ASTROLABE

Version 1.0

Interface Control Document

ABSTRACT

This interface control document (ICD) is the specification (SP) of the GEMMA system's ASTROLABE 1.0 data interfaces.

DOCUMENT STATUS

Document title:	ASTROLABE 1.0 Interface Control Document
Document reference:	ASTROLABE_1-0_SP_ICD
Document file name:	ASTROLABE_1-0_SP_ICD.pdf
Nature:	<report (RP) | **specification (SP)** | note (NO) | minutes (MI) | others (OT) >
Status:	<in preparation | **checked** | approved | authorized>
Author(s):	M. Eulàlia Parés (CTTC), José A. Navarro (CTTC), Ismael Colomina (CTTC, currently at GeoNumerics S. L.)
Version:	1.0.15
Last change:	2019-12-09T13:29:28
Distribution:	<team | project | organisation | **public**>
Security classification:	<**public** | internal | confidential | secret>
Number of pages:	107

DOCUMENT CHANGE RECORD

Version	Date	Section(s) affected	Description of change, change request reference or remarks.
0.9	2015-07-31	all	First public deliverable version.
1.0	2016-02-05	all	Overall review. Extended version.
1.0.1	2016-04-08	1.1, 3.2, 3.5, 3.6.1.1, 3.6.1.2, 3.6.2, 3.7, 3.7.2.3, 3.7.2.4, 3.7.3.1.1 3.7.x.1.1 (3 >= x >=7)	Correction of small typographical errors. Subsections move from heading level 5 to 4, thus being reduced by 1 the number of levels in the subsection number. For instance, former subsection 3.7.3.1.1 is now 3.7.3.1.
1.0.2	2016-04-21	2.2.2.2 3.6.1.2 3.6.1.3	Corrected typographical errors (unknown parameters changed to known parameters in two places). It was wrongly stated that <i_spec> was very similar to <p_spec> and <l_spec>. Example 7: minor corrections. Example 8. minor corrections.

		3.6.1.4	Example 9: minor corrections, affecting the line numbering. The text in this section referring to these lines has been also adapted to these changes.
		3.7.2.2	Clarification on how the “s” attribute works.
		3.7.10	Corrections to Table 14.
1.0.3	2016-04-29	3.6.1.1.9	Small typographic errors corrected.
		3.6.1.2	Example 7: removed incorrect <c> and <s> tags in <a_spec>. Corrected erroneous indentation.
1.0.4	2016-09-02	3.5 3.7.1.3 3.7.1.4 3.7.1.5.2 3.7.1.5.3 3.7.3.1 3.7.5.1 3.7.6.1	Normalize I- and o-records identifiers, changing attributes l_id and o_id to id in both cases. Introduce the n attribute to state the instance identifier. Rearrange the order in which I- and o-records attributes appear to make identification more visual. Correct related examples.
		3.7.2.3 3.7.11	Correction of typos.
1.0.5	2016-09-07	3.5 3.6.1.1 3.6.1.1.1 3.6.1.2 3.6.1.2.1 3.6.1.2.3 3.6.1.3 3.6.1.3.1 3.6.1.4 3.7.3.1	Substitution of several <*_type> tags in metadata by a single <type> tag.
		3.6.1.2.3 3.6.1.3 3.6.1.3.4 3.6.1.3.5 3.6.1.3.6 3.6.1.3.7 3.6.1.4	Substitution of several <*_item> tags in various lists in metadata by a single <item> tag.
		3.5	Short example about instance identifiers included.
		3.7.1.4 3.7.1.5.1 3.7.1.5.3 3.7.1.5.4 3.7.2.4 3.7.2.4.1 3.7.5.1 3.7.7.1	Change of “fi” and “fbi” keywords by more meaningful ones (“text_file” and “binary_file” respectively).
1.0.6	2016-02-26	3.2 3.6.1 3.6.1.4 3.6.1.5	Addition of the <t_tag> (time specification in metadata) object. Note that this change has made section 3.6.1.4 become 3.6.1.5, since a new section with this number has appeared. The numbering of the affected sections on the left

			refer to the new numbering.
		3.7.4	ASTROLABE does not impose the existence of a binary format for instrument files.
1.0.7	2017-04-03	3.2	More accurate definition of the kind of files making the ASTROLABE FI.
		3.6.1 3.6.1.4 3.6.1.5 Examples 9 & 10	Replacement of all occurrences of t_spec by time_spec
		3.6.1.1 3.6.1.1.9 3.6.1.2 3.6.1.2.2 Examples 6 & 7	Replacement of all occurrences of a_spec by t_spec
		3.6.1.1 3.6.1.1.9 3.6.1.2.2 3.7.1.5.2 3.7.3.1 Tables 5, 7 & 15	The concept “auxiliary value” has been replaced by the concept “tag”
		3.5	Former figures 1 & 2 replaced by a single, more comprehensive figure 1. References to old Figure 2 have been corrected to point to new figure 1.
		3.7.1.3 3.7.1.5.2 Tables 5 & 7	State clearly that covariance matrices are optional in l-records and that when omitted these must be specified using metadata and default covariance matrices.
		Tables 5 & 7	Tags are optional. State it clearly.
		3.7.1.3 3.7.1.5.2 Tables 6 & 8	The list of instrument instance identifiers is optional.
1.0.8	2017-05-24	3.4.1	Examples 1 & 2 changed. The text referring to these has been conveniently updated due to the change in line numbers.
		3.6.1	Rewriting of two sentences (no change in meaning).
		3.6.1.1.3	Better wording of a single sentence (again, no change in meaning)
		3.6.1.2	Incorrect closing </c_item_type> in Example 7. Changed to the correct form, </type>.
		3.6.1.3.4	A new section describing the <dynamic> tag has been created. Consequently, former sections 3.6.1.3.4 to 3.6.1.3.7 have changed their numbers from 3.6.1.3.5 to 3.6.1.3.8.

		3.6.1.5	Example 10 has been modified to insert the <dynamic> tag in the examples describing models. The line numbers in the example have changed. Consequently, the text referring to these lines has been updated.
		Table 8	It is now stated that the list of instrument instance identifeirs is optional.
1.0.9	2017-06-22	Example 11	Tag <tr_states_correlation_Rxx> incorrectly shown as mandatory, when it is optional. Now it has been corrected. Removed the “version” attribute at the header of the file.
1.0.10	2017-10-03	3.2	Removed the paragraph describing "Header files for the communications interface". All data files or streams have header files, so this paragraph was redundant.
		3.4	Title changed from "XML-based formats : common traits" to "The file interface: common traits". Contents rewritten to explain that 3.x sections will deal with data files only.
		3.4.1	Complete rewriting. Now it is explained that all ASTROLABE data files consists of two parts, a header plus a data file, whatever the format of the second is.
		3.6	Better wording, some rephrasing.
		3.6.1	Time has been removed from the list of metadata objects to define. It is now part of the other ones. Text, figures and examples now refer to time as a component of the other metadata objects (as <l_spec>, for instance).
		3.6.1.1.7 to 3.6.1.1.9	Numbering changed. Now these are numbered from 3.6.1.1.8 to 3.6.1.1.10, due to the insertion of a new section with number 3.6.1.1.7 (describing the time_spec object)
		Example 5	It now includes a <time_spec> object.
		3.6.1.2	Added a reference to the section defining <time_spec> objects. Included the <time_spec> object in the list of sub-objects making an <i_spec> object.
		Example 6	It now includes a <time_spec> object.
		3.6.1.2.2	Title slightly changed (including now the <time_spec> object). Reference to the section where the <time_spec> objects are defined.
		3.6.1.3.1	Title was incorrect (m_type). Corrected (type).
		3.6.1.4	Removed. There are no more independent <time_spec> objects (now, these are sub-objects of <l_spec>, <p_spec> and <i_spec> tags).
		3.6.1.5	As a consequence of removal of section 3.6.1.4, this one's name is now 3.6.1.4 instead of 3.6.1.5.
		3.6.1.4	Since Example 5 has been changed (see next log entry) all

			references to line numbers in the aforementioned example has been updated. All explanations related to the former, unique <time_spec> tag have also been removed.
		Example 9	Because of the removal of the aforementioned section, that included one example, this one is renamed from 9 to 8.
		Example 8	New example, removing the unique <time_spec> object valid in former definition. Now, <time_spec> appears in all <l_spec>, <p_spec> and <i_spec> objects.
1.0.11	2017-10-04	3.6.2	Slight rephrasing of some paragraphs.
		3.7	Renamed from "Data formats and files" to "Data files and formats"
		3.7.1	A new section, named "The header accompanying actual raw data files" has been created, with number 3.7.1. This changes the numbering of former section 3.7.1 to 3.7.11, that now is 3.7.2 to 3.7.12. This new section describes the format of ASTROLABE (data) header files. Both an example and a table (numbers 11 and 5 respectively) have been included. The numbering of later examples and tables has been, obviously, affected (increased by one everyone after the places where the new example and table have been inserted).
		Example 1	It has been slightly modified to show more clearly that it describes a socket connection. The footer of the example has been changed accordingly.
		3.7.2.4	Rewritten partially to show that now all obs-e files consist of a header and external data, that is, the concept of embedded header + data files disappears. Examples 12 & 13 are modified accordingly. Rewritten also to show that there are no embedded text formats anymore. Figure 4, "Hierarchical depiction of the available <obs_e-file> formats", has been removed, since it does not make sense now (the set of formats has been reduced to two.)
		3.7.2.5.1	It has been renamed (new title: Raw data (text or binary) files. Organization. Naming conventions). Slighty rewritten to remove the concept of "external" (versus embedded) files. Example 14 changed (and legend modified) to adapt it to the latest header file syntax.
		3.7.2.4.2	It has been renamed (new title: The format of raw text data files). Again, slightly rewritten to remove the concept of external vs. embedded text files. Example 18 changed accordingly.
		3.7.2.5.3	This section ("External text files") has been completely removed, since it makes no sense now. Examples 19 & 20 have disappeared too. Consequently, section 3.7.2.5.4 is now 3.7.2.5.3, and all the examples appearing after the former 20th one have seen their numbers reduced by two.
		3.7.2.5.3	This section, formerly 3.7.2.5.4, has been renamed (new

			title: the format of binary files). Subsection 3.7.2.5.3.1 has been merged with this section. Example 19 has been removed, since it makes sense any more (later examples get their numbers decreased by one).
		3.7.3.4	Section renamed (new title: The organization of the <r-matrix_file>. Text and binary formats). Rewritten to eliminate any references to embedded / external r-matrix files. Example 19 changed accordingly (now it contains a header file for r-matrix data). Example 22 removed. Later examples are automatically changed decreasing one unit their numbers.
		3.7.3.4.1	The section header is removed and its contents merged with its parent section (3.7.3.4). Slight rewriting to refer to the now standard header files. Example 22 removed (it is redundant). Another example added (header file) so no renumbering takes place on further examples.
		3.7.5.1	Section simplified, including the example (not showing unnecessary headers). Description of example shortened by this reason.
1.0.12	2017-10-04	3.7.6	Very few changes to remove references to embedded / external data files.
		3.7.6.1	Again, the references to embedded / external data files have been removed. Example 25 changed to include a header file instead of an obsolete header. Explanations about examples 25 and 26 rewritten.
		3.7.7	Slight changes (external / embedded files).
		3.7.7.1	Slight changes, once more because of the same reason. Example 27 modified to remove the obsolete headers (line numbering has changed). Corrected the text referring to line numbers in this example.
		3.7.8.1	Usual changes to remove any references to embedded / external files. Example 28 replaced (now is a valid header, instead of the obsolete one). Explanations referred to example 28 conveniently modified.
		Table 15	Corrected to remove references to embedded files.
		3.7.12	Shortened: now the description of the header files defining socket data is included in section 3.7.1, so it was redundant here. Rephrasing of some sentences to refer to the new header files.
		4.1	Slight rephrasing to refer to header files correctly (avoiding the old, obsolete naming).
		4.2	Style correction (a single sentence). Stated clearly that the observation record is used to transmit not only observations but also parameters and instruments. Added the correlation matrix (data) record (including descriptive table).

		4.2.4	Removed the precedences set between data blocks. Stated that it is the responsibility of each application using the CI to define which blocks of data and in which order these will be transmitted.
1.0.13	2017-10-09	3.6.1.1.5	New syntax and semantics of CRF, RF and CS codes. Partially rewritten.
1.0.14	2017-10-30	3.6.1.1.5	Final rewriting of the section.
1.0.15	2017-11-21	3.6.1.1.6	Wrong units example corrected. Added explanation on how to write dimensionless units.

QUICK TABLE OF CONTENTS

FULL TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EXAMPLES

1 INTRODUCTION

Einfeichheit ist das Resultat der Reife.
(Simplicity is the result of maturity.)
– *Friedrich Schiller*

ASTROLABE stands for "**A ST**andard **R**epresentation **O**f time-tagged, **L**oc**A**tion-**B**ased data for **E**xchange purposes".

1.1 Purpose, scope and approach

The purpose of this document is to describe CTTC's ASTROLABE data specification, which comprises the different formats defined by the CTTC to represent observation-event data and related metadata.

The document does not describe, in general, interfaces to any navigation or positioning systems, unless they implement official or popular de facto standards and standard implementations (usually formats).

