
    </Default_Values>
  </Mapping>
</Orbit>
```

The Mapping section (included in the Orbit section) defines the correspondence between each item composing an Orbit State Vector (OSV) and an element in the Mission Specification Schema. Values are Xpath expressions indicating the location of the element. For example:

```
<X correction_factor="1000.">
/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/X
</X>
```

/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/X is an Xpath expression indicating where the element corresponding to the X component of the OSV can be found in the schema.

The following items are included in the Mapping section:

- Time: the OSV epoch. Its time reference is specified by the element Time_Reference (see below);
- X, Y, Z: position. The correction_factor attribute defines a correction to be applied to the data in order to have the position consistent to EOCFI conventions (i.e. expressed in meters);
- VX, VY, VZ: position. The correction_factor attribute defines a correction to be applied to the data in order to have the velocity consistent to EOCFI conventions (i.e. expressed in meters/sec);
- Orbit_Number: the absolute orbit number as written for each OSV in the Orbit File;
- Quality: the Quality field as written for each OSV in the Orbit File;
- Default_Values: this section is used when the attribute default is set to true. It contains:
  - A default value for Quality;
  - A default value of Orbit_number (if the type is set to OSF, the Orbit number will be calculated using the Orbit Scenario File whose name is within the OSF_Filename element).
- Time_Reference: either **`TAI, UTC, UT1`** or **`N/A`**. N/A implies that the time reference is defined implicitely in the DFDL4S schema;
- Reference_Frame: one of the following: **`BAR_MEAN_2000, HEL_MEAN_2000, GEO_MEAN_2000, MEAN_DATE, TRUE_DATE, EARTH_FIXED, BAR_MEAN_1950, QUASI_MEAN_DATE, PSEUDO_EARTH_FIXED`**

```
<Attitude status="…">
  <Mapping type="…">
    <Time_Reference>…</Time_Reference>
    <Time>… </Time>
    <Reference_Frame>…</Reference_Frame>
    <Q1 correction_factor="…">… </Q1>
    <Q2 correction_factor="…">… </Q2>
    <Q3 correction_factor="…">… </Q3>
    <Q4 correction_factor="…">… </Q4>
  </Mapping>
  <Axes_Mapping>
    <X>…</X>
    <Y>…</Y>
    <Z>…</Z>
  </Axes_Mapping>
</Attitude>
</Parser_Configuration>
```

The Attitude section has a structure similar to the Orbit section.
The type attribute in the Mapping section can be either **`Quaternions`** (mapped items are **`Q1, Q2, Q3, Q4`**) or **`Angles`** (mapped items are **`Roll, Pitch, Yaw`**).
Quaternions numbering follows the EOCFI conventions (i.e. Q4 is the scalar component).
Axes of this attitude frame are not necessarily defined according to EOCFI conventions. The Axes_Mapping section provides the correspondence between co-ordinate system axes in EOCFI conventions and input data. For example The following defines that X,Y,Z in EOCFI conventions correspond to Y positive, X positive and Z negative axes in data convention:

```
<Axes_mapping>
  <X>Y_positive</X>
  <Y>X_positive</Y>
  <Z>Z_negative</Z>
</Axes_mapping>
```

For example, <X>Y_positive</X> means that the X (positive) axis in EOCFI conventions corresponds to Y (positive) axis of the frame defined by the quaternions.

```
    <Common>
      <Model type="DEFAULT"/>
      <Time_Correlations type="FILE">
       <Time_Correlations_File>
          ./data/S3A_OPER_MPL_ORBSCT_20160216T192404_99999999T999999_0001.EOF
       </Time_Correlations_File>
       <Default_Values>
        <UTC_UT1 unit="s">0.</UTC_UT1>
        <UTC_TAI unit="s">-36.0</UTC_TAI>
        <UTC_GPS unit="s">-17.0</UTC_GPS>
       </Default_Values>
      </Time_Correlations>
    </Common>
```

The Common section contains:

- The Model section: it can be used to specify the model (i.e. model_id, see [RD1], Lib SUM) used by EOCFI computations. Only the **`DEFAULT`** model is supported;
- The Time_Correlations section is used to define the relation amongst UTC, TAI, UT1, GPS. The method used depends on the type attribute:
  - **`FILE`**: the file defined in Time_Correlations_File element will be used. The file will be used to initialise an EOCFI SW time_id;
  - **`FIXED_CORRELATIONS`**: the fields UTC_UT1, UTC_TAI, UTC_GPS will be used (e.g. UTC_UT1 = UTC time – UT1 time in seconds);
  - **`NONE:`** UTC_UT1 will be set to 0 and UTC_TAI / UTC_GPS to the values known by the Adapter.

