SOFTWARE USER MANUAL

Earth Observation Software.
Orbit and Attitude Adapter Tool
(C++)

Code:
Issue: 1.4
Approval Date: 31/08/2023
Confidentiality Level:

Prepared by: Juan José Borrego
Reviewed by: Inês Estrela
Approved by: Inês Estrela

Approval Signature:

This page intentionally left blank

Document Status Log

Issue	Section	Change Description	Date
1.0		First Release	19/12/2016
1.1		- Configuration file format updated (rotation angles specification) - Allow processing the input data from memory - Add inconsistencies check when processing NAVATT packages - Quaternion generation according to EOCFI convention - Time initialization based on IERS Bulletins enabled - Support for all the missions supported by EOCFI	21/06/2019
1.2		- EOCFI libraries updated to latest version 4.20 - Support for all the missions supported by EOCFI	01/12/2020
1.3		- EOCFI libraries updated to latest version 4.21	08/11/2021
1.4		- EOCFI libraries updated to latest version 4.25	31/08/2023

Table of Contents

List of Tables

1. INTRODUCTION

1.1. Purpose

The Earth Observation Mission Software (EOCFI SW) is a set of multiplatform software libraries which are made available free of charge to any user involved in supporting the Earth Observation missions preparation and exploitation. The EOCFI SW is typically used in Ground Segment operational facilities and in tools developed by ESA and their partners for Earth Observation Missions.

The EOCFI SW libraries provide functionalities for a wide range of high-level computations: orbit (e.g. interpolation, propagation using different models); attitude (e.g. interpolation, attitude law); target pointing (e.g. direct/inverse geo-location with DEM); geometric properties of calculated targets; instrument swath computation and zone intersection; zone/station visibility events; observation opportunities for instruments (time segments and coverage). Low level functions are also provided, for example to support for several file formats read/write; co-ordinates / time transformations.

The main inputs to the above functions are related to the satellite state. This information can be originated from different sources i.e.: models (e.g. propagators) implemented within the EOCFI SW itself; simulations or dedicated services (e.g. POD); S/C telemetry.

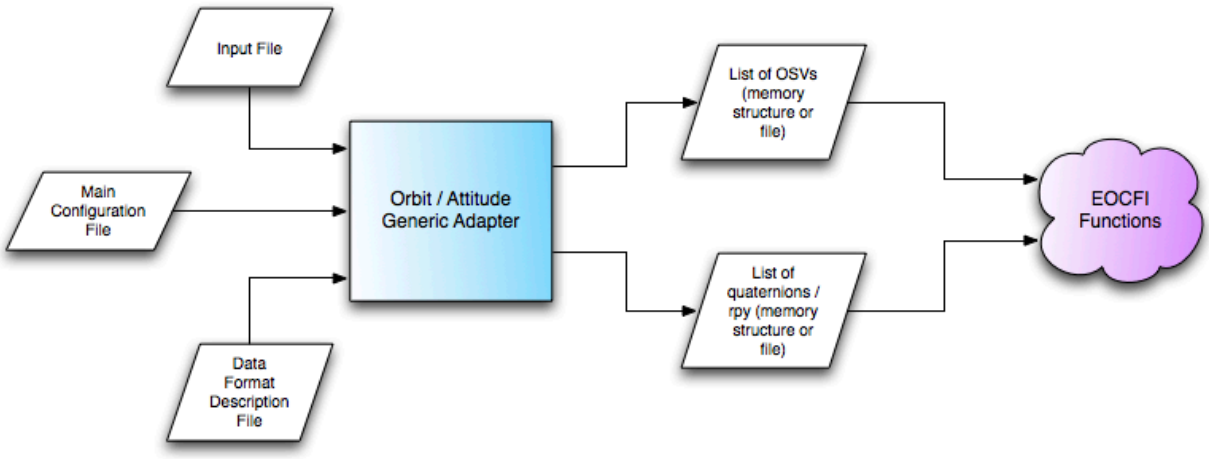

Figure 1 - Generic adapter overall architecture

Data required to initialise the satellite state is typically:

- a list of position and velocity at given times (OSVs);
- a list of attitude parameters at a given times (e.g. quaternions, roll-pitch-yaw angles).