The definition of the interface shall be funded in the fact that it should not complicate the extensibility of the system. Therefore, the definition of such interfaces should take into account future extensions of the system. An incorrect interface definition leads to an under-utilization of the capabilities of the system and to an increase on development efforts. The interface proposed on this document is based on a philosophy firstly presented in the 80's for the ICC's photogrammetric software GeoTeX [1]. Later on, the same idea was further developed for the interface definition of the Geonumeric's generic block adjustment platform GENA [2]. ASTROLABE assumes the principles of the fundamental reference modeling of those interfaces and includes new features related to time dependent issues as well multi-sensor modelling.

The ASTROLABE "space of interfaces" has two dimensions: the File Interface (FI) and the Communications Interface (CI). The FI describes the file formats used to store observation-event data and related metadata. The CI describes how observation-event data should be transferred between different processes or even computers, using TCP / IP socket technology. Future versions of ASTROLABE may include other communication technologies.

1.2 Organization of this document

The organization of this ICD directly mirrors the structure of the ASTROLABE interface, which is described thoroughly in chapters 3 (the file interface) and 4 (the communications interface). Chapter 2 presents an overview of ASTROLABE, stating its design principles and describing the four abstraction pillars on which this interface is based. Chapters 5, 6 and 7 cover the usual need to document acronyms, definitions and provide with a (short) bibliography on the subject.

2 OVERVIEW OF THE ASTROLABE INTERFACE

2.1 The ASTROLABE interface design principles and context

The design principle of the ASTROLABE interface is usability and correctness. To achieve those the interface shall be as simple as possible. Furthermore, ASTROLABE data is founded on the fact that it may in no way be an obstacle to the extensibility of modern trajectory determination systems.

An incorrect – or non-existent – abstraction of the data entities represented in a data interface may lead to an unnecessary complexity in the software using such interface. A bad data model may also compromise the correct evolution of such software systems, since the incorporation of – apparently – new data entities will imply the ad hoc modification of the source code to cope with such changes.

Specifically, in the context of trajectory determination systems, sensors play a very important role, since these are the main source of data to deal with. Sensors evolve continuously, and this continuous change must find its way in a correctly defined data interface if the aforementioned issues are to be avoided.

The answer to this challenge is the proper definition of an abstraction describing the essential properties of the data involved in the interface.

Generally speaking, a model is an abstract (or actual) representation of an object or system from a particular viewpoint. There are functional, stochastic, mathematical, technical, geomatic, computer and SW models (see section 6 for a formal definition of each and Figure 1. for their hierarchical relationship).

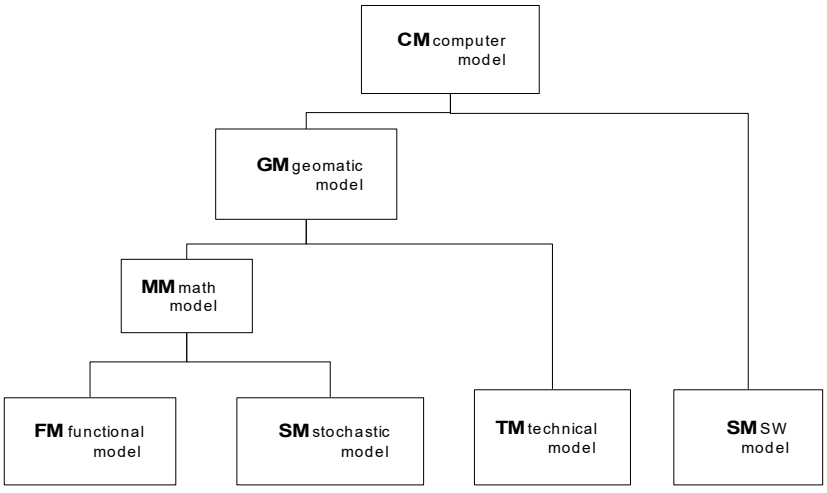

Figure 1. The ASTROLABE tree of models

The ASTROLABE data abstraction (computer model) identifies four different concepts (pillars) on which its navigation-oriented interface is based: (input) measurements, (input) auxiliary instrument constant values, (output) states and (equations) models relating all those elements. The modelling of the information in these four categories (observation/measurement, state, model and instrument) also fosters the utilization of object oriented modelling of the systems using the ASTROLABE interface, thus enabling such powerful mechanisms as inheritance, encapsulation and polymorphism so common in well designed software. It could be said that a proper data abstraction leads to a (mirrored) correct software abstraction, able to cope with continuous change.

If not otherwise specified, in the ASTROLABE context, model will refer to a mathematical relation between the three entity objects (instrument, observation, parameter) of the ASTROLABE fundamental reference model (computer model).

Hereafter a detailed definition of each of those four pillars is presented; that is, the ASTROLABE interface is described.

2.2 Observations, parameters-states, instruments and models

2.2.1 Observables and observations

An **observable** (a noun) is a numerical property of a physical system that can be determined by a sequence of physical or mathematical operations. Technically, it is a random variable.

An **observation** is one of the values that an observable or random variable may take; i.e., it is a random sample of a random variable. (For example, the various repeated measurements between A and B of a distance meter instrument –the measured distances– are observations and the abstract concept of distance between A and B is the observable.)

Observations may be associated to static and dynamic models. Although it is theoretically incorrect, when it is clear from the context, we may talk about *static* and *dynamic* observations. A static observation will refer to an observation that participates in a static model; a dynamic observation will be an observation that participates in a dynamic model.

2.2.2 Parameters-states

A **parameter** (non-time dependent) and a **state** (time-dependant) are random variables whose expectation and covariance have to be estimated from known observations, instruments and models.

2.2.2.1 Time dependent and time independent (random constant) unknowns

In trajectory determination systems, the vast majority of unknowns are time dependent; that is, they are vectors of time dependent random variables or multidimensional stochastic processes. However, some parameters may just happen to be random constants of unknown expectation vector and covariance matrix. These random constants will be referred to as time independent parameters and shall not be confused with constant parameters (see next section below). A time independent parameter shall be estimated by the trajectory determination system and its value may change from one mission to the other.

Technically speaking, a **time dependent unknown** or **state** is a random process which in general is not a random constant.; and a **time independent unknown** or **parameter** is a random constant.

2.2.2.2 Constant known parameter / states

A **constant known** is a parameter or state whose expectation and covariance are known and shall not be estimated. (As we will see later on, the expectation and covariance of a constant parameter can be set with pseudoobservations.)

A common use of constant known parameters (states) is that of model parameters that cannot be determined due to limited data or weak configuration circumstances. Thus, for instance, in GNSS

navigation the ephemeris parameters are usually not estimated but set to constant and assigned with pseudoobservations.

2.2.3 Instruments

An **instrument** or a **sensor** is a device used to measure. An instrument is a software entity which contains the constants that characterize an actual instrument. It is one of the four objects (instrument, observation, parameter or state and model) of the ASTROLABE fundamental reference model [4].

2.2.4 Models and observation equations

A **model** is a mathematical description of a system or process. In the context of ASTROLABE, it is a stochastic equation or a stochastic differential equation; traditionally, this has also been described as a functional model plus a stochastic model. The model describes the relation of states, observations and instruments.

An **observation equation** will be the mechanism provided by the data abstraction to relate specific observations, states and instruments to the models that have to be used to estimate the new state.

3 FI – THE FILE INTERFACE

3.1 General overview and approach of the ASTROLABE FI

The ASTROLABE FI comprises a variety of files and formats, representing different data entities. Examples of these data entities would be observations collected from diverse sensors, the equations that relate these or metadata describing – obviously- other data entities.

Most files in the ASTROLABE FI are text files, although binary formats are used too because of performance reasons. Compressed files including both text and binary files also exist.

Most formats of the ASTROLABE FI for text files are XML based ones. XML is a well spread and supported data exchange standard, whose main advantages – as for instance, simplicity or extensibility – makes it the right choice to represent these data entities in the context of the ASTROLABE FI.

3.2 The files in the FI at a glance

The ASTROLABE FI comprises the following types of files:

- Metadata (for time, observations / parameters (states) / instruments and models) files. Serve to describe the characteristics of time, observations, parameters, instruments and mathematical models.
- Navigation directory (metadata) file. Used to list the files, either input or output, involved in the process of estimating trajectories.
- Data files. The main kind of data file is the *observation-event* file. It is used to store either input observations (measurements) and the equations relating these, input instrument files – containing the set of constants that characterize these –, and output estimated trajectories. Files storing *output correlation matrices* for observations, parameter (states) and their residuals are also part of the ASTROLABE FI.
- Optionally, and using free formats not defined by ASTROLABE, an input option file (describing the options used to estimate output trajectories) and an output report (log) file logging the evolution of the output trajectory estimation process.
- Navigation file. A compressed (in zip format) file containing all the aforementioned files involved in the process of estimating trajectories, either input or output, metadata or data.

Later sections of this document describe the format used by all the files in the list above.

3.3 FI general characteristics

Most files in the ASTROLABE FI are text XML files. Although the XML standard defines a syntax that must be respected to create well-formed files, it is, at the same time, flexible enough to allow infinite variations.

For instance, XML does not define a fixed set of tags, which may be created freely to meet the needs of any kind of data entity to be represented as an XML file. Item plus, there are no restrictions on capitalization of tags or values to include in such files.

Additionally, ASTROLABE XML files use tags or reserved words that need to be specified in the different sections describing these files. To avoid misinterpretation, the next subsections explain formally how such descriptions will be presented to the reader.

3.3.1 Nomenclature conventions of the FI

To avoid confusion between the data valid representations and codes and the actual text, its punctuation marks and associated graphic characters, all representations are contained in brackets [ ]. The brackets are not part of the representation and should be omitted when implementing the representations. The XML tags start with the symbol “<” (less than) and end with the symbol “>” (larger than), like in <dim>. The symbols “<” and “>” belong to the tag themselves.

All matter outside the brackets [], <> and the XML start-tags and end-tags is normal text, and not part of the representation. In the associated examples, the brackets [] and typographical markings are omitted. Thus, for instance, we will write “the recommended extension for the file of type <nav_file> is [nf]” and not “... for the file of type <nav_file> is nf.” Or, we will write “the two valid values for the XML field <role> are [FREE], [CONSTANT], [free] and [constant]” instead of “the two valid values for the XML field <role> are FREE, CONSTANT, free and constant.” Note, however, that this will translate into, for instance, the following illustrative example

```
<role> FREE | CONSTANT | free | constant </role>
```

instead of the incorrect one

~~<role> [FREE] | [CONSTANT] | [free] | [constant] </role>~~.

3.3.2 File, XML-tag and ASTROLABE-keyword letter case

In the ASTROLABE FI, the XML tags of the various files formats are case sensitive. This is an XML rule.

In the FI, some tags require information like the name of a person or the name of an organization. In these cases, the content between the start-tag and the end-tag is freely given in ASTROLABE by the user. Other tags, require a keyword from a valid set of keywords. Typical examples thereof, are keywords like [YES], [NO], [constant] and [FREE]. In these cases, unless otherwise stated, keywords’ letters shall be simultaneously either capitalized or low case. Thus, [YES] and [yes] will be accepted by ASTROLABE, but not [YeS]. This decision balances user’s freedom and parsing complexity.

To avoid unnecessary complexity and to easy the reading of this document, when specifying an interface object (an XML tag pair) only one of the two possibilities discussed above will be presented. Thus, it will be written either

```
<role> constant | free </role>
```

or

```
<role> CONSTANT | FREE </role>
```

instead of the formally more correct

```
<role> constant | free | CONSTANT | FREE </role>.
```

Another sensible issue is file naming conventions, letter case and file systems in different operating systems.

In several sections of this ICD rules are given to build the names of files upon user-input codes of several kinds. Some other sections show how to introduce the names of input or output data files into the system.

Different operating systems handle letter case in file names in disparate ways. For instance, the Windows family of operating systems makes no difference between lower and upper case. On the contrary, both Linux and *NIX operating systems do differentiate files upon letter case.

This means that the files with names

```
CODE1_code2_CodeThree.ext
```

and

```
code1_Code2_CODETHREE.Ext
```

are the same one in Windows but different in Linux and *NIX.

Software using ASTROLABE must follow the Linux and *NIX convention as much as possible (see below). Whenever a file name is built upon a series of user-input codes, the letter case must be preserved.

For example, if a file name is to be built according the following rule:

```
<code1>_fixed-text_<code2>.ext
```

and the actual values of <code1> and <code2> are respectively [PREFIX] and [suffix], the file name built by the software using ASTROLABE must be

```
PREFIX_fixed-text_suffix.ext
```

In summary, letter case must never be tampered with.

Developers of software using ASTROLABE must be aware of this fact and should always provide with files with a name that exactly matches the letter case used in the file name specification.

However, the alert reader must keep in mind that Windows systems will never complain about a difference in letter case, thus leading to a false sense of security. Linux and *NIX certainly will always complain about this situation.

3.4 The file interface: common traits

The ASTROLABE file interface comprises several kinds of files, including data and metadata. The following sections will describe the common traits of the data files found in such file interface. Metadata files will be described in detail in section 3.6 unless otherwise stated.