```
<Output_Configuration>
 <Orbit type="EO_FILE">
  <Default_Filename></Default_Filename>
  <Time_Reference> </Time_Reference>
  <Reference_Frame> </Reference_Frame>
  <Header_Configuration>
   <Notes> </Notes>
   <File_Class> </File_Class>
   <File_Type> </File_Type>
   <File_Version></File_Version>
   <Source>
    <System> </System>
   </Source>
  </Header_Configuration>
 </Orbit>
 <Attitude type="EO_FILE">
  <Default_Filename></Default_Filename>
  <Time_Reference> </Time_Reference>
  <Reference_Frame> </Reference_Frame>
  <Target_Frame>Sat_Attitude</Target_Frame>
  <Header_Configuration>
   <Notes> </Notes>
   <File_Class> </File_Class>
   <File_Type> </File_Type>
   <File_Version></File_Version>
   <Source>
    <System> </System>
   </Source>
  </Header_Configuration>
 </Attitude>
 <Common>
  <Satellite_Id>SENTINEL_3A</Satellite_Id>
 </Common>
</Output_Configuration>
```

The Output_Configuration section contains the Orbit and Attitude sections. For each of them the type can be set to to **`EO_FILE`**, i.e. the output is written to a EOCFI compliant file.

The following elements are included:

- Default_Filename: this is the filename that will be used if not specified via the Adapter interface. If this element is empty, the file name will be generated as per File Format Standard [RD8].
- Time_Reference: either **`UTC, TAI, UT1`**. This is the time reference that will be written in the Orbit File variable header:

```
<Variable_Header>
  <Ref_Frame>GEO_MEAN_2000</Ref_Frame>
  <Time_Reference>TAI</Time_Reference>
</Variable_Header>
```

- Reference_Frame: one of the following: **BAR_MEAN_2000, HEL_MEAN_2000, GEO_MEAN_2000, MEAN_DATE, TRUE_DATE, EARTH_FIXED, BAR_MEAN_1950, QUASI_MEAN_DATE, PSEUDO_EARTH_FIXED.** If this value is different to the one in the Input Configuration, a frame transformation will be performed. This value will be written in the Orbit File variable header or in the Quaternion_Data node in the Attitude File

```
<Quaternion_Data>
  <Reference_Frame>GM2000</Reference_Frame>
```

- Target_Frame (only for Attitude): one of **Sat_Nominal_Attitude; Sat_Attitude; Instr_Attitude**. This value is used to fill the corresponding element Attitude_File_Type in the Attitude file.

The Header_Configuration section defines the Fixed Header fields as defined by the File Format Standard [RD8] that cannot be generated automatically by the Adapter.

The Common section provides the Satellite_Id element; it can be one fo the following:
ERS1,ERS2,ENVISAT,METOP1,METOP2,METOP3,
CRYOSAT,ADM,GOCE,SMOS,EARTHCARE,SWARM_A,SWARM_B,SWARM_C,
SENTINEL_1A,SENTINEL_1B,SENTINEL_1C,
SENTINEL_2A,SENTINEL_2B,SENTINEL_2C,
SENTINEL_3A,SENTINEL_3B,SENTINEL_3C,
JASON_CSA,JASON_CSB,
METOP_SG_A1,METOP_SG_A2,METOP_SG_A3,METOP_SG_B1,METOP_SG_B2,METOP_SG_B3,
SEOSAT,SENTINEL_5P,BIOMASS,SENTINEL_5,SAOCOM_CS,GENERIC,GENERIC_GEO,MTG

An optional element EOFFS_Version can be added in the Orbit and/or Attitude section to write files according to a give File Format Standard version (either 1.4, 2.0 or 3.0, see example below). If not specified, the used version will be the one applicable for the mission specified in the Mission_Id element.