Such inputs can be passed to EOCFI functions via dedicated data structures or files. The EOCFI SW supports several types of files (see [RD 2]) and, in particular, orbit and attitude files compliant with the EO GS File Format Standard (see [RD 3]). However, very often such inputs may be made available using formats that are not directly supported by the EOCFI SW. For example, downlinked OSVs and quaternions are received within Instrument Source Packets (ISPs) whose format is mission dependent. Normally, for each specific format, a dedicated decoder needs to be implemented. The Orbit Attitude adapter tool preforms the translation between formats.

The Orbit Attitude adapter tool / API receives as input:

- Data (i.e. files or data in memory) containing orbit and attitude;
- Optionally, data format description in e.g. DFDL language (DFDL is "a modeling language for describing general text and binary data in a standard way", see [RD 4]);
- A main configuration file that includes the required configuration parameters (for example the parser that, for the DFDL language is the DFDL4S parser, see[RD 5]).

And generating as output OSV and quaternions/rpy as lists in memory structures or files that can be ingested by EOCFI functions.

1.2. The Orbit Attitude Adapter Tool

The Orbit Attitude Adapter tool consists in a C++ library (a *.so* file) that can be used in a C++ program.

The executable program allows parsing input data files using a parser defined by the user ("CUSTOM" parser).

The following sections of this document describe how to install and use the tool:

- Section 3 describes the installation instructions.
- Section 4 describes how to use the program and/or the JAVA library.
- Section 5 describes the main configuration file to be used by the tool.
- Section 6 describes the API for the C++ library.

1.3. Acronyms and Abbreviations

TBW

1.4. Definitions

TBW

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]			
[AD 2]			

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	Title	Issue
[RD 1]	EOCFI documentation main page, http://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software/branch-4-x/eocfi-v4x-documentation	-
[RD 2]	"Earth Observation Mission CFI Software – Data Handling User Manual", v4.25, https://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/CFI/EOCFI/BRANCH_4X/releases/4.21/C-Docs/SUM/DataHandlingSUM_v4_25.pdf	4.25
[RD 3]	"Earth Observation Ground Segment File Format Standard" https://eop-cfi.esa.int/Repo/PUBLIC/DOCUMENTATION/SYSTEM_SUPPORT_DOCS/PE-TN-ESA-GS-0001%20EO%20GS%20File%20Format%20Standard%203.0%20signed.pdf	3.0
[RD 4]	Wikipedia: "Data Format Description Language", https://en.wikipedia.org/wiki/Data_Format_Description_Language	-
[RD 5]	DFDL4S Project, http://eop-cfi.esa.int/index.php/applications/dfdl4s	-

3. INSTALLATION

3.1. Requirements

Software requirements:

- EOCFI libraries v4.25
- Compiler and additional libraries required for the EOCFI libraries (please, consult the software and hardware requirements *here*).

3.2. Installation instructions

To install the library simply unzip the distribution archive for the proper platform (*OS*) (EO_ADAPTER-*X.X*-*CPP*-*OS*.zip) into an installation directory. IMPORTANT: the installation directory must not contain white spaces.

The folder structure resultant of this action is as follows:

- EO_ADAPTER: main folder containing the LICENSE and README files.
- EO_ADAPTER example: Ready-to-use example with a standalone Java program (including a script to compile and run).
- EO_ADAPTER/files: It contains the schema and a template for the configuration file.
- EO_ADAPTER/lib: The EO Orbit and Attitude adapter library

4. USAGE GUIDE

In order to decode a data file using the C++ library, the user has to create its own parser (CUSTOM). The configuration file has to specify that the the parser to be used is the "CUSTOM" one.

The custom parser has to be implemented with the following C++ classes:

- A Class *UserParser*, that inherits the class **AdpDefaultParser**. This class has to override the abstract methods:
  - ***UserParser*::parse(string dataFile):** This method is in charge of reading the data file and filling the output objects with the data.
  - ***UserParser*::parse(const void* dataArr):** This method is in charge of parsing the input generic data in memory and filling the output objects with that data.