3.4.1 The overall, common structure

ASTROLABE data files always consist of at least **two** parts:

- An XML header file stating what kind of data is stored in
- the actual data file.

ASTROLABE header files take care of defining every aspect of the accompanying data file, as the kind of data being stored (i.e., observations, instruments, etc.) as well as the kind of storage (file or socket stream) and the format (pseudo-XML text files, binary files,

Header files are described in detail in section 3.7.1. Example 1 depicts a valid ASTROLABE header file describing a socket data stream (communications interface, see section 4).. Note the inclusion of a <lineage> tag. Such <lineage> block is used in all the XML-based files present in the file interface. It is described in section 3.4.2.

Data files may be of several types, containing observations, states, instruments, or correlation matrices, for instance (see section 3.7). The contents of these will directly depend on the kind of data they store. Data files may be stored in either text (pseudo-XML) or binary formats.

Both text and binary data files may be split in several fragments for security reasons. Data files may be generated in real time by applications directly capturing data from sensors. An unexpected malfunctioning in such applications may lead to data corruption. Splitting the output in several files serves to guarantee that at least all files but the last one will contain correct data (see section 3.7.2.5.1 for further details).. Figure 1 depicts the header / data file tandem and the ability to split data file in several fragments or chunks.

```
<?xml version="1.0" encoding="UTF-8"?>
<astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
  <lineage version="1.0">
    <id> id1 </id>
    <name> Header file describing data sent through sockets </name>
    <author>
      <item> J. Navarro </item>
    </author>
    <organization> CTTC </organization>
    <department> GEON </department>
    <date_time>2016-10-18T11:41:47-05:00</date_time>
    <ref_document>
      <item> ASTROLABE ICD </item>
    </ref_document>
    <project> GEMMA </project>
    <task> ASTROLABE interface control document </task>
    <remarks>
      Header file to describe obs-e data sent by means of a socket connection.
    </remarks>
  </lineage>
  <data>
    <device type="obs-e_file" format="socket"> localhost:2000 </device>
  </data>
</astrolabe-header_file>
```

Example 1: An ASTROLABE header file used to describe obs-e data sent by means of sockets

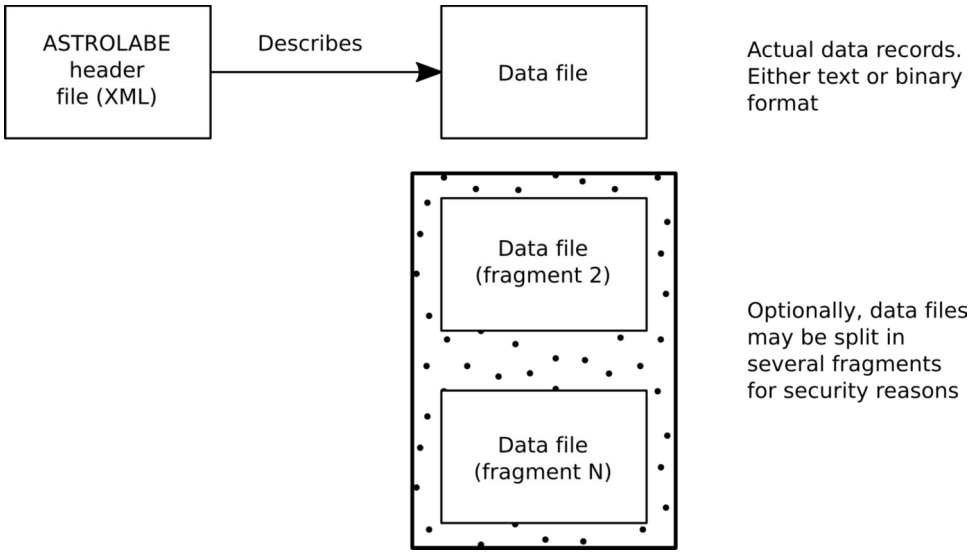

Figure 1: Overall structure of ASTROLABE data files

With regard to the lineage tag, present for instance in the aforementioned header files (see again section 3.4.2), it must be said that it characterizes some data field or complete data set. Lineage tags may be found in several places in the file interface.

Depending on the contents of the file (either data or metadata), the following situations are possible:

- A single lineage block, describing the history of the whole file or data set, appears just after the header block.
- No lineage block appears after the header, but one or more elements in the specific set of tags (the third block) include their own lineage section, which describe the history of these specific

elements.

Header files fall within the first category; metadata files are examples of the second case described above.

There are many examples of complete ASTROLABE XML files along this document. See the list of examples at the beginning for a complete list of these. Examples 8 to 29 are the most appropriate for the purpose of understanding how these files are.

3.4.2 The lineage type

The lineage type is used to describe the history of a field or data set. All XML-based ASTROLABE files include one or more lineage types as stated in section 3.4.1.

It is a flexible type whose only mandatory field is the identification code -the “id” - of the data set. This identification code (tag: <id>) is a very important element in ASTROLABE, since it is used to cross-reference different entities when working with metadata or data files (see the discussion on section 3.5 about types and identifiers and how these are used to relate actual data and metadata).

A <lineage> serialized object file is defined here through a layout of its XML representation. All fields are optional with the exception of the identification field (corresponding to the XML tag <id>) whose function is to uniquely identify the object –that is, no two <lineage> objects may have the same value for their <id> tag. For this type of objects, unless otherwise stated, the type of the elementary basic fields is “string.” The multiplicity of the fields nested within a higher level optional field is obviously subject to the existence of the higher level field.

Level	XML tag	#	type	m/o
1	<id>	1	string	m
1	<name>	0 or 1	string	o
1	<author>	0 or 1		o
2	<item>	>=1	string	m
1	<organization>	0 or 1	string	o
1	<department>	0 or 1	string	o
1	<date_time>	0 or 1	ISO standard	o
1	<ref_documents>	0 or 1		o
2	<item>	>=1	string	m
1	<project>	0 or 1	string	o
1	<task>	0 or 1	string	o
1	<remarks>	0 or 1	string	o

m: Mandatory tag | o: Optional tag

Mandatory tags included in optional (parent) tags may appear only when the parent is actually used.

Table 1: Tags in a <lineage> object

Example 2 shows a full lineage type. Greyed parts are optional.

```
<lineage>
  <id> INAV-FJI-FOG-QA2000 </id>
  <name> IMU model for navigation grade IMUs of the INAV-FJI-IDEG-001 type </name>
  <author>
    <item> Anna Blanch </item>
    <item> Carles Duch </item>
  </author>
  <organization> AIR CAT </organization>
  <department> NAV_sol </department>
  <date_time> 2009-12-21T16:00:00.234 </date_time>
  <ref_documents>
    <item> Contract 2010.2 </item>
  </ref_documents>
  <project> PLAN-Cerdanya </project>
  <task> WP2100 </task>
</lineage>
```

Example 2: The lineage object

3.5 Data identification: types, identifiers and instance identifiers

In order to identify data an unambiguous naming mechanism is required. ASTROLABE identification principle is based on a categorization criteria. Data is named through its type, specification and instance.

The first two concepts, that is, types and specification, are an essential part of metadata (used, nonetheless, in actual data for cross-referencing purposes). To start the discussion on types and identifiers, example 3 depicts a partial characterization of an ASTROLABE instrument (metadata).

```
<i_spec>
  <type> barometer_t_compensated </type>
  …
  <lineage>
    <id>   bar_tc_model_X </id>
    …
  </lineage>
  …
</i_spec>
```

Example 3: Types and identifiers

In example 3, the aforementioned type and specification identifier are materialized by means of the XML tags <type> and <id> respectively. The last one is part of the <lineage> object described in section 3.4.2.

It is important to clarify what is the difference between these apparently redundant concepts to understand how metadata and data themselves are related.

The **type** is a code tagging a *unique* kind of object (observation / measurement, parameter /state, instrument or model), characterized by some set of properties. An example of such objects could be a certain type of instrument, as for example, a temperature compensated barometer. This kind of barometer would then be assigned a *unique* type.

Should a different kind of barometer exist, as for instance, one whose measurements were not compensated using the value of the temperature, a new type code would be assigned to it.

These type codes could be “barometer_t_compensated” and “barometer_basic” respectively.

On the other side, there may be many barometers that use temperature to correct their measurements. *All these would be instances of the same type of barometer* (“barometer_t_compensated”). However, these may perform very differently, being affected by different levels of noise; or these could deliver data using different units.

In other words, even though all the temperature-compensated barometers may be described *by the same set of properties*, the *values* of these properties may differ. Therefore, a mechanism to tell apart different materializations of the same type of barometer is needed. This mechanism is the *identifier*.

The **identifier** is a code used to tell apart different instances of objects (temperature-compensated barometers in the example above) that, in spite of being characterized by the same set of properties, have different values for these properties.

Then, two temperature-compensated barometers with different performances would belong to the same type of object but would have different identifiers. See example 4.

```
<type>                                  <type>
  barometer_t_compensated                 barometer_t_compensated
</type>                                 </type>
…                                       …
<lineage>                               <lineage>
  <id>                                    <id>
    bar_tc_model_X                          bar_tc_model_YZ
  </id>                                   </id>
…                                       …
```

Example 4: Two instances (identifiers) of the same kind (type) of object

The type tag is used by several entities included in ASTROLABE metadata files (see section 3.6), namely observations and parameters (section 3.6.1.1.1), instruments (section 3.6.1.2.1) and models (section 3.6.1.3.1).

The identifier is materialized by means of the <id> tag included in the <lineage> object (see section 3.4.2).

The third concept listed at the beginning of this section, the instance identifier, is related to data (as opposed to *metadata*). Data include actual values for the fundamental ASTROLABE data entities (as observations or instruments).

Sections 3.6 and 3.7 later in this document describe respectively how metadata is used to characterize actual data. Hereafter a brief overview on the main points of this description regarding identification are described.

As far as metadata is concerned, since there exist a hierarchical relationship between types and identifiers, a single code (the <id> tag) suffices to convey this information (that is, actual data values are fully identified by using the <id> code).

l- and o-records (see sections 3.7.2.3 and 3.7.2.5.2) as well as Example 18), are the fundamental entities using to provide actual data in ASTROLABE. These include the value of the <id> tag identifying the kind of objects they represent as shown below:

```
<l id="some_identifier">       (actual data) </l>
<o id="some_other_identifier"> (actual data) </o>
```

Note that identifiers are specified using the `id` attribute included in the <I> tags of both I- and o-records.

The hierarchical organization of types, identifiers in metadata as well as their use in actual data (I- and o-records) is depicted by Figure 2.

Going still a step further, it must be said that given a type / identifier unique combination, *there may be several instances of objects* that correspond to such description when processing ASTROLABE data.

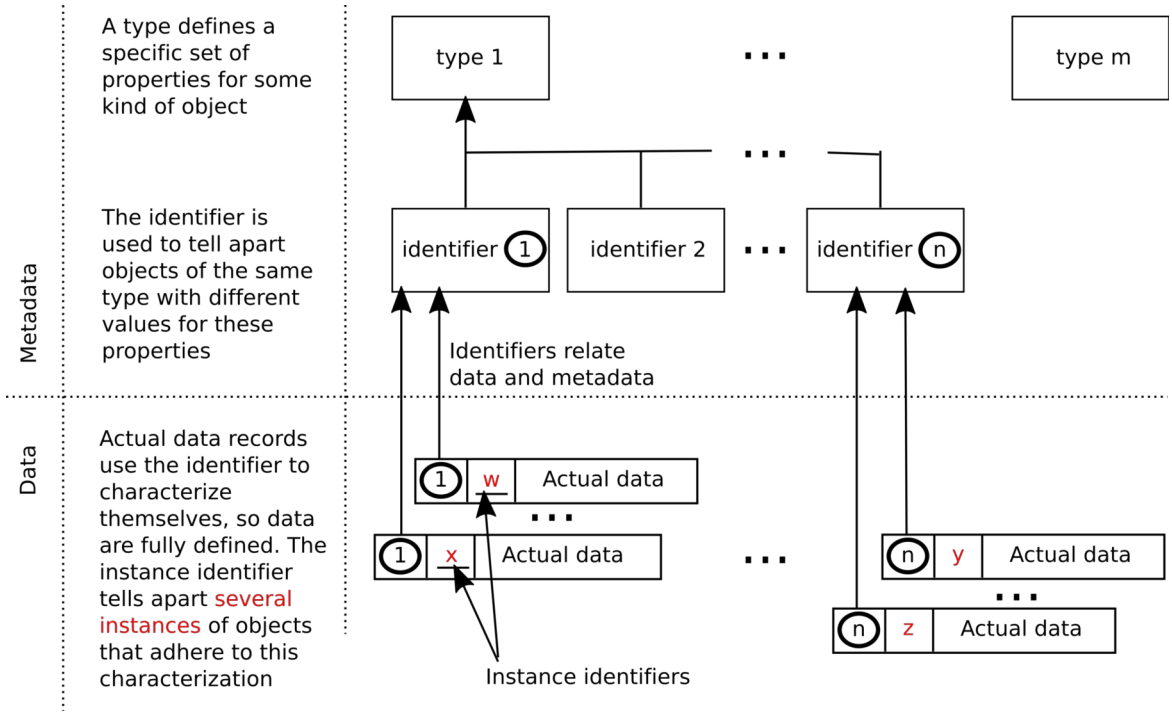

Figure 2: Metadata types and identifiers; data instance identifiers. Cross-referencing

For instance, in example 3, a specific kind of barometer “barometer_tc_model_X” is defined (its type is “barometer_t_compensated”). If two or more of these barometers are used *simultaneously* in a data logging campaign, it will be necessary to be able to tell apart what data correspond to what barometer. That is, even these are equivalent from the metadata standpoint, the information (actual data) they deliver must be clearly distinguishable.