```
<Orbit type="EO_FILE">
  <EOFFS_Version>3.0</EOFFS_Version>
  <Default_Filename></Default_Filename>
```

4.1.2 Example

The Sentinel3X-bandTMISP.xsd file located in examples/example_1/data/schema is part of the Mission Specification Schema for a Sentinel-3A. It defines the structure of an ISP (Instrument Source Packet) for this mission. A single ISP is defined as the sequence of two elements: Packet_Primary_Header and Packet_Data_Field. The corresponding complex types are defined in the included Sentinel3X-bandTMISPTypes.xsd file.

```
<xs:schema …>
…
  <xs:include schemaLocation="Sentinel3X-bandTMISPTypes.xsd"/>
…
  <xs:element name="ISP">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Packet_Primary_Header" type="TypePacketPrimaryHeader"/>
        <xs:element name="Packet_Data_Field" type="TypePacketData"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
…
</xs:schema>
```

Sentinel3X-bandTMISP.xsd (extract)

As defined in Sentinel3X-bandTMISPTypes.xsd, the structure of the TypePacketData depends on the APID. If the APID is set to 361, than the packet is identified as a NAVATT packet.

```
<xs:complexType name="TypePacketData">
  <xs:sequence>
    <xs:choice>
…
      <xs:sequence> <!-- Choice for NAVATT -->
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator test="{/Packet_Primary_Header/Packet_Identification/APID in [361]}"/>
          </xs:appinfo>
        </xs:annotation>
        <xs:element name="NAVATT_Packet_Secondary_Header" type="TypePacketData_NAVATT"/>
        <xs:element name="NAVATT_User_Data_Field" type="TypeUserData_NAVATT"/>
      </xs:sequence>
…
```

Sentinel3X-bandTMISPTypes.xsd (extract)

Information related to Navigation and Attitude is located in the element NAVATT_User_Data_Field of type TypeUserData_NAVATT. This type is defined as the sequence of ISP_Data of TypeISPData_NAVATT type and Packet_Error_Control element.

```
  <!-- User Data NAVATT -->
  <xs:complexType name="TypeUserData_NAVATT">
    <xs:sequence>
      <xs:element name="ISP_Data" type="TypeISPData_NAVATT"/>
      <xs:element name="Packet_Error_Control" type="TypeCRC"/>
    </xs:sequence>
  </xs:complexType>
```

Sentinel3X-bandTMISPTypes.xsd (extract)

The TypeISPData_NAVATT type is defined within the Sentinel3X-bandTMISPData.xsd file.
The location of each item is defined by an Xpath expression, for example:

NAVATT Time:
/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/NAVATT_Time

OSV, position, X component:
/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/X

Quaternions, scalar part:
/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q0

```
<!-- User Data NAVATT -->
  <xs:complexType name="TypeISPData_NAVATT">
    <xs:sequence>
      …
      <xs:element name="Navigation_and_Attitude" type="TypeNavigationAttitude_NAVATT"/>
      …
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="TypeNavigationAttitude_NAVATT">
    <xs:sequence>
      <xs:element name="NAVATT_Time" type="TypeTimeCode_CUC_32_24" dmx:representation="Time"
        dmx:epoch="GPS"/>
      <xs:element name="Satellite_Position" type="TypeSatellitePosition_NAVATT"/>
      <xs:element name="Satellite_Velocity" type="TypeSatelliteVelocity_NAVATT"/>
      <xs:element name="Attitude_Data" type="TypeSatelliteAttitude_NAVATT"/>
      <xs:element name="Attitude_Quaternion_Difference" type="xs:hexBinary"
        dfdl:lengthKind="explicit" dfdl:lengthUnits="bytes" dfdl:length="32"
        dmx:representation="Hexadecimal"/>
    </xs:sequence>
  </xs:complexType>
```

```
<xs:complexType name="TypeSatellitePosition_NAVATT">
  <xs:sequence>
    <xs:element name="X" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Y" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Z" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="TypeSatelliteVelocity_NAVATT">
  <xs:sequence>
    <xs:element name="X" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Y" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Z" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="TypeSatelliteAttitude_NAVATT">
  <xs:sequence>
    <xs:element name="Q0" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Q1" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Q2" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="Q3" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="SC_RATE_1" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="SC_RATE_2" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
    <xs:element name="SC_RATE_3" type="xs:double" dfdl:lengthKind="explicit"
      dfdl:lengthUnits="bytes" dfdl:length="8" dmx:representation="Float64"/>
  </xs:sequence>
</xs:complexType>
```

Sentinel3X-bandTMISPData.xsd (extract)

The following eoadp_configuration_file.xml configuration file can be used to extract Orbit and Attitude data and generate Orbit and Attitude files:

```
<Data_Block type="xml">
 <Orbit_Attitude_Adapter_Configuration>
  <Input_Configuration>
   <Default_Values>
    <Input_Data_Filename> </Input_Data_Filename>
   </Default_Values>
   <Parser_Configuration type="DFDL4S" version="1.4">
    <Schema_Filename>./data/schema/Sentinel3X-bandTM/Sentinel3X-bandTMISP.xsd</Schema_Filename>
    <Orbit status="enabled">
     <Mapping type="OSV">
      <Time_Reference>N/A</Time_Reference>
      <Time>/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/NAVATT_Time</Time>
      <Reference_Frame>GEO_MEAN_2000</Reference_Frame>
      <X correction_factor="1000.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/X</X>
      <Y correction_factor="1000.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/Y</Y>
      <Z correction_factor="1000.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Position/Z</Z>
      <VX correction_factor="1000.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Velocity/X</VX>
      <VY correction_factor="1000">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Velocity/Y</VY>
      <VZ correction_factor="1000.">
```

```
            /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Satellite_Velocity/Z</VZ>
        <Orbit_Number default="true"/>
        <Quality default="true"/>
        <Default_Values>
          <Quality>0000000000000</Quality>
          <Orbit_Number type="OSF">
            <OSF_Filename>./data/S3A_OPER_MPL_ORBSCT_20160216T192404_99999999T999999_0001.EOF</OSF_Filename>
          </Orbit_Number>
        </Default_Values>
      </Mapping>
    </Orbit>
    <Attitude status="enabled">
      <Mapping type="Quaternions">
        <Time_Reference>N/A</Time_Reference>
        <Time>/Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/NAVATT_Time</Time>
        <Reference_Frame>GEO_MEAN_2000</Reference_Frame>
        <Q1 correction_factor="1.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q1
        </Q1>
        <Q2 correction_factor="1.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q2</Q2>
        <Q3 correction_factor="1.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q3</Q3>
        <Q4 correction_factor="1.">
          /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q0</Q4>
      </Mapping>
      <Axes_Mapping>
        <X>Y_positive</X>
        <Y>X_positive</Y>
        <Z>Z_negative</Z>
      </Axes_Mapping>
    </Attitude>
  </Parser_Configuration>
  <Common>
    <Model type="DEFAULT"/>
    <Time_Correlations type="FILE">
      <Time_Correlations_File>
          ./data/S3A_OPER_MPL_ORBSCT_20160216T192404_99999999T999999_0001.EOF
      </Time_Correlations_File>
      <Default_Values>
        <UTC_UT1 unit="s">0.</UTC_UT1>
        <UTC_TAI unit="s">-36.0</UTC_TAI>
        <UTC_GPS unit="s">-17.0</UTC_GPS>
      </Default_Values>
    </Time_Correlations>
  </Common>
</Input_Configuration>
<Output_Configuration>
  <Orbit type="EO_FILE">

    <Default_Filename></Default_Filename>
    <Time_Reference>TAI</Time_Reference>

    <Reference_Frame>EARTH_FIXED</Reference_Frame>
    <Header_Configuration>
      <Notes>Orbit File generated by EoOrbAttAdapter</Notes>
      <File_Class>OPER</File_Class>
      <File_Type>AUX_ORBRES</File_Type>
      <File_Version>0001</File_Version>
      <Source>
        <System>EoOrbAttAdapter</System>
      </Source>
    </Header_Configuration>
  </Orbit>
  <Attitude type="EO_FILE">
    <Default_Filename></Default_Filename>
    <Time_Reference>UTC</Time_Reference>
    <Reference_Frame>GEO_MEAN_2000</Reference_Frame>
    <Target_Frame>Sat_Attitude</Target_Frame>
    <Header_Configuration>
      <Notes>Attitude File generated by EoOrbAttAdapter</Notes>
      <File_Class>OPER</File_Class>
      <File_Type>AUX_ATTRES</File_Type>
      <File_Version>0001</File_Version>
```