  For an example of a custom parser, see the class MyParser in Example 1 and Example 2.
- A Class *UserParserConfiguration* that inherits the class **AdpParserConfiguration**. This class overrides the method **loadFromFile**. This method reads the configuration file, that is inside the path:
  /Earth_Explorer_File/Data_Block/Orbit_Attitude_Adapter_Configuration/Input_Configuration/Parser_Configuration
  If the parser does not need any parameter from configuration, the method could be empty.
  For an example of a custom parser configuration, see the class MyParserConfiguration in Example 1.

Once the custom parser is defined, the following steps should be given:

1. Include the tool package:

```
include <EoOrbAttAdapter.h>;
```

2. Set the adapter main configuration with a file:
   2.1. Create the custom parser configuration
   ```
   UserParserConfiguration parserConfig;
   ```
   2.2. Create the custom parser and set the configuration
   ```
   UserParser myTextParser;
   myTextParser.setParserConfiguration(&parserConfig);
   ```
   2.3. Create the adapter and set parser and configuration with an input file
   ```
   EoOrbAttAdapter myAdapter;
   myAdapter.setParser(&myTextParser);
   myAdapter.setMainConfiguration("./adp_configuration_file.xml");
   ```

3. Parse the data files or the data in memory. The data file can be given as an input parameter or if it is not specified, the data file will be the given in the AdpMainConfiguration object. Note that several data files can be parsed so that all data will be joined together to be returned afterwards as a single file:

```
myAdapter.parse(); // parse the file in the AdpMainConfiguration object
```

   or

```
myAdapter.parse(inputDataFile_1); // inputDataFile_1 = path + file name for 1st file
myAdapter.parse(inputDataFile_2); // inputDataFile_2 = path + file name for 2nd file
myAdapter.parse(inputDataFile_3); // inputDataFile_3 = path + file name for 3rd file
or
myAdapter.parse(inputData); // inputData is data stored as generic pointer (void*);
```

   …

4. Get the orbit/attitude data and/or write the files to disk.

```
OrbitFile myOrbitFile;
```

```
myAdapter.getOrbitData(myOrbitFile);
AttFile   myAttFile;
myAdapter.getAttitudeData(myAttFile);
myAdapter.writeOrbitDataToFile("./output_orbit_file.xml");
myAdapter.writeAttitudeDataToFile("./output_att_file.xml");
```

if the file name is not provided in the writing methods, the file name is taken from the **AdpMainConfiguration** object. If the file name is not yet in the configuration, then an automatic name is given following the file format standard.

4.1.1. Examples

Examples are provided in *installation_folder/*examples/.

The explanation of the examples are inside the .cpp files.

For instructions to compile and run the examples, see the README file provided in the example directory.

4.1.1.1. Example 1

This example consists in parsing a text file using a parser implemented by the user (Custom Parser). It is implemented in three classes:

- EoAdapterCustomAdapterExample: contains the main program where the input text file is parsed and written to disk as an EOCFI orbit file.
- MyParser: it implements a custom parser consisting in reading a text file with the orbital state vectors organised in lines.
- - MyParserConfiguration: it contains the configuration for MyParser. It implements the method loadFromFile that reads a parameter from the configuration file.

4.1.1.2. Example 2

This example consists in parsing the data in memory using a parser implemented by the user (Custom Parser). It is implemented in three classes:

- EoAdapterCustomAdapterExample: contains the main program where the input data is parsed and written to disk as an EOCFI orbit file. Note that the data to be parsed is got calling a simple function called *getDataToParse.*
- MyParser: it implements a custom parser consisting in:
  - Casting the input generic data (void*) to the suitable data type (vector<string>)
  - Interpreting the text lines stored in the vector to get the orbital state vectors.
- - MyParserConfiguration: it contains the configuration for MyParser. It implements the method loadFromFile that reads a parameter from the configuration file.