This is the reason why I- and o-records (sections 3.7.2.3 and 3.7.2.5.2) not only must directly reference the aforementioned identifier (the <id> included in metadata) to fully characterize the information these convey, but, additionally, these must include an additional **instance identifier** pointing to the specific object *instance* from which data comes.

Revisiting once more the barometer example, a data acquisition system may be equipped with *two* barometers whose identifier is “barometer_tc_model_X”. From the metadata standpoint, both share the same type / identifier codes, so they are identical. From the data point of view, all I- and o-records will include the identifier “barometer_tc_model_X” to allow their proper characterization. Additionally, these records must incorporate an extra instance identifier (for instance, 1 and 2 respectively) to distinguish which of these two barometers is delivering actual data.

Figure 2 depicts graphically the description given above.

Instance identifiers exist for all the data entities included in the ASTROLABE data model, namely observations (measurements), parameters (states), instruments and models (mapped to “observation equations” - o-records - in data streams).

Instance identifiers are materialized as integer numbers. There are no restrictions about the values these may take (that is, 23 is as good as 9876 to become an instance identifier). Next is an example showing how instance identifiers are materialized in I-records by means of the “n” attribute:

```
<l id="some_identifier" n="223">        (actual data) </l>
```

Nonetheless, there are some restrictions and recommendations that should be taken into account:

The *general recommendation* is to use the same instance identifier for every type / identifier / object instance intervening in an ASTROLABE dataset. Coming back to the barometer example, and assuming again that two identical units are used, each of these should keep the same instance identifier in all data records.

For instance, the first actual barometer would be fully identified by its type ("barometer_t_compensated"), its identifier ("bar_tc_model_X") and its instance identifier, for example 30. The second barometer would be characterized by the same type and identifier than the first one but would be tagged with a different instance identifier, for example, 31.

So, according to the general recommendation, these two units would use, respectively, instance identifiers 30 and 31 respectively in all data records (l- and o-) in the whole data set.

This recommendation must always be respected in the case of parameters (states). That is, this restriction becomes a rule for parameters.

Strictly speaking, the instance identifier for the remaining data entities (observations, instruments and models) must be the kept the same at most within data records that belong to the same *epoch* (see section 3.7.2.1 for a definition). **In this case** (data belonging to the same epoch) **the recommendation also becomes a rule**, being thus mandatory to keep the same instance identifier.

In general, it is highly recommended to keep instance identifiers the same in whole datasets for two reasons:

- Legibility. In the case of data text files (see section 3.7.2.5 for an overview of the different formats available for data files) keeping the same instance identifier across the whole dataset will help to understand their contents.
- Ease and efficiency of implementation. Although this is an implementation-oriented justification that should not affect the formal definition of a data model, using the same instance identifier across whole datasets may help to improve to ease and efficiency of the implementation of the software in charge of processing ASTROLABE data.

3.6 Metadata files

Metadata are of paramount importance. They define completely all the relevant aspects that characterize data, leaving no room to misinterpretation – for instance, assuming the invalid set of physical units for some values.

The metadata files available in the ASTROLABE file interface are the nav-metadata files (<nav_metadata_file> file type).and the navigation directory file (<nav-directory_file> file type).

Strictly speaking, the header files introduced in section 3.4.1 should be considered as metadata, since these describe some of the characteristics of the related data files. Nonetheless, and due to the indissolubility of the tandem header + data files, those are described in the sections related to data files, not metadata ones.

3.6.1 Navigation metadata files: <nav-metadata_file>

These metadata files contain serialized objects with the metadata or specifications for the observations, parameters, instruments and mathematical models. Examples of such metadata for observations range from physical units specifications to coordinate reference frame ones or to default covariance matrices.

In the current version of ASTROLABE, there are four main types of high-level metadata objects: observations, parameters (states), instruments and mathematical models. These are serialized using different *XML tags (*referred to as *serialized objects).*

- Observations: <l_spec>.
- Parameters (states): <p_spec>.
- Instruments: <i_spec>

- Mathematical models: <m_spec>

The metadata serialized objects listed above may be stored in as many files as desired; that is, just a single file may be used to describe all the metadata needed by a software tool using ASTROLABE data or, on the contrary, such information may be spread across several files. The only limitation is that each <nav-metadata_file> must contain at least one serialized object (whatever its type). There are no upper limits on the number of these objects this kind of file may contain.

The recommended file extension of a <nav_metadata_file> type file is [nmd].

Observation and parameter (state) metadata serialized objects are almost identical; therefore, these are described together in section 3.6.1.1. Section 3.6.1.2 takes care of instrument metadata. Section 3.6.1.3 is in charge of describing the mathematical model serialized objects.

3.6.1.1 Observations / state metadata: <l_spec> / <p_spec> serialized objects

The <l_spec> / <p_spec> objects describe, correspondingly, observations and parameters (states). These are almost identical; the only difference between these is the tag used to tell apart the kind of metadata object being described (that is, the <l_spec> or <p_spec> tags themselves). Consequently, these are described together in this section.

Example 5 shows a general <*_spec> object (where * stands for either “l”, or “p”) depicting the general structure of these. Note that the <*_spec> or </*_spec> tags shown in the example are not real ones; these are just markers to show the places where the <l_spec>, </l_spec>, <p_spec> or </p_spec> actual (and real) tags must be located. Note that, in this example, greyed parts are optional.

<*_spec> objects may be either active or removed using the attribute “s” (status) that may be specified in this XML tag. This mechanism allows the apparent removal of a <*_spec> object from later processing without the need to actually delete it from the metadata file. The legal values for the s attribute are [a] (for active <*_spec>) and [r] (to remove these). If the “s” attribute is not present in the <*_spec> tag, the object is considered to be active by default. The following are two examples showing, respectively, an active and removed <*_spec>.

```
<type s=”a”> pseudorange </type>
<type s=”r”> position </type>
```

As seen in the former example, the sub-objects composing a <*_spec> object are the following:

- <type> - (mandatory),
- <toolbox> (optional),
- <lineage> (mandatory),
- <dimension> (mandatory),
- <ref> (mandatory),
- <units> (mandatory),
- <c> (optional) and
- <s> (optional).
- <t_spec>, which is needed only if the observation or parameter (state) object includes *tags* (see section 3.6.1.1.10). Therefore, this sub-object is optional.

The aforementioned sub-objects making a <*_spec> object are described in the next sections.

```
<*_spec s="a">
  <type> baro <type>
  <toolbox> BAROMETER </toolbox>
  <lineage>
    <id>   bar1 </id>
  </lineage>
  <dimension> 1 </dimension>
  <ref>
    <ref_frame_VC>   QNH </ref_frame_VC>
    <coor_system_VC> cartesian </coor_system_VC>
  </ref>
  <units> hPa </units>
  <time_spec>
    <ref>
      <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
    </ref>
    <units> s </units>
  </time_spec>
  <c>     1 </c>
  <s>     1 </s>
  <t_spec>
    <dimension> 1 </dimension>
    <ref>
      <coor_ref_frame_VC> Celsius  </coor_ref_frame_VC>
    </ref>
    <units> °C </units>
  </t_spec>
</*_spec>
```

Example 5: Common structure of the different <l_spec> and <p_spec> objects

3.6.1.1.1 <type>

The <type> sub-object provides, the univocal code identifying the *type* of observations and parameters (states). These codes must match those used by the classes implementing the observation or parameter (state). This tag is mandatory.

The following are examples of these <type> tags.

```
<type> pseudorange </type>
```

```
<type> position </type>
```

See the discussion on section 3.5 about types, identifiers and instance identifiers and how these are used to relate actual data and metadata.

3.6.1.1.2 <lineage>

Refer to section 3.4.2 for a description of this object. <lineage> objects are the same for all the different files included in the file interface. The <lineage> object is mandatory. See again section 3.5 for a discussion about types, identifiers (included in the <id> tag of the <lineage> object) and instance identifiers.

3.6.1.1.3 <toolbox>

The <toolbox> object is optional. When provided, it must contain a *code or identifier* pointing to a software module implementing the logic defining the observation / parameter (state) being described. The way this toolbox code is interpreted – that is, translated to a fully qualified software module path or naming mechanism – is not defined by ASTROLABE.

Typically, the toolbox code or identifier will refer either to a Dynamic Link Library (DLL, Windows environments) or to a Shared Library (Linux environments), although other technologies may be used. Nonetheless, the value for the <toolbox> object should contain only characters valid in file names (as, for instance, upper- and lower case letters, digits and a few special characters; no white space

characters are allowed).

This is the form of the <toolbox> tag:

```
<toolbox> GNSS </toolbox>
```

3.6.1.1.4 <dimension>

The dimension sub-object, which is mandatory, is used to state the number of values in the observations or parameter (state) vector. It must contain a strictly positive (> 0) integer value.

Example:

```
<dimension> 3 </dimension>
```

3.6.1.1.5 <ref>

A <ref> object specifies reference frames and coordinate systems or, equivalently, coordinate reference frames.

If the <ref> object specifies the coordinate reference frame, then it has the form

```
<ref>
 <coor_ref_frame_VC>
   [CRF_VC], [CRF_VC], … , [CRF_VC]
 </coor_ref_frame_VC>
</ref>
```

where [CRF_VC] is a character string representing a CRF valid code. Several codes may be specified by means of a comma separated list. The number of valid codes in the list will depend on the kind of observation, parameter, etc. being described. The following is a simple example where two CRF valid codes are shown.

```
<coor_ref_frame_VC> GPSTime, Celsius </coor_ref_frame_VC>
```

If the <ref> object specifies reference frames and coordinate systems, then it has the form

```
<ref>
 <ref_frame_VC>   [RF_VC], [RF_VC], … , [RF_VC] </ref_frame_VC>
 <coor_system_VC> [CS_VC], [CS_VC], … , [CS_VC] </coor_system_VC>
</ref>
```

where [RF_VC] and [CS_VC] are character strings representing, respectively, reference frame and coordinate system valid codes. Again, several codes may be specified by means of a comma separated list. The number of valid codes in both lists (reference frame, coordinate system) must match, and will depend on the actual observation, parameter, etc. being described.

The following is an example including two couples of reference frame and coordinate system valid codes.

```
<ref>
 <ref_frame_VC>   Bfrd,      WGS84 </ref_frame_VC>
 <coor_system_VC> cartesian, Lned, </coor_system_VC>
</ref>
```

The object <ref_frame_VC> is mandatory and the object <coor_system_VC> object is optional.

The [CRF_VC], [RF_VC] and [CS_VC] codes adhere to exactly the same syntax, which is formally defined in Figure 3.

```
(1) parameters: string |
                string, parameters

(2) base_code:  string |
                string (parameters)

(3) code:       base_code |
                base_code+base_code
```

Figure 3: Grammar defining valid CS, RF and CRF codes

In Figure 3 “string” stands for “series of consecutive characters, including letters, numbers, “-” and “_”.

Definition (1) “parameters”, states that these are either a single string or a list of strings separated by commas. The following are examples of valid parameters

```
par1
```

```
par1, par2, par3, …, parN
```

Definition (2) states that a base_code is either a single string or a string plus parameters enclosed in parethesis:

```
code1
```

```
code1(par1)
```

```
code1(par1,par2,…,parN)
```

Finally, definition (3) states how a valid code (either CS, RF or CRF) is built: it is either a single base_code (see definition 2) or a series of base_code(s) separated by plus sighs (+) (also called *compound code*):

```
code1
```

```
code2+code3(par1,par2)
```

```
code4(par3)+code5(par6,par7,par8)+code6
```

The codes are used to tell apart the different reference frame, coordinate system or coordinate reference frames in use. Actual examples of these codes coud be ED50, UTM, WGS84 or QNH.

Some of these codes are meaningless if not complemented with certain parameters. For instance, the specification of an UTM projection needs a couple of parameters to be complete (the zone and hemisphere). Thus, UTM(31,N) is a valid code, but not just UTM. Other codes may need different sets of parameters to be fully defined.

Compound codes (those including several codes linked with plus signs, see above) are normally used to state that the data being described require more than one coordinate reference frame (or reference frame plus coordinate system) to be correctly defined. Maybe the clearest example are 3D coordinates. In this case, it is usual to provide a 2D code for the planimetric components and one more, this case 1D, for the altimetric component.

That is, assuming that a measurement consisting of *(x, y, z)* values, UTM(33, S) could be the coordinate system for *x* & *y*, while Hn could be the one for the *z* component. Thus, the compound code would be UTM(33,S)+Hn.