```
        <Source>
          <System>EoOrbAttAdapter</System>
        </Source>
      </Header_Configuration>
    </Attitude>
    <Common>
      <Satellite_Id>SENTINEL_3A</Satellite_Id>
    </Common>
  </Output_Configuration>
 </Orbit_Attitude_Adapter_Configuration>
</Data_Block>
```

eoadp_configuration_file.xml (extract)

The mapping section gives the correspondence between OSV / Quaternions items and elements defined in the schema. For example:

```
<Q4 correction_factor="1.">
   /Packet_Data_Field/NAVATT_User_Data_Field/ISP_Data/Navigation_and_Attitude/Attitude_Data/Q0
</Q4>
```

Item Q4 in EOCFI conventions (i.e. scalar part) has to be taken form item located in element Q0 located in element Attitude_Data (Q0 is the scalar part in NAVATT conventions).

NAVATT_Time is defined as follows:

```
<xs:element name="NAVATT_Time" type="TypeTimeCode_CUC_32_24" dmx:representation="Time" dmx:epoch="GPS"/>
```

When the representation is set to Time, DFDL4S always return an UTC time, therefore the Time_Reference field is ignored and therefore set to N/A.
The Reference_Frame defines the reference frame of OSVs in the input files (GM2000 in case of NAVATT);

The correction_factor value is a correction factor needed to get the OSV expressed in m and m/s as per EOCFI conventions. Since in the NAVATT packets position and velocity are expressed in km and km/s, a value of 1000. Is needed.

The Orbit Scenario File is used for setting the time correlations and compute Orbit number fro each OSV.

The following Orbit file is created.
The Variable Header has been written according to the output configuration. OSVs have been converted from GM2000 co-ordinate system to Earth Fixed.

```
<Earth_Explorer_File xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://eop-cfi.esa.int/CFI
http://eop-cfi.esa.int/CFI/EE_CFI_SCHEMAS/EO_OPER_MPL_ORBPRE_0203.XSD" schemaVersion="2.3" xmlns="http://eop-
cfi.esa.int/CFI">
 <Earth_Explorer_Header>
  <Fixed_Header>
   <File_Name>S3A_OPER_AUX_ORBRES_20160407T041544_20160407T055759_0001</File_Name>
   <File_Description>Orbit file</File_Description>
   <Notes/>
   <Mission>SENTINEL_3A</Mission>
   <File_Class>OPER</File_Class>
   <File_Type>AUX_ORBRES</File_Type>
   <Validity_Period>
    <Validity_Start>UTC=2016-04-07T04:15:44</Validity_Start>
    <Validity_Stop>UTC=2016-04-07T05:57:59</Validity_Stop>
   </Validity_Period>
   <File_Version>0001</File_Version>
   <Source>
    <System>EoOrbAttAdapter</System>
    <Creator>ESA EoOrbAttAdapater tool.</Creator>
    <Creator_Version>1.0</Creator_Version>
    <Creation_Date>UTC=2016-12-16T18:51:17</Creation_Date>
   </Source>
  </Fixed_Header>
  <Variable_Header>
   <Ref_Frame>EARTH_FIXED</Ref_Frame>
   <Time_Reference>TAI</Time_Reference>
```

```
    </Variable_Header>
  </Earth_Explorer_Header>
  <Data_Block type="xml">
    <List_of_OSVs count="6136">
      <OSV>
        <TAI>TAI=2016-04-07T04:16:19.875000</TAI>
        <UTC>UTC=2016-04-07T04:15:43.875000</UTC>
        <UT1>UT1=2016-04-07T04:15:43.875000</UT1>
        <Absolute_Orbit>+00719</Absolute_Orbit>
        <X unit="m">-1130750.045</X>
        <Y unit="m">-0896561.605</Y>
        <Z unit="m">+7026142.694</Z>
        <VX unit="m/s">+0295.077144</VX>
        <VY unit="m/s">+7459.407650</VY>
        <VZ unit="m/s">+0997.248885</VZ>
        <Quality>0000000000000</Quality>
      </OSV>
...
    </List_of_OSVs>
  </Data_Block>
</Earth_Explorer_File>
```

Similarly, an Attitude file is written:

```
<Earth_Explorer_File xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://eop-cfi.esa.int/CFI
http://eop-cfi.esa.int/CFI/EE_CFI_SCHEMAS/EO_OPER_INT_ATTREF_0203.XSD" schemaVersion="2.3" xmlns="http://eop-
cfi.esa.int/CFI">
  <Earth_Explorer_Header>
    <Fixed_Header>
      <File_Name>S3A_OPER_AUX_ATTRES_20160407T041544_20160407T055759_0001</File_Name>
      <File_Description>Attitude file</File_Description>
      <Notes/>
      <Mission>SENTINEL_3A</Mission>
      <File_Class>OPER</File_Class>
      <File_Type>AUX_ATTRES</File_Type>
      <Validity_Period>
        <Validity_Start>UTC=2016-04-07T04:15:44</Validity_Start>
        <Validity_Stop>UTC=2016-04-07T05:57:59</Validity_Stop>
      </Validity_Period>
      <File_Version>0001</File_Version>
      <Source>
        <System>EoOrbAttAdapter</System>
        <Creator>ESA EoOrbAttAdapater tool.</Creator>
        <Creator_Version>1.0</Creator_Version>
        <Creation_Date>UTC=2016-12-16T18:51:17</Creation_Date>
      </Source>
    </Fixed_Header>
    <Variable_Header/>
  </Earth_Explorer_Header>
  <Data_Block type="xml">
    <Attitude_File_Type>Sat_Attitude</Attitude_File_Type>
    <Attitude_Data_Type>Quaternions</Attitude_Data_Type>
    <Max_Gap unit="s">1.100000</Max_Gap>
    <Quaternion_Data>
      <Reference_Frame>GM2000</Reference_Frame>
      <List_of_Quaternions count="6136">
        <Quaternions>
          <Time ref="UTC">UTC=2016-04-07T04:15:43.875000</Time>
          <Q1>-0.988338255</Q1>
          <Q2>-0.114712107</Q2>
          <Q3>-0.082772440</Q3>
          <Q4>-0.056367975</Q4>
        </Quaternions>
...
```

4.2 API Reference

4.2.1 Overview

The following classes are provided within the **`EoOrbAttAdapter`** package (EoOrbAttAdapter.jar file in the lib directory):

- **`EoOrbAttAdapter`**: the main Adapter Interface. It provides methods to set the configuration, parse files, retrieve orbit/attitude data, write orbit/attitude files.
- **`AdpMainConfiguration`**: class containing the Adapter configuration parameters. It provides methods to load the configuration from a file.
- **`AdpParser`**: parser interface. This interface is implemented by predefined parsers (e.g. DFDL4S) and has to be implemented by user defined (CUSTOM) parsers.
- **`AdpDefaultParser`**: abstract parser class. This class implements all methods provided by **`AdpParser`** interface but the **`parse`** abstract method that is parser specific. A class that implements a parser (e.g. DFDL4S or CUSTOM parser) has to extend this class, i.e. implement the abstract **`parse`** method.
- **`AdpDfdl4sParser`**: DFDL4S parser class. It extends the **`AdpDefaultParser`** abstract class.
  `AdpParserConfiguration`: parser configuration interface. A class implementing this interface contanins the parser configuration parameters and implement the **`loadFromFile`** method that reads parser specific parameters from the Parser_Configuration section in the configuration file:

```
<Input_Configuration>
  <Parser_Configuration>
  …
  </Parser_Configuration>
```

- **`AdpDfdl4sParserConfiguration`**: DFDL4S parser configuration class. It contains configuration parameters for the DFDL4S parser. It implements the **`AdpParserConfiguration`** interface.
- **`EoAdapterProgressLogger`**: this class can be used to log progress of the parsing process. It provides two methods: **`start`** and **`stop`** to respectively start and stop monitoring and stop to stop monitornig the parsing process.
- **`EoAdapterError`**: class for error handling. An object of this class is thrown (and has to be thrown by user applications) in case of errors. The exception must be catched with a catch statement as shown in the examples.
- **`EoOrbAttAdapter_CLI`**: calss implementing the Command Line program.
- Other helper classes: **`AdpAttitudeOutputConfiguration, AdpCommonInputConfiguration, AdpCommonOutputConfiguration, AdpFileHeaderConfiguration, AdpInputConfiguration, AdpOrbitOutputConfiguration, AdpOutputConfiguration, AdpUtils, EoAdapterLogFormatter.`**

Example of usage are given in:

- examples/example_1/EoAdapterDfdl4sExample.java for usage of **`EoOrbAttAdapter, AdpMainConfiguration, EoAdapterProgressLogger, EoAdapterError.`**
- examples/example_2/EoAdapterCustomExample.java for the implementation of **`AdpParser`** the and **`AdpParserConfiguration`** interfaces.

All classes are described in the API HTML documentation [RD10]. The following sections give a detailed description for the most important classes.

4.2.2 `EoOrbAttAdapter` Class

The **`EoOrbAttAdapter`** class provides the following methods:

Method	Description
`void clearAttitudeData()`	Clear internal attitude data. *When the `parse` method is called, an internal memory structure is filled. The following calls to the `parse` method will add data to the structure. Calling the `clearAttitudeData` method will clear such memory structure.*
`void clearOrbitData()`	Clear internal orbit data (see `clearAttitudeData`).
`void disableAttitudeParser()`	Disable the attitude parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void disableOrbitParser()`	Disable the orbit parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void enableAttitudeParser()`	Enable the attitude parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void enableOrbitParser()`	Enable the orbit parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`AttFile getAttitudeData()`	Return parsed Attitude data as an EOCFI AttFile object.
`AdpMainConfiguration getMainConfiguration()`	Return the currently set configuration as an AdpMainConfiguration object.
`long getNofItems()`	Get Number of items (packets) contained in the data file. *This method returns a valid value only during the execution of the parse method.*
`long getNofProcessedItems()`	Get Number of processed items (packets). *This method returns a valid value only during the execution of the parse method.*
`OrbitFile getOrbitData()`	Return parsed Attitude data as an EOCFI OrbitFile object.
`java.lang.String getVersion()`	Returns Adapter tool version
`java.util.logging.Logger initLogger()`	Init and returns the Logger. *Log messages are sent to standard output.*
`java.util.logging.Logger initLogger (java.lang.String logFile)`	Init and returns the Logger. *Log messages are sent to standard output and* logFile.
`void parse()`	Parse the input data file defined in the main configuration file. *The file to be parsed will be taken from the main configuration file:* <Input_Configuration> <Default_Values> <Input_Data_Filename>… </Input_Data_Filename> </Default_Values>
`void parse(java.lang.String dataFile)`	Parse the input dataFile.
`void setMainConfiguration (AdpMainConfiguration inputConfig)`	Set the Adapter main configuration using an AdpMainConfiguration object.
`void setMainConfiguration (java.lang.String configurationFile)`	Set the Adapter main configuration using the input configurationFile.
`void setParser (AdpDefaultParser inputParser)`	Set a user defined (custom) parser *The inputParser will be used by the next calls to the parse method if the parser type is CUSTOM.*
`java.lang.String writeAttitudeDataToFile()`	Write attitude data to attitude File and return the filename *Filename will be taken from Main Configuration File:* <Output_Configuration> <Attitude type="EO_FILE"> <Default_Filename>…</Default_Filename> *If empty the filename will be generated according to File Format Standard [RD8].*
`java.lang.String writeAttitudeDataToFile (java.lang.String OutputAttFile)`	Write attitude data to OutputAttFile and return the filename
`java.lang.String writeOrbitDataToFile()`	Write orbit data to Orbit File and return the filename *Filename will be taken from Main Configuration File:* <Output_Configuration> <Orbit type="EO_FILE"> <Default_Filename>…</Default_Filename> *If empty the filename will be generated according to File Format Standard [RD8].*
`Java.lang.String writeOrbitDataToFile (java.lang.String OutputOrbitFile)`	Write attitude data to OutputAttFile and return the filename

4.2.3 AdpMainConfiguration Class

The **`AdpMainConfiguration`** class provides the following methods:

Method	Description
`void loadMainConfiguration (String configurationFile)`	Load the configuration parameters from a configuration file.
`boolean getAttitudeEnabled()`	Return true (false) if the attitude parser is enabled (disabled).
`void setAttitudeEnabled (boolean attEnabled)`	Set the status of the attitude parser (true: enabled, false: disabled)
`boolean getOrbitEnabled()`	Return true (false) if the orbit parser is enabled (disabled).
`void setOrbitEnabled (boolean orbEnabled)`	Set the status of the orbit parser (true: enabled, false: disabled)
`AdpInputConfiguration getInputConfiguration()`	Return the input configuration section i.e. parameters included in the Input_Configuration section of the main configuration file.
`void setInputConfiguration (AdpInputConfiguration inputConf)`	Set the input configuration section.
`AdpOutputConfiguration getOutputConfiguration()`	Get Output configuration i.e. parameters included in the Output_Configuration section of the main configuration file.
`void setOutputConfiguration (AdpOutputConfiguration outputConf)`	Set Output configuration.
`String getInputDataFile()`	Get Input data file name (i.e. the name of the file that will be parsed).
`void setInputDataFile (java.lang.String dataFile)`	Set Input data file name (i.e. the name of the file that will be parsed).
`String getParserType()`	Get parser type.
`String getParserVersion()`	Get parser version.
`TimeCorrelation getTimeCorrelations()`	Get Time correlations
`protected void initTimeCorrelations()`	Set the time correlations

4.2.4 AdpParserConfiguration Interface

A class implementing this interface contains user defined parser configuration parameters. The class has to declare such parameters as public variable members and implement the **`loadFromFile`** that reads such parameters from the main configuration file.

For example, to read the Field_Separator and Quality variables from the configuration file:

```
<Parser_Configuration type="CUSTOM" version="1.0">
     <Schema_Filename></Schema_Filename>
     <Orbit status="enabled">
        <Field_Separator>, </Field_Separator>
        <Quality>1</Quality>
     </Orbit>
     <Attitude status="disabled">
     </Attitude>
   </Parser_Configuration>
```

The user can implement the following class (see examples/example_2):

```
public class MyParserConfiguration implements AdpParserConfiguration
{
  public double quality;
  public String fieldSeparator;

  public void loadFromFile(String fileName) throws EoAdapterError
  {
      // ... read variables from configuration file and assign to the public variables ...

      // read field_separator tag
      fieldSeparator = ...;

      // read quality tag
      quality = ...;
```

4.2.5 `AdpDefaultParser` Abstract class

The **`AdpDefaultParser`** class provides the following methods:

Method	Description
`void disableAttitudeParser()`	Disable the attitude parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void disableOrbitParser()`	Disable the orbit parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void enableAttitudeParser()`	Enable the attitude parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`void enableOrbitParser()`	Enable the orbit parser. *This method overrides the status attribute enabled/disabled in the main configuration.*
`AttFile getAttitudeData()`	Return parsed Attitude data as an EOCFI AttFile object.
`long getNofItems()`	Get Number of items (packets) contained in the data file. *This method returns a valid value only during the execution of the parse method.*
`long getNofProcessedItems()`	Get Number of processed items (packets). *This method returns a valid value only during the execution of the parse method.*
`OrbitFile getOrbitData()`	Return parsed Attitude data as an EOCFI OrbitFile object.
`abstract void parse(String dataFile)`	Parse the input dataFile.
`AdpParserConfiguration getParserConfiguration()`	Get the parser configuration.
`protected void setMainConfiguration (AdpMainConfiguration inMainConfiguration)`	Set the main configuration (as it could be needed by the parser).
`Void setParserConfiguration (AdpParserConfiguration inParserConfiguration)`	Set the configuration for the parser.

A user defined (CUSTOM) parser has to extend this class, in particular implement the parse method, see example/example_2.

4.2.6 `EoAdapterError` Class

When an error is raised by the Adapter methods, an **`EoAdapterError`** exception is raised. This can be handled with a try / catch block:

```
try
  {
  EoOrbAttAdapter myAdapter = new EoOrbAttAdapter();
  ...
  myAdapter.parse(inputDataFile);
  ...
  }
  catch (EoAdapterError err)
  {
  ...
  }
```

The class provides the **`getError`** method, that return a list of error messages.

To throw an error, the **`addError`** and **`addCfiErrors`** methods can be used to add error messages to the list.

```
    EoAdapterError err = new EoAdapterError();
    err.addError("Log file could not be created");
    throw(err);
```