5. CONFIGURATION FILE DESCRIPTION

Operations of the adapter are controlled by a main configuration file. This configuration file is compliant with Earth Observation file format standard. 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>...</Default_Values>
            <Parser_Configuration type=”DFDL4S” version=”1.3”>...</Parser_Configuration>
            <Common>...</Common>
            <Checks>...</Checks>
        </Input_Configuration>
        <Output_Configuration>
            <Orbit>...</Orbit>
            <Attitude>...</Attitude>
            <Common>...</Common>
        </Output_Configuration>
    </Orbit_Attitude_Adapter_Configuration>
</Data_Block>
```

Figure 2 – Main Configuration File skeleton

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

5.1. Input Configuration

The <Input_Configuration> tag contains the following data:

- Default Values: contains the tag Input_Data_Filename, that defines a default filename for the input data file. This file name is used only in case that the user did not provide explicitly a data file name in the tool inputs.

```
    <Default_Values>
        <Input_Data_Filename>...</Input_Data_Filename>
    </Default_Values>
```

Figure 3 – Main Configuration File: Input configuration, Default_Values section

- Parser Configuration: contains specific data for the parser configuration. In general the parser configuration tag has the following structure:

```
<Parser_Configuration type=”...” version=”...”>
   <Schema_Filename>...</Schema_Filename>
   <Orbit>...</Orbit>
   <Attitude>...</Attitude>
</Parser_Configuration>
```

Figure 4 – Main Configuration File: Input configuration, Parser_Configuration section

The content of the tags <Orbit> and <Attitude> are different depending on parser to be used. For the custom parser, it can be filled with tags chosen by the users to their needs. For the predefined parsers (as for the DFDL4S, not available for C++ library) the data structure is fixed (see the following subsections).

For the custom parser, the reading of the tags inside <Orbit> and <Attitude> has to be implemented by the user. For this, the user has to create a class that implements the interface for AdpParserConfiguration. This interface contains a method (loadFromFile) where the <Parser_Configuration> can be read.

- Common: This section contains:

- The Model section: it can be used to specify the model 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:
  - FIXED_CORRELATIONS: the fields UTC_UT1, UTC_TAI, UTC_GPS will be used (e.g. UTC_UT1 = UTC time – UT1 time in seconds).
  - NONE: All time correlations will be set to 0 (no time correlations).
  - For the following types, the time correlations are initialized using the set of input files (They have to be provided ordered by time). The type can be any of these:.
    - FILE: One input time correlation file that will be detected automatically
    - IERS_B_PREDICTED: IERS Bulletin B table 1 (predicted)
    - IERS_B_RESTITUTED: IERS Bulletin B table 2 (restituted)
    - FOS_PREDICTED: FOS Predicted orbit files
    - FOS_RESTITUTED: FOS Restituted orbit files
    - DORIS_PRELIMINARY: DORIS preliminary orbit files
    - DORIS_PRECISE: DORIS precise orbit files
    - DORIS_NAVIGATOR: DORIS navigator files.
    - OSF: One orbit scenario file.
    - IERS_A_ONLY_PREDICTION: IERS bulleting A (prediction table)
    - IERS_A_PREDICTION_AND_FORMULA: (prediction table and extrapolation formula)
    - IERS_B_AND_A_ONLY_PREDICTION: IERS bulletin B and prediction table from bulletin A (the files have to be provided in this order, first the bulletin B and then the bulletin A)

```
<Common>
    <Model type=”...”\>
    <Time_Correlations type="...">
        <Time_Correlations_File>...</Time_Correlations_File>
        <Time_Correlations_File>...</Time_Correlations_File>
        <Default_Values>
            <UTC_UT1 unit="s">...</UTC_UT1>
            <UTC_TAI unit="s">...</UTC_TAI>
            <UTC_GPS unit="s">...</UTC_GPS>
        </Default_Values>
    <Time_Correlations>
</Common>
```

Figure 5 – Main Configuration File: Input configuration, Common section

- Checks: This section is optional (new from version 1.1). The parsed orbit and attitude data will be checked to look for different incongruent data, according to the following parameters. Those orbit/attitude records suspicious of wrong data can be reported.
  - Orbit_Checks:
    - Time gaps validation:
      - Maximum_Gap: maximum allowed time interval between orbit records
      - Time_Step: expected time interval between records.
      - Duplicated_Threshold: Two records will be considered to be duplicated if they differ less than this value.
      - Time_Step_Threshold: Threshold to consider than the time interval is kept within its expected regular interval.
  - Attitude_Checks:
    - Time gaps validation:
      - Maximum_Gap: maximum allowed time interval between records
      - Time_Step: expected time interval between records.
      - Duplicated_Threshold: Two records will be considered to be duplicated if they differ less than this value.
      - Time_Step_Threshold: Threshold to consider than the time interval is kept within its expected regular interval.