The number of elements in a list of CRF or CS + RF codes depends on each instrument, observation or parameter. Extending the former example, and assuming an observation consisting of a position *(x,y,z)*, a temperature *(t)* and a rime *(T)*, the whole observation would look like:

(x,y,z,t,T)

The reference frame for the (x,y,z) component of the observation could be the compound WGS84+UB91, while the ones for the temperature and speed could be Celsius and GPSTime respectively. Thus, the complete list of reference frames would be:

```
<ref_frame_VC> WGS84+UB91, Celsius, GPSTime </ref_frame_VC>
```

In short, the number of CRF, RF or CS codes in their respective lists depends directly on the kind of data being described.

When describing *transformation parameters*, ASTROLABE recommends that the “from”, “target” or “destination” item precedes the “to”, “source” or “origin” one. For instance, the reference frames of a vector parameter $q^b{}_a$ that characterize a rotation matrix $r(q^b{}_a)$ such that brings coordinates x_a referred to a frame a, into coordinates x_b referred to a frame b, shall be written

<ref_frame_VC> b, a </ref_frame_VC>.

Note that, besides de syntax just defined, *ASTROLABE imposes no restrictions* on the values used to specify either coordinate reference frames ([CRF_VC]), reference frames ([RF_VC]) or coordinate systems ([CS_VC]). Developers of software components relying on ASTROLABE data are therefore free to select their own codes.

To facilitate the exchange of ASTROLABE data between software components, it will be necessary to describe precisely the meaning of these codes so, for instance, conversion tools or mapping tables may be built.

Of course, it is possible to use – when available – standardized values for these codes, as those provided by several organizations. Examples of these are the EPSG codes provided by the IOGP (International Association of Oil & Gas Producers), those defined by ESRI or by the IERS (International Earth Rotation and Reference Systems Service). Nonetheless, using standardized codes does not automatically imply the ability to exchange ASTROLABE data between different implementations of the ASTROLABE ICD.

3.6.1.1.6 <units>

This object is mandatory.

Through the <units> object the units of every individual value in the observation or parameter (state) vectors can be specified. It must contain a comma separated list of valid units. The number of elements in this list must match the dimension stated by the <dimension> sub-object (section 3.6.1.1.4).

Units in the aforementioned list must be written according to the syntax defined by section 6 of the UDUNITS-2 C API Guide. At the moment of writing this document, such description may be found at:

http://www.unidata.ucar.edu/software/udunits/udunits-current/doc/udunits/udunits2lib.html#UDUNITS-Library

The following are several examples of unit specifications according to this syntax.

```
<units> m*s-1 </units>
<units> m*s-2, m*s-2, m*s-2 </units>
<units> kg*m*s-2 </units>
```

Dimensionless magnitudes must be written using a single dash (“-”).

3.6.1.1.7 <time_spec>

This object is mandatory.

Observation-event data is related to time (the time when an observation was measured is an integral part of such observation, for instance). Therefore, observations, parameters (states) and instruments must include the <time_spec> object to define time properly.

Two sub-objects are included to do so: <units>, to describe the units used to input time data and <ref>, to define the coordinate system and reference frame or, alternatively, the coordinate reference frame. See section 3.6.1.1.5 for a detailed description of the<ref> sub-object. The <units> tag is described in section 3.6.1.1.6.

Note that a single coordinate reference frame or combination of coordinate system plus reference frame is accepted – since there is only one magnitude to define, the time. Obviously, these codes may be as complex as necessary.

3.6.1.1.8 <c>

The optional <c> object contains the covariance matrix for the observation or parameter (state) vectors. By specifying a <c> object, it is possible to provide default covariance data to observations or parameters (states) when it is not possible to obtain these by any other means; for instance, if some sensor does not deliver quality information about its measurements, then it is possible to assign a default covariance matrix to these (typically, the nominal quality values provided by the manufacturer of the sensor).

Assuming that n is the dimension of the observations / parameter (state) vector stated in the <dimension> sub-object (section 3.6.1.1.4), the dimension of the covariance matrix may be either n (*only* standard deviations) or $(n\cdot(n+1))/2$ (full covariance matrix). Standard deviations come first. If correlations are provided, these must appear in left-to-right, then up-down order.

The units of the covariance matrix are the units specified in the <unit> object of the <l_spec> / <p_spec> object. If the units of the observation / parameter (state) are not specified through the <unit> objects but through the system defaults for the particular observation or parameter (state) types, then the units of the <c> object are the default units as well.

The covariance values are written in the form of a list, separated by white space. Assuming than the following is a full covariance matrix:

$$\begin{pmatrix} 5.4 & 0.8 & -0.2 \\ 0.8 & 1.9 & 0.2 \\ -0.2 & 0.2 & 2.4 \end{pmatrix}$$

then, the next two examples show how to specify a standard deviation only / full covariance matrices respectively:

```
<c> 5.4 1.9 2.4 </c>
<c> 5.4 1.9 2.4 0.8 -0.2 0.2 </c>
```

3.6.1.1.9 <s>

The optional <s> object contains a list of positive scale factors for the standard deviations of the covariance matrices of the observations. The number of elements of the list equals the dimension of the observation / parameter (state) vector – so it matches the dimension stated by the <dimension> sub-object (section 3.6.1.1.4).

The values in the list must be separated by white space, as shown in the example below:

```
<s> 7.0e0 8.0e0 </s>
```

Scale factor values of 1 are assumed when the optional <s> tag is not specified.

3.6.1.1.10 <t_spec>

The optional <t_spec> object may be use to describe *tags*, that is, auxiliary data related to observations or parameter (states).

A tag is not an intrinsic part of the observation or parameter (state) but helps to *complement* it. For instance, a distance measurement may include as a tag the reading delivered by a thermometer at the moment the main observation (distance) was obtained. The reasons why such tags become helpful may be very diverse; for example, the accuracy or precision of the distancemeter used to obtain the aforementioned observation may vary depending on, precisely, the temperature and the availability of this auxiliary tag value may help to correct these deviations.

Observations and parameters (states) may include a list of tags. To specify these, if any, the <t_spec> object must be used.

The sub-objects included in the <t_spec> object are the following:

- <dimension> - This is the number of tag values. This sub-object is mandatory. It is the equivalent of the <dimension> object described in section 3.6.1.1.4, but, in this case, the dimension it refers to is the number of elements in the tags array, not that of the observation or parameter (state) vectors.

- <ref> - The reference frames and coordinate systems or, equivalently, coordinate reference frames of the different tags. See section 3.6.1.1.5 for a complete description of this sub-object. The <ref> sub-object is mandatory.
- <units> - The units for the different tags. The syntax of the <units> tag is described in section 3.6.1.1.6. Note again that the dimension of the list of units must match with the value specified in the <dimension> tag included in the <t_spec> sub-object, not that included in the <*_spec> objects. The <units> object is also mandatory.

As a final comment, the three previous objects (<dimension>, <ref> and <units>) are mandatory if and only if the <t_spec> object is present. If such sub-object is not used, the aforementioned objects may not be specified.

3.6.1.2 Instruments metadata: <i_spec> serialized objects

The <i_spec> object is used to describe instruments. The specific tags that make the instrument sub-object specification different will be explained here.

Example 6 shows how an instrument must be characterized. Note how a new sub-object, namely <c_list>, is included in the metadata defining the instrument. Greyed parts are optional.

An <i_spec> object may be either be activated or deactivated using the “s” (status) attribute provided by this tag. This mechanism allows the apparent removal of a <i_spec> object from later processing without the need to actually delete it from the metadata file. The legal values for the s attribute are [a] (for active <i_spec>) and [r] (to remove these). If the “s” attribute is not present in the <i_spec> tag, the object is considered to be active by default. The following are examples where the active or removed status is explicitly indicated:

```
<i_spec s=”a”>
<i_spec s=”r”>
```

The sub-objects describing an instrument are:

- <type> (mandatory),
- <lineage> (mandatory),
- <toolbox> (optional),
- <time_spec> (mandatory)
- <c_list> (mandatory),
- <t_spec> (optional).

The <lineage>, <toolbox>, <time_spec> and <t_spec> sub-objects in the list above are described, respectively, in sections 3.6.1.1.2, 3.6.1.1.3, 3.6.1.1.7 and 3.6.1.1.10 and will not be repeated here. Obviously, any reference to observations or parameters (states) made in these sections must be understood as referred to their instrument equivalents.

Note how the <dimension>, <ref>, <units>, <c> and <s> sub-objects have been moved to the new <c_list> sub-object, which characterizes the different *instrument constants*.

```
<i_spec s= ”a”>
  <type> baro_p0_h0 </type>
  <lineage>
    <id>   delta_p0_h0  </id>
    <name> Initial values of the test </name>
    <date_time> 2007-01-21T16:00:00.234 </date_time>
  </lineage>
  <toolbox> BAROMETER </toolbox>
  <time_spec>
    <ref>
      <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
    </ref>
    <units> s </units>
  </time_spec>
  <c_list>
    <dimension> 2 </dimension>
    <item n="1">
      <type> scalar </type>
      <ref>
        <ref_frame_VC>   QNH </ref_frame_VC>
        <coor_system_VC> pressure </coor_system_VC>
      </ref>
      <units> mBar </units>
      <c> 1.2 </c>
      <s> 1 </s>
    </item>
    <item n="2">
      <type> scalar </type>
      <ref>
        <ref_coordinate_frame_VC>
          WGS84-ellipsoidal-height
        </ref_coordinate_frame_VC>
      </ref>
      <units> m </units>
      <c> 0.15 </c>
      <s> 1 </s>
    </item>
  </c_list>
  <t_spec>
    <dimension> 2 </dimension>
    <ref>
      <ref_frame_VC>   QNH, WGS84 </ref_frame_VC>
      <coor_system_VC> pressure, cartesian </coor_system_VC>
    </ref>
    <units> mBar, m </units>
  </t_spec>
</i_spec>
```

Example 6: The <i_spec> (instrument metadata) object

3.6.1.2.1 <type>

The <type> object provides the univocal code identifying the unique kind of instrument. This code must match the one used by the class implementing the instrument.

The following is an examples of this <type> tag.

```
<type> antenna_PC </type>
```

See the discussion on section 3.5 about types, identifiers and instance identifiers and how these are used to relate actual data and metadata.

3.6.1.2.2 <lineage>, <toolbox>, <time_spec> and <t_spec>

Please, refer to sections 3.6.1.1.2, 3.6.1.1.3, 3.6.1.1.7 and 3.6.1.1.10 for a detailed description of these

sub-objects. All references to observations or parameter (states) made in these sections must be interpreted as references to instruments.

Note that the use of tag data may seem strange in the context of instruments, since these are considered as pre-computed constants and therefore there is no need to determine any of its values. Nonetheless, tags are still provided in the <i_spec> object to complement instrument data. For instance, an instrument constant may be used to specify the adjusted focal length of a camera; a tag, on the contrary, may be useful to provide with its nominal (as provided by the manufacturer) value, just in case knowing this information helps the software using ASTROLABE data to better determine a trajectory.

It is not possible to foresee all the situations where tags for instruments may be useful. Therefore, these are provided to leave an open door to introduce any kind of data that might help to better characterize instruments.

3.6.1.2.3 <c_list>

The <c_list> sub-object is used to describe the different constants that characterize an instrument. The tags included in the <c_list> sub-object are:

- <dimension> (mandatory). This tag is used to state the number of constants characterizing the instrument.
- <item> (mandatory). There must be as many <item> entries as stated by the <dimension> tag above. Each of these describe one of the instrument constants.

The <item> include, on its turn, several sub-tags to define all the necessary facets of an instrument constant. These are:

- <type> (mandatory), Instrument constants may be either scalar values or multidimensional matrices. With this tag it is possible to select one of these two types.

  Scalar constants are identified by the value [SCALAR] in the <c_item_type> tag Multidimensional matrix instrument constants use the syntax [MATRIX](size of dimension1, size of dimension 2, … , size of dimension n) to be defined.

  The following is a simple example showing how to specify a scalar instrument constant:

  ```
  <type> scalar </type>
  ```

  while the next one describes another that is represented by a 2 x 3 matrix:

  ```
  <type> matrix(2,3) </type>
  ```

  In the case of multi-dimensional matrix instrument constants, there is no upper limit to the number of dimensions. The minimum number of dimensions, however, is one, as shown in the next example:

  ```
  <type> matrix(4) </type>
  ```

- <ref> (mandatory). Define the reference frame and coordinate system or, equivalently, coordinate reference frame, for the instrument constant. See section 3.6.1.1.5 for details on the syntax of the <ref> object.
- <units> (mandatory), The units for the instrument constant. See section 3.6.1.1.6 for a detailed description on how to specify units. Only one unit may be specified for each <item>, no matter whether it is a scalar or multidimensional matrix. This should be evident in the case of scalar instrument constants; when defining matrices, all the elements in these must be written using the same units, so only one unit item is required.
- <c> (optional), The covariance matrix for the instrument constant. See section 3.6.1.1.8 for a detailed description on how to specify covariance matrices. Note that scalar values only accept a single standard deviation (no correlations available). Multidimensional matrices accept both forms of covariance matrices, that is, those including standard deviations only or full ones (including correlations).

  Note that instrument constants are not random variables, that is, are not affected by time.

Therefore, covariance matrices, when present, must be interpreted *as a mere indication of the quality of the instrument constant.*

The units of the standard deviations of the covariance matrix will always be the same as those for the constant value(s) itself, either specified in the <units> tag above (when present) or the default ones (when absent).

- <s> (optional). List of positive scale factors for the standard deviations in the covariance matrices of the instrument constant (see section 3.6.1.1.9). Its dimension must match the number of standard deviations for the instrument constant. When absent, default scale factors of 1.0 for all standard deviations are assumed.

3.6.1.3 Models metadata: <m_spec> serialized objects

The <m_spec> object is used to characterize models.

As it happens with the <l_spec>, <p_spec> and <i_spec> objects, <m_spec> objects may be either active or removed using the attribute “s” (status) that may be specified in the <m_spec> XML tag. Again, this mechanism allows the apparent removal of a <m_spec> object from later processing without the need to actually delete it from the metadata file. The legal values for the s attribute are [a] (for active <m_spec>) and [r] (to remove these). If the “s” attribute is not present in the <m_spec> tag, the object is considered to be active by default.

The <m_spec> object characterizes a model. It is composed of the following sub-objects:

- <type> (mandatory),
- <lineage> (mandatory),
- <toolbox> (optional),
- <dynamic> (mandatory)
- <l_list> (mandatory),
- <p_list> (mandatory),
- <i_list> (optional) and,
- <sub-m_list> (optional).

Example 7 depicts an <m_spec> object that includes all the possible sub-objects listed above. Greyed parts are optional. Details on the syntax of every of these sub-objects are given in the next subsections.

3.6.1.3.1 <type>

The <type> object provides the univocal code identifying the model. This code must match the one used by the class implementing the model.

The following is an examples of this <type> tag.

```
<type> GALILEO-D-simple </type>
```

See the discussion on section 3.5 about types, identifiers and instance identifiers and how these are used to relate actual data and metadata.

3.6.1.3.2 <lineage>

Refer to section 3.4.2 for a description of this object. <lineage> objects are the same for all the different files included in the file interface. The <lineage> object is mandatory. See again section 3.5 for a discussion about types, identifiers (included in the <id> tag of the <lineage> object) and instance identifiers.

```
<m_spec s="a">
  <type> local_mec_eq_bias_d </type>
  <lineage>
    <id>   pva1_d  </id>
    <name> Mechanization equations </name>
    <date_time> 2007-01-21T16:00:00.234 </date_time>
  </lineage>
  <toolbox> INS </toolbox>
  <dynamic> YES </dynamic>
  <l_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> imu1 </id>
    </item>
  </l_list>
  <p_list>
    <dimension> 2 </dimension>
    <item n="1">
      <id> pva1 </id>
      <role> free </role>
    </item>
    <item n="2">
      <id> imu1_bias </id>
      <role> free </role>
    </item>
  </p_list>
  <i_list>
    <dimension> 1 </dimension>
    <item n=”1”>
      <id> p0_h0 </id>
    </item>
  </i_list>
  <sub-m_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> WGS84 </id>
    </item>
  </sub-m_list>
</m_spec>
```

Example 7: The <m_spec> object

3.6.1.3.3 <toolbox>

The <toolbox> object is optional. When provided, it must contain a *code or identifier* pointing somehow to a software module implementing the logic defining the model being described. The way this toolbox code is interpreted – that is, translated to a fully qualified software module path or naming mechanism – is not defined by ASTROLABE.

Typically, the toolbox code or identifier will refer either to a Dynamic Link Library (DLL, Windows environments) or to a Shared Library (Linux environments), although other technologies may be used. Nonetheless, the value for the <toolbox> object should contain only characters valid in file names (as, for instance, upper- and lower case letters, digits and a few special characters; no white space characters are allowed).

This is the form of the <toolbox> tag:

```
<toolbox> GNSS </toolbox>
```

3.6.1.3.4 <dynamic>

This keyword is used to specify whether the model is time-dependent (or dynamic) since it uses differential equations to do its work (value: [YES]) or not (non time-dependent, not dynamic, no use of differential equations, value [NO]). The <dynamic> object is mandatory. The following are examples of (1) a time-dependent or dynamic model and (2) a non time-dependent (non dynamic) one.

```
<dynamic> YES </dynamic>
```

```
<dynamic> NO </dynamic>
```

3.6.1.3.5 <l_list>

By means of this tag, a list of the identifiers of the observations related to the model may be specified. These identifiers are those specified in the <id> tag of the mandatory <lineage> object included in the metadata (<l_spec>) describing each of the observations involved in the list. See section 3.5 for a discussion about types, identifiers and instance identifiers.

Each identifier is specified by means of an <item> object:

```
<l_list>
   <dimension> m </dimension>
   <item n="1">
     <id> pseudo_galileo </id>
   </item>

   <item n="2">
     <id> clock_correction </id>
   </item>
       …
  </l_list>
```

The <dimension> tag in the example above states the number of <item> sub-objects to follow. Its value (shown as *m* in the aforementioned example) must be a positive integer.

There must be as many <item> objects in the <l_list> object as identifiers to include and its number must match the value of the <dimension> tag defined above. There must be at least one <item> object. Each <item> must be numbered by means of its attribute "n". The <item>s objects need *not* to be sorted according to the n attribute. However, if m <item>s are included, the values for n must be all different and ranging from 1 to m. The actual number of <item>s in a <l_list> object depends on the model being described.

The <id> tag within the <item> object is used to provide the observation identifier, which must correspond to the value of the <id> tag of the <lineage> object of an existing <l_spec> object.

The <l_list> object is mandatory.

3.6.1.3.6 <p_list>

The list of identifiers of the parameters (states) related to the model is specified by means of the <p_list> object. These identifiers are those specified in the <id> tag of the mandatory <lineage> object included in the metadata (<p_spec>) describing each of the parameters involved in the list. See section 3.5 for a discussion about types, identifiers and instance identifiers.

Each identifier is given by means of an <item> object.

```
<p_list>
  <dimension> m </dimension>
  <item n="1">
    <id> pos_wgs84 </id>
    <role> free </role>
  </item>

  <item n="2">
    <id> dt_gal </id>
    <role> free </role>
```

```
  </item>
       …
 </p_list>
```

The <dimension> tag in the example above states the number of <item> sub-objects to follow. Its value (shown as *m* in the aforementioned example) must be a positive integer.

There must be as many <item> objects in the <p_list> object as identifiers to include and its number must match the value of the <dimension> tag defined above. There must be at least one <item> object. Each <item> must be numbered by means of its attribute “n”. The <item>s objects need *not* to be sorted according to the n attribute. However, if m <item>s are included, the values for n must be all different and ranging from 1 to m. The actual number of <item>s in a <p_list> object depends on the model being described.

A <item> object includes two tags:

- <id>: used to provide the identifier of the parameter (state), which must correspond to the value of the <id> tag of the <lineage> object included in an existing <p_spec> object, and
- <role>: may take two values: [FREE] or [CONSTANT]. [FREE] is used to specify that the parameter must be computed when generating the trajectory; [CONSTANT] is used on the opposite situation, that the value of the parameter will not be modified by such computation.

Note that a parameter / state may play different roles (that is, [FREE] or [CONSTANT]) when included (via <p_list> and <item>) in more than one model.

The <p_list> object is mandatory.

3.6.1.3.7 <i_list>

There may be no <i_list> objects included in a <m_spec> tag, that is, <i_list> is optional.

To specify the list of identifiers of the different instruments related to the model, the <i_list> object must be used. These identifiers are those specified in the <id> tag of the mandatory <lineage> object included in the metadata (<i_spec>) describing each of the instruments involved in the list. See section 3.5 for a discussion about types, identifiers and instance identifiers.

Each identifier is given by means of an <item> object.

```
<i_list>
  <dimension> m </dimension>
  <item n=”1”>
    <id> antenna_rover_pc </id>
  </item>

  <item n=”2”>
    <id> antenna_master_pc </id>
  </item>
       …
</i_list>
```

The <dimension> tag in the example above states the number of <item> sub-objects to follow. Its value (shown as *m* in the aforementioned example) must be a positive integer.

There must be as many <item> objects in the <i_list> object as identifiers to include and its number must match the value of the <dimension> tag defined above. There must be at least one <item> object. Each <item> must be numbered by means of its attribute “n”. The <item>s objects need *not* to be sorted according to the n attribute. However, if m <item>s are included, the values for n must be all different and ranging from 1 to m. The actual number of <item>s in a <i_list> object depends on the model being described.

As stated above, the <i_list> object is optional.

3.6.1.3.8 <sub-m_list>

This optional object is a list including the identifiers of external generators of auxiliary values. These generators are provided by developers according to their needs and must be included in the same DLL or shared library where the model is implemented. The identifiers in this list are local to this DLL / shared library. This is a way to facilitate the computation of any kind of value that might be needed by the model.

For instance, a model implementing an IMU might need to compute the values related to the gravity field. Several mechanism may be available to do so; by means of this list of external generators, any of the available methods to compute gravity data may be selected. These generators and their identifiers must be handled by the DLL or Shared library where the IMU model is implemented.

<sub-m_list> is used to provide the identifiers of the external generators needed by the model. These are defined by means of a number of <item> sub-objects..

```
<sub-m_list>
  <dimension> m </dimension>
  <item n=”1”>
    <id> EGNOS_tropo_model </id>
  </item>

  <item n=”2”>
    <id> EGNOS_tropo_model </id>
  </item>
  …
</sub-m_list>
```

The <dimension> tag in the example above states the number of <item> sub-objects to follow. Its value (shown as *m* in the aforementioned example) must be a positive integer.

There must be as many <item> objects in the <sub-m_list> object as identifiers to include and its number must match the value of the <dimension> tag defined above. There must be at least one <item> object. Each <item> must be numbered by means of its attribute “n”. The <item>s objects need *not* to be sorted according to the n attribute. However, if m <item>s are included, the values for n must be all different and ranging from 1 to m. The actual number of <item>s in a <sub-m_list> object depends on the model being described.

The <id> tag within the <item> object is used to provide the external generator identifier.

The <sub-m_list> object is optional.

3.6.1.4 Example

Example 8 depicts a complete navigation metadata file. Note the opening tag for these files (<nav_metadata_file>) and the name of the validating XML schema (“nav_metadata.xsd”). It includes an instance of all the sub-objects that may be found in such files (<l_spec>, <p_spec>, <i_spec> and <m_spec>). Except for some sub-objects of the <lineage> object, all optional fields are shown.

The navigation metadata file in example 8 below could have been split, for instance, into several files, each of these containing at least one of the first level sub-objects (see section 3.6.1).

As stated above, greyed areas in example 8 are optional. The line numbers located on the left of the example are not part of the navigation metadata file. These are included for referencing the example. The colouring for these line numbers helps to identify full high-level objects (as <l_spec> or <m_spec>).

In example 8:

- The metadata for three observations, three parameters, one instrument and three models are defined.
- Observations are defined in lines 4 to 79. Parameters are specified in lines 80 to 148. The unique instrument present in this example is characterized in lines 149 to 187. The three models defined in this file are characterized in lines 188 to 276.

- The first observation (lines 4 to 33) represents data obtained from a barometer. Its type is “baro” (line 5). This particular instance of a barometer is named “baro1” (line 7). The number of values in the observation vector is 1 (line 12). The reference frame and coordinate system are “QNH” and “cartesian” respectively (lines 14 and 15). The units for the observation vector (just one) are hPa (line 17). The reference frame and units for time are defined in lines 18 to 23. More specifically, the reference frame code is input in line 20 while time units are defined in line 22. The default covariance matrix is 1 (only standard deviations, correlations do not exist in this case) and is defined in line 24. The scale factor for the covariance matrix's standard deviations is 1 (line 25). The observations obtained from the barometer may be complemented by means of a single tag (lines 26 to 32): temperature (barometers may be affected by changes in temperature). There is only one tag (line 27), its coordinate reference frame code is “Celsius” (line 29) and its units degrees Celsius (line 31). The implementation of this kind of observation is stored in the toolbox named “BAROMETER” (line 11).
- The two remaining observations (lines 34 to 79) represent, respectively, those provided by an inertial measuring unit (IMU) and their biasses. Note how the dimension of the observation vector for these two observations is 6 (lines 42 and 65).
- The description of the first parameter (lines 80 – 106) states that the toolbox where such parameter is implemented is “INS” (line 87), which differs from that used for parameter 1 (that is, toolboxes may be used simultaneously by several parameters or observations...). It represents a position, velocity and attitude in the WGS84 reference frame. Therefore, the parameter vector is made of 9 components (line 88). The reference frame code for these 9 components are provided in line 90). On the contrary, Coordinate system codes are input in lines 93-97.
- The unique instrument defined in this navigation metadata file may be found in lines 149 to 187. It is a barometer. Its type is “baro_p0_h0” (line 150) and this particular instance has been tagged with the identifier “p0_h0” (line 152). A set of two constants (line 164) are used to characterize the instrument. Both are scalar values (lines 166 and 176) but use different reference frames / coordinate systems or reference coordinate frames (lines 168-169 and 179). The covariance matrices (only standard deviations) for these two constants are provided in lines 172 and 183 respectively. Note that covariance matrices play only an informative role when characterizing instrument constants.
- The example also includes three models (lines 188 – 276). The first one (lines 188 – 220) is the most comprehensive one. It is a time-dependent (dynamic) model – see line 196. It involves only one observation (lines 197 – 202) whose identifier (not type) is “imu1” (defined in lines 34 – 56, <id> stated in line 37). Two parameters are needed by this model (lines 203 – 213) and their identifiers are, respectively, “pva1” and “imu1_bias” (see lines 83 and 110). The model makes use of no instruments, but states that an additional sub-model (“WGS84”) is needed (lines 214 – 219). Note that this sub-model is not a model in the sense defined in section 3.6.1.3 but an external generator of auxiliary values needed by the model itself. Therefore, “WGS84” is not defined in the navigation metadata file but just referenced.
- The last model (lines 244 – 276) includes an instrument using the <i_list> object (lines 270 – 275) whose identifier is “p0_h0” (the barometer).