```
<Checks>
    <Orbit_Checks>
      <TimeGapValidation>
```

```
            <Maximum_Gap units="s">5</Maximum_Gap>
            <Time_Step units="s">1</Time_Step>
            <Duplicated_Threshold units="s">0.1</Duplicated_Threshold>
            <Time_Step_Threshold units="s">0.05</Time_Step_Threshold>
          </TimeGapValidation>
        </Orbit_Checks>
        <Attitude_Checks>
          <TimeGapValidation>
            <Maximum_Gap units="s">2</Maximum_Gap>
            <Time_Step units="s">1</Time_Step>
            <Duplicated_Threshold units="s">0.001</Duplicated_Threshold>
            <Time_Step_Threshold units="s">0.05</Time_Step_Threshold>
          </TimeGapValidation>
      </Attitude_Checks>
</Cheks>
```

- ***Figure 6 – Main Configuration File: Input configuration, Checks section***

5.1.1. DFDL4S Parser configuration (not available for C++ library)

As described above, the parser configuration tag contains three sections: the <Schema_Filename>, the <Orbit> and the <Attitude>, being the <Orbit> and the <Attitude> the specifics parts. For the DFDL4S are as follows:

```
<Parser_Configuration type=”DFDL4S” version=”1.4”>
   <Schema_Filename>...</Schema_Filename>
   <Orbit status=”enabled”>...</Orbit>
      <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="true">...</Orbit_number>
         <Quality default="true">...</Quality>
         <Default_Values>
            <Quality>...</Quality>
            <Orbit_Number type="OSF">
               <OSF_Filename>...</OSF_Filename>
            </Orbit_Mumber>
         </Default_Values>
      </Mapping>
   </Orbit>
   <Attitude status=”enabled”>
      <Mapping>
        <Mapping_Quat>
          <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_Quat>
      </Mapping>
      <Axis_Mapping>
           <X>...</X>
           <Y>...</Y>
           <Z>...</Z>
      </Axis_Mapping>
   </Attitude>
</Parser_Configuration >
```

Figure 7 – Main Configuration File: Input configuration, Parser_Configuration section for DFDL4S parser

The Orbit section in the input configuration contains:

- Attribute *status*: it can be "enabled" or "disabled". When it is disabled, the orbit data will not be parsed.
- The Mapping section: it defines the correspondence between items defining an orbit and elements in the DFDL4S schema. The type of mapping is defined by the *type* attribute (currently only *OSV* is allowed). The elements are described as a path from the root node. In particular:
  - *Time_Reference*: either TAI, UTC, UT1 or N/A; It is the time reference for the read time;
  - *Time*: the OSV epoch;
  - *Reference_Frame*: Reference frame of the OSV (BAR_MEAN_2000, HEL_MEAN_2000, GEO_MEAN_2000, MEAN_DATE, TRUE_DATE, EARTH_FIXED, BAR_MEAN_1950, QUASI_MEAN_DATE, PSE_TRUE_DATE, PSEUDO_EARTH_FIXED);
  - *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:* Orbit number of the OSV. It contains the attribute *default*(=true or false).When *default* is true, the orbit number is not read from the file with the path but computed with an OSF specified below, in the *Default_Values* section.
  - Quality: quality value of the OSV; It contains the attribute *default*(=true or false).When *default* is true, the *quality* is not read from the path specified, it will have the value specified below, in the *Default_Values* section.
  - *Default_Values*: this section is used when the attribute default is set to true. It contains:
    - A default value of *Orbit_Number* (if its attribute *type* is set to "OSF" (currently the only value allowed), the Orbit number will be calculated using an Orbit Scenario File).
    - A default value for Quality; All OSV will have the same quality value.

The Attitude section has a structure similar to the Orbit section:

- Attribute *status*: it can be "enabled" or "disabled". When it is disabled, the attitude data will not be parsed.
- The Mapping section: it defines the correspondence between items defining an attitude and elements in the DFDL4S schema. The type of mapping is defined by the *type* attribute, it could be "Quaternions" or "Angles".
  For Quaternions:
  - *Time_Reference*: either TAI, UTC, UT1 or N/A; It is the time reference for the read time;
  - *Time*: the quaternion epoch;
  - *Reference_Frame*: Reference frame that is the origin frame for the rotation represented by the quaternions (BAR_MEAN_2000, HEL_MEAN_2000, GEO_MEAN_2000, MEAN_DATE, TRUE_DATE, EARTH_FIXED, BAR_MEAN_1950, QUASI_MEAN_DATE, PSE_TRUE_DATE, PSEUDO_EARTH_FIXED);
  - *Q1, Q2, Q3, Q4*: Path for the Quaternions elements. Quaternions numbering follows the EOCFI conventions (i.e. Q4 is the real component).

  For Angles:
  - *Time_Reference, Time and Reference_Frame*: as defined for the quaternions.
  - *Pitch, Roll, Yaw*: Path for the rotation angle elements.
- *Axis_Mapping:* Attitude data represent a transformation from two co-ordinate systems, for example GM2000 to satellite attitude frame. 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: <X>Y_positive</X> means that the EOCFI X axos os the Y positive axis in the data convention.

5.2. Output Configuration

The <Output_Configuration> tag has the following tags (see Figure 8):

- Orbit: contains the configuration data required for the output orbit file.
- Attitude: contains the configuration data required for the output attitude file.
- Common: Contains the tag <Satellite_Id> with the satellite/mission name.

The <Orbit> and <Attitude> tags are have a similar structure. The content is as follows:

- Attribute Type: it can be:
  - EOCFI_FILE: the output is written to a EOCFI compliant file.
- Default_Filename: this is the filename used in case no filename is provided via the adapter interface. If the Default_Filename field is empty and no output file is specified via the adapter interface, the file name will be generated as per File Format Standard.
- Time_Reference: Default Time reference. (Allowed values: TAI, UTC, UT1, N/A).
- Reference_Frame: For the orbit file, it is the reference frame of the state vectors. For the attitude file, it is the reference frame for the source inertial frame. (Allowed values: BAR_MEAN_2000, HEL_MEAN_2000, GEO_MEAN_2000, MEAN_DATE, TRUE_DATE, EARTH_FIXED, BAR_MEAN_1950, QUASI_MEAN_DATE, PSE_TRUE_DATE, PSEUDO_EARTH_FIXED)
- Target_Frame (only for Attitude): Attitude target frame (Allowed values: Sat_Nominal_Attitude, Sat_Attitude, Instr_Attitude).
- Header_Configuration defines the header fields as defined by the File Format Standard that cannot be generated automatically by the adapter:
  - Notes
  - File_Class
  - File_Type
  - File_Version
  - Source/System

```
<Output_Configuration>
  <Orbit type="EO_FILE">
    <Default_Filename>...</Default_Filename>
    <Time_Reference>...</Time_Reference>
    <Reference_Frame>...</Reference_Frame>
    <Header_Configuration>
        <Notes>...</Notes>
        <Mission>...</Mission>
        <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>...</Target_Frame>
    <Header_Configuration>
        <Notes>...</Notes>
        <Mission>...</Mission>
        <File_Class>...</File_Class>
        <File_Type>...</File_Type>
        <File_Version>...</File_Version>
        <Source>
            <System>...</System>
        </Source>
    </Header_Configuration>
  </Attitude>

  <Common>
     <Satellite_Id>...<Satellite_Id>
  </Common>
```

```
<Output_Configuration>
```

Figure 8 – Main Configuration File: Output configuration

6. USER INTERFACE

The Orbit Attitude Adapter tool library consists in a set of C++ classes. The API of the library can be found in the HTML documentation: TBD