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <nav_metadata_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
003  xsi:noNamespaceSchemaLocation="nav_metadata.xsd">
004   <l_spec s="a">
005     <type> baro </type>
006     <lineage>
007       <id>   baro1 </id>
008       <name> Barometer measurements </name>
009       <date_time> 2015-06-09T16:00:00.234 </date_time>
010     </lineage>
011     <toolbox> BAROMETER </toolbox>
012     <dimension> 1 </dimension>
013     <ref>
014       <ref_frame_VC>   QNH </ref_frame_VC>
015       <coor_system_VC> cartesian </coor_system_VC>
016     </ref>
017     <units> hPa </units>
018     <time_spec>
019       <ref>
020         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
021       </ref>
022       <units> s </units>
023     </time_spec>
024     <c>     1 </c>
025     <s>     1 </s>
026     <t_spec>
027       <dimension> 1 </dimension>
028       <ref>
029         <coor_ref_frame_VC> Celsius </coor_ref_frame_VC>
030       </ref>
031       <units> ° </units>
032     </t_spec>
033   </l_spec>
034   <l_spec s="a">
035     <type> imu </type>
036     <lineage>
037       <id>   imu1 </id>
038       <name> IMU measurements </name>
039       <date_time> 2015-06-09T16:00:00.234 </date_time>
040     </lineage>
041      <toolbox> INS </toolbox>
042     <dimension> 6 </dimension>
043     <ref>
044       <ref_frame_VC>    Bfrd </ref_frame_VC>
045       <coor_system_VC>  cartesian </coor_system_VC>
046     </ref>
047     <units> rad/s , rad/s , rad/s , m/s^2, m/s^2, m/s^2 </units>
048     <time_spec>
049       <ref>
050         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
051       </ref>
052       <units> s </units>
053     </time_spec>
054     <c>     5e-2 5e-2 5e-2 1e-1 1e-1 1e-1 </c>
055     <s>     1 </s>
056   </l_spec>
057   <l_spec s="a">
058     <type> RW_6_PN </type>
059     <lineage>
060       <id>   imu1_bias_pn </id>
061       <name> IMU measurements </name>
062       <date_time> 2015-06-09T16:00:00.234 </date_time>
063     </lineage>
064     <toolbox> INS </toolbox>
```

```
065     <dimension> 6 </dimension>
066     <ref>
067       <ref_frame_VC>    Bfrd </ref_frame_VC>
068       <coor_system_VC>  cartesian </coor_system_VC>
069     </ref>
070     <time_spec>
071       <ref>
072         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
073       </ref>
074       <units> s </units>
075     </time_spec>
076     <units> rad/s , rad/s , rad/s , m/s^2, m/s^2, m/s^2 </units>
077     <c>     5e-2 5e-2 5e-2 1e-1 1e-1 1e-1 </c>
078     <s>     1 </s>
079   </l_spec>
080   <p_spec s="a">
081     <type>  pva </type>
082     <lineage>
083       <id>   pva1  </id>
084       <name> Position, velocity and attitude in WGS84 </name>
085       <date_time> 2007-01-21T16:00:00.234 </date_time>
086     </lineage>
087     <toolbox> INS </toolbox>
088     <dimension> 9 </dimension>
089     <ref>
090      <ref_frame_VC>
091        WGS84, WGS84, WGS84, WGS84,
092        WGS84, WGS84, WGS84, WGS84, WGS84 </ref_frame_VC>
093      <coor_system_VC>
094        geodetic, geodetic, geodetic,
095        Lned, Lned, Lned,
096        Bfrd-Lned, Bfrd-Lned, Bfrd-Lned
097      </coor_system_VC>
098     </ref>
099     <units> rad, rad, m, m/s, m/s, m/s, rad, rad, rad </units>
100     <time_spec>
101       <ref>
102         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
103       </ref>
104       <units> s </units>
105     </time_spec>
106   </p_spec>
107   <p_spec s="a">
108     <type>  IMU_bias </type>
109     <lineage>
110       <id>   imu1_bias  </id>
111       <name> Calibration biases of IMU </name>
112       <date_time> 2007-01-21T16:00:00.234 </date_time>
113     </lineage>
114     <toolbox> INS </toolbox>
115     <dimension> 6 </dimension>
116     <ref>
117      <ref_frame_VC>    Bfrd </ref_frame_VC>
118      <coor_system_VC>  cartesian </coor_system_VC>
119     </ref>
120     <units> rad/s, rad/s, rad/s, m/s^2, m/s^2, m/s^2 </units>
121     <time_spec>
122       <ref>
123         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
124       </ref>
125       <units> s </units>
126     </time_spec>
127   </p_spec>
128   <p_spec s="a">
129     <type>  baro_cal </type>
```

```
130     <lineage>
131       <id>   baro1_cal  </id>
132       <name> Calibration values of the barometer </name>
133       <date_time> 2007-01-21T16:00:00.234 </date_time>
134     </lineage>
135     <toolbox> BAROMETER </toolbox>
136     <dimension> 1 </dimension>
137     <ref>
138       <ref_frame_VC>   QNH </ref_frame_VC>
139       <coor_system_VC> cartesian </coor_system_VC>
140     </ref>
141     <units> hPa </units>
142     <time_spec>
143       <ref>
144         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
145       </ref>
146       <units> s </units>
147     </time_spec>
148   </p_spec>
149   <i_spec s="a">
150     <type> baro_p0_h0 </type>
151     <lineage>
152       <id>   p0_h0  </id>
153       <name> Initial values of the test </name>
154       <date_time> 2007-01-21T16:00:00.234 </date_time>
155     </lineage>
156     <toolbox> BAROMETER </toolbox>
157     <time_spec>
158       <ref>
159         <coor_ref_frame_VC> GPStime </coor_ref_frame_VC>
160       </ref>
161       <units> s </units>
162     </time_spec>
163     <c_list>
164       <dimension> 2 </dimension>
165       <item n="1">
166         <type> scalar </type>
167         <ref>
168           <ref_frame_VC>   QNH </ref_frame_VC>
169           <coor_system_VC> pressure </coor_system_VC>
170         </ref>
171         <units> mBar </units>
172         <c> 1.2 </c>
173         <s> 1 </s>
174       </item>
175       <item n="2">
176         <type> scalar </type>
177              <ref>
178           <coor_ref_frame_VC>
179             WGS84-ellipsoidal-height
180           </coor_ref_frame_VC>
181               </ref>
182         <units> m </units>
183         <c> 0.15 </c>
184         <s> 1 </s>
185       </item>
186     </c_list>
187   </i_spec>
188   <m_spec s="a">
189     <type> local_mec_eq_bias_d </type>
190     <lineage>
191       <id>   pva1_d  </id>
192       <name> Mechanization equations </name>
193       <date_time> 2007-01-21T16:00:00.234 </date_time>
194     </lineage>
```

```
195      <toolbox> INS </toolbox>
196      <dynamic> YES </dynamic>
197      <l_list>
198        <dimension> 1 </dimension>
199        <item n="1">
200          <id> imu1 </id>
201        </item>
202      </l_list>
203      <p_list>
204        <dimension> 2 </dimension>
205        <item n="1">
206          <id> pva1 </id>
207          <role> free </role>
208        </item>
209        <item n="2">
210          <id> imu1_bias </id>
211          <role> free </role>
212        </item>
213      </p_list>
214      <sub-m_list>
215        <dimension> 1 </dimension>
216        <item n="1">
217          <id> WGS84 </id>
218        </item>
219      </sub-m_list>
220    </m_spec>
221     <m_spec s="a">
222      <type> RW_6_d </type>
223      <lineage>
224        <id>   imu1_bias_d  </id>
225        <name> IMU biases as random walk </name>
226        <date_time> 2007-01-21T16:00:00.234 </date_time>
227      </lineage>
228      <toolbox> INS </toolbox>
229      <dynamic> YES </dynamic>
230      <l_list>
231        <dimension> 1 </dimension>
232        <item n="1">
233          <id> imu1_bias_pn </id>
234        </item>
235      </l_list>
236      <p_list>
237        <dimension> 1 </dimension>
238        <item n="1">
239          <id> imu1_bias </id>
240          <role> free </role>
241        </item>
242      </p_list>
243    </m_spec>
244    <m_spec s="a">
245      <type> PVA_baro_U </type>
246      <lineage>
247        <id> height_update  </id>
248        <name> Update of height </name>
249        <date_time> 2007-01-21T16:00:00.234 </date_time>
250      </lineage>
251      <toolbox> BAROMETER </toolbox>
252      <dynamic> NO </dynamic>
253      <l_list>
254        <dimension> 1 </dimension>
255        <item n="1">
256          <id> baro1 </id>
257        </item>
258      </l_list>
259      <p_list>
```

```
260       <dimension> 2 </dimension>
261       <item n="1">
262         <id> pva1 </id>
263         <role> free </role>
264       </item>
265       <item n="2">
266         <id> baro1_cal </id>
267         <role> free </role>
268       </item>
269    </p_list>
270    <i_list>
271      <dimension> 1 </dimension>
272      <item n="1">
273        <id> p0_h0 </id>
274      </item>
275    </i_list>
276   </m_spec>
277 </nav_metadata_file>
```

Example 8: A complete navigation metadata file

3.6.2 The navigation directory metadata file: <nav-directory_file>

The <nav-directory_file> lists all the necessary input, output and internal working files needed to describe the estimated trajectory. It may be seen as a directory listing all the files involved in the process.

<nav-directory_file> file names must strictly adhere to a **fixed** naming convention. See below.

Depending on the processing mode, there will be ***either*** one ***or*** three files, whose names are defined in Table 2.

The reason to use exactly the names shown in Table 2 and not to allow arbitrary file names is that it is necessary to identify which of the file(s) stored in a navigation file (see section 3.7.11) are the ones defining the trajectory(ies). Arbitrary names would make this task impossible.

Processing mode	# of files	File name(s)
Forward	1	`nav-directory_fw.ndf`
Backward	1	`nav-directory_bw.ndf`
Combined (block adjustment)	1	`nav-directory_cb.ndf`

Combined (forward, backward & smoothing)	3	`nav-directory_fw.ndf`
		`nav-directory_bw.ndf`
		`nav-directory_cb.ndf`

Table 2: Processing modes and <nav_directory_file> files

The naming policy of this ASTROLABE FI only *recommends* values for the extensions of the different files that it is composed of. This translates, in practice, to an absolute free file naming schema. In such a context, not setting a predefined value for the names of the <nav-directory_file> files leads to a situation where it is impossible to determine what are these. The obvious solution is to set standardized name(s) for <nav-directory_file> files when stored in a navigation <nav_file> object.

The navigation directory file is stored as a text, XML file. Table 3 depicts its contents schematically.

A <nav-directory_file> file contains a list of all the files, either input or output, involved in the estimation of a trajectory. Therefore, all the leaf XML tags contain file names. Table 4 shows how tags in a <nav-directory_file> map to actual files.

Level	XML tag	#	Type	m/o
1	<lineage>	1	See section 3.4.2	m
1	<input>	1		m
2	<options>	0 or 1		o
3	<op_file>	1	string	m
2	<metadata>	1		m
3	<nav_metadata>	>= 1	string	m
2	<observations>	1		m
3	<obs_e>	1	string	m
3	<tr_obs_correlation_Rll>	0 or 1	string	o
2	<instruments>	0 or 1		o
3	<obs_e>	1	string	m
1	<output>	1		m
2	<log>	0 or 1	string	o
2	<states>	1		m
3	<obs_e>	1	string	m
3	<tr_states_correlation_Rxx>	0 or 1	string	o
2	<residuals>	0 or 1		o
3	<obs_e>	1	string	m
3	<tr_res_correlation_Rvv>	0 or 1	string	o

m: Mandatory tag | o: Optional tag

Mandatory tags included in optional (parent) tags may appear only when the parent is actually used.

Table 3: Tag structure of a navigation directory file

Example 9 shows a full navigation directory file. Optional parts are highlighted in light grey. Note the specific opening tag <nav_directory_file> and the validating XML schema (nav-directory.xsd).

3.7 Data files and formats

ASTROLABE includes several types of data files. These are:

- Input files:
  - Observations (measurements).
  - Instruments.
  - Correlation matrices for input observations.
  - Process options (optional).

XML tag	File / description
<op_file>	Options file. Details about option files may be found in section 3.7.9.
<nav_metadata>	Observation / state / instrument / model metadata file(s). See section 3.6 for details.
<observations> <obs_e>	File with input observations and observation equations. See sections 3.7.2 and 3.7.4 for a detailed description of the format of obs-e files.
<observations> <tr_obs_correlation_Rll>	File with the correlation matrices for the input observation-events file. The format of these files are described in sections 3.7.3 and 3.7.8.
<instruments> <obs_e>	File with the constants characterizing the instruments. See sections 3.7.2 and 3.7.5.
<log>	The progress report showing how the computation of the output trajectory evolved. See section 3.7.10.
<states> <obs_e>	In this file, the estimated output parameters (states) computed by the system are stored. This is the output trajectory. This is an obs-e file, whose format is described in sections 3.7.2 and 3.7.6.
<states> <tr_states_correlation_Rxx>	This file contains the correlation matrices for the output parameter (states) file. See sections 3.7.3 and 3.7.8 for a description of its format.
<residuals> <obs_e>	The estimated residuals for the input observations are saved to this file. obs-e files are described in section 3.7.2. See also section 3.7.7.
<residuals> <tr_res_correlation_Rvv>	This file is used to store the correlation matrices of the estimated residuals for the input observations. See sections 3.7.3 and 3.7.8.

Table 4: Navigation directory file: XML tags and files

- Output files:
  - Parameter (states).
  - Residuals (observations)
  - Correlation matrices for (1) output parameters and (2) observation residuals.
  - Trajectory estimation process log (optional).
  - Navigation directory file.

Finally, all data files (or streams sent via socket connections), either input or output, are accompanied by a header file characterizing their contents

The input observations and instruments files, as well as the output residuals for observations and parameters (states) files *adhere to the same file format*, the observation-events (<obs-e_file>) format. The three correlation matrix files adhere to the *correlation matrix file* (<r-matrix_file>) *format*.

```
<?xml version="1.0" encoding="UTF-8"?>
<nav-directory_file
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="nav-directory_file.xsd">

  <lineage>
    <id> my-trajectory_bw </id>
  </lineage>

  <input>

    <options>
      <op_file format="NAVEGA" version="1.0"> INS_GNSS_CC.op </op_file>
    </options>

    <metadata>
      < nav_metadata> INS_GNSS_CC_01.nmd </nav_metadata>
      < nav_metadata> INS_GNSS_CC_02.nmd </nav_metadata>
    </metadata>

    <observations>
      <obs_e>                    INS_GNSS_CC.obs </obs_e>
      <tr_obs_correlation_Rll>  INS_GNSS_CC.rll </tr_obs_correlation_Rll>
    </observations>

    <instruments>
      <obs_e>                    INS_GNSS.ins </obs_e>
    </instruments>

  </input>

  <output>

    <log> INS_GNSS_CC_bw.log </log>

    <states>
      <obs_e> output_bw.par </obs_e>
      <tr_states_correlation_Rxx> INS_GNSS_CC_bw.rxx </tr_states_correlation_Rxx>
    </states>

    <residuals>
      <obs_e> INS_GNSS_CC_residuals_bw.res </obs_e>
      <tr_res_correlation_Rvv> INS_GNSS_CC_res_bw.rvv </tr_res_correlation_Rvv>
    </residuals>

  </output>

</nav-directory_file>
```

Example 9: An XML navigation directory file

Header files are described in section 3.7.1. The description of the observation-event file format may be found in section 3.7.2. In section 3.7.3 the format for correlation matrix files is described. Sections 3.7.4 to 3.7.7 describe the several kinds of files that adhere to the observations-event file format: observations, instruments, parameters (states) and observation's residuals. The description of the three types of files using the correlation matrix file format (correlation matrices for observations, parameters and observation's residuals) may be read in section 3.7.8.

Sections 3.7.9 and 3.7.10 comment on, respectively, (input) options files and (output) process log files. Section 3.7.11 describe the navigation metadata file.

Finally, section 3.7.12 describes the header files used in the CI interface. These are particular instances of observations and instruments files tailored to be used with the network (CI) interface.

3.7.1 The header accompanying actual raw data files

As stated in section 3.4.1, all data files in the ASTROLABE FI consist, in fact of at least two files: a header file plus at least a single “raw” data file. Data files may, in turn, be split into several fragments for security reasons. Section 3.7.2.5.1 describe how raw data files that have been split must be organized and what are the rules to name these.

This section takes care of describing the format of the header file (in fact, it is a metadata file used to describe data properly). Example 1 on page 30 depicts a header used to describe observation-event (obs-e_file) data sent through a socket connection using port 2000. Example 10 below shows another header file, this time describing obs-e raw data stored in a binary file whose name (or base name, see section 3.7.2.5.1 on split files’ organization and naming) is `february_campaign.obs`.

```
<?xml version="1.0" encoding="UTF-8"?>
<astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
    <lineage version="1.0">
      <id> id1 </id>
      <name> Example of an ASTROLABE header file </name>
      <author>
        <item> J. Navarro </item>
      </author>
      <organization> CTTC </organization>
      <department> GEON </department>
      <date_time>2017-10-04T09:23:15-08:00</date_time>
      <ref_document>
        <item> ASTROLABE ICD </item>
      </ref_document>
      <project> GEMMA </project>
      <task> ASTROLABE interface control document </task>
      <remarks> Header for obs-e raw data stored in a binary file </remarks>
  </lineage>
  <data>
    <device type="obs-e_file" format="binary_file"> february_campaign.obs </device>
  </data>
</astrolabe-header_file>
```

Example 10: A header file for obs-e data stored in a binary file

ASTROLABE (data) header files are composed of:

1. A standard XML header,,
2. A lineage object (see section 3.4.2 for details).
3. A <data> tag, describing the file (or data stream in the case of sockets) where data is stored (or how it is sent / received).
4. A closing XML tag.

The XML header (bullet 1 above) is shown in examples 1 and 10. It is necessary to remark that the root tag is `astrolabe-header_file` and that the name of the schema used to validate the grammar of such kind of files must be named `astrolabe-header_file.xsd` (both in red in example 10). The closing tag (bullet 4) must obviously correspond to the root tag, so it is `/astrolabe-header_file` as shown in the last line of the example. Lineages (bullet 2) are described in section 3.4.2 since these are extensively used by many kind of files in the ASTROLABE FI).

The <data> tag is actually where the definition of data takes place. By means of the <device> sub-tag, all the parameters characterizing data are provided.

The <device> sub-tag accepts two attributes, `type` and `format`, and one value.

Attribute type accepts two values: [OBS-E_FILE] and [R-MATRIX_FILE]. When type is set to [OBS-E_FILE], it indicates that the raw data stored (in the case of files) or transmitted (sockets) adhere to the observation-event format described in section 3.7.2. On the contrary, when correlation matrices of whatever kind (section 3.7.3) have to be stored or transmitted, type must be set to [R-MATRIX_FILE].

The format attribute indicates how information is stored or represented. There exist three possible values for this attribute, [BINARY_FILE], [TEXT_FILE] and [SOCKET]. The first two values stand for files in binary or text formats respectively; the third one is used to characterize data that will be sent / received through a socket connection.

The value related to the <device> tag is either a file name (when type is either [BYNARY_FILE] or [TEXT_FILE]) or a host plus port specification (type is [SOCKET]).

File names may not include a leading path. That is, file names like the following ones

- `/home/myusername/mydata/february_campaign.obs`
- `C:\users\myusername\mydata\february_campaign.obs`

are illegal. This implies that **a header file must reside in the same directory or folder than the data it is describing.**

When sockets are used, the syntax for the host / port value is:

```
host_or_ip_address:port_number
```

- host_or_ip_address: This element is optional. Denotes the name or ip address of the host to connect to. header IP addresses are of the form xxx.xxx.xxx.xxx. It may be present only when the CI data file is used to configure a *listener* (also *consumer or receiver*) process. This includes de colon(:) separating both parts of the specification.
- port_number: Port used to set the connection. Must be a valid TCP / IP port. Mandatory in all cases, either for *sender* (or *producer*) or listener (consumer, receiver) processes.

Example 11 shows several valid and invalid values for the <device> tag in CI data files, depending on whether these are targeted at sender or listener processes.

<device> tag value	Listener (server socket)	Sender (client socket)
`35000`	`Valid. Only a port number is needed.`	`Invalid. Missing host or IP address.`
`some.host.com:3528`	`Invalid. Host may not be specified.`	`Valid. All required data is present (host name is needed and present).`
`some.host.com 3533`	`Invalid. Host may not be specified. Incorrect syntax: missing colon.`	`Invalid. Incorrect syntax. Missing colon.`
`92.188.23.15:3566`	`Invalid. Host may not be specified.`	`Valid. All required data is present (host set as an IP address).`

Example 11: <device> tag legal and illegal values for socket data transmission

Almost all the combinations of the type and formats attributes are accepted. In fact, the only exceptions are instrument and observation residuals data. For clarity reasons, table 5 shows the valid combinations of these attributes

Data	Sections	Types	Formats
Observations (measurements)	3.7.2,3.7.4	[OBS-E_FILE]	All accepted formats
Instruments	3.7.2,3.7.5	[OBS-E_FILE]	[TEXT_FILE], [SOCKET]
Parameters (states)	3.7.2,3.7.6	[OBS-E_FILE]	All accepted formats
Observation residuals	3.7.2,3.7.7	[OBS-E_FILE]	[BINARY_FILE], [TEXT_FILE]
Correlation matrices	3.7.3,3.7.8	[R-MATRIX_FILE]	All accepted formats

Table 5: Valid combinations of type and format attributes in data header files

Note that future versions of the specification might include the missing formats for instruments and observation residuals. However, only the ones listed in table 5 are supported nowadays.

3.7.2 The observation-event file format: <obs-e_file>

The <obs-e_file> format is used by several types of input and output data files included in the ASTROLABE FI. For the sake of clarity, the following sections, describing this format, refer constantly to “observations” or “measurements” as if this kind of data were the only one that could be represented using these files.

Sections 3.7.5 to 3.7.8 describe, respectively, instrument and parameter (state) files. These use the <obs-e_file> format to store their data. Therefore, these references to “observation” or “measurements” must be conveniently replaced by “instrument”, or “parameter” or when interpreting the following sections in the context of these other data files.

Since there are several kind of files relying on the <obs-e_file> format, no default extension is suggested. These extensions are proposed individually for each kind of <obs-e_file> based file in later sections of this document.

3.7.2.1 An overview

Observations and the way these must be processed are contained in the ASTROLABE observation-event files (<obs-e_file> file type).

This file contains, essentially, two types of serialized objects: observations (measurements) and reference to observation (measurement) equations. In the context of the observation-event files, these two serialized objects will be also called “I record” and “o record” respectively.

For the sake of simplicity, from now on, observations (measurements) will be called simply “observations” and observation (measurement) equations will be called just “observation equations”.

Observations will be stored using I-records; observation equations are saved as o-records.

An observations-event file is a (ascending) time-sorted set of *epochs*. An epoch is a set of observations and observation equations that share the same time tag – that is, the time tag value for both types or records have exactly the same value. Epochs must start with an observation (an I-record). See Figure 4 for a depiction of such structure.

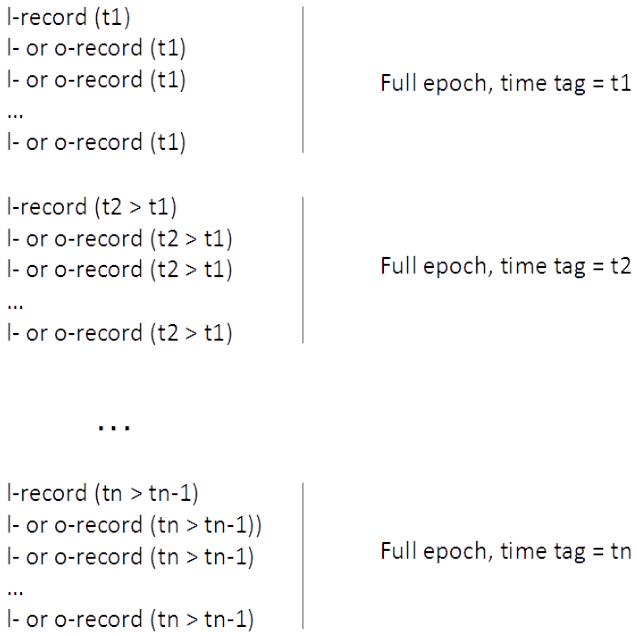

Figure 4: Conceptual structure of an observations-event file

3.7.2.2 Active versus removed records

The description of both I and o records includes an item named “Active flag” (see section 3.7.2.3). Such flag is provided with one purpose in mind: identifying those records that should take part or not in any computation using these data as input. Active records qualify to take part in these processes; removed records, do not.

The active flag is provided, in fact, as a mechanism to avoid the need to physically remove an unused record from the file where it is stored. Changing the value of this flag to “removed” must have the same effect than actually removing it from the file.

Software reading observation-events files must take this flag into account; a removed record must be ignored.

An important comment about removed records is that these are not to be taken into account in the following situations:

- New epochs – Removed records do not start new epochs, even if these really have a time tag that changes with respect to the last active record read or written.
- Epochs must start with an I-record – The FI interface states that new epochs must start with an I-record. Reading or writing an o-record at the beginning of an epoch is illegal *unless* it is removed (since it must be automatically discarded)

In general, removed records are to be considered as if they would have never existed.

3.7.2.3 The I and o record types

Tables 6 and 7 describe, respectively, the contents of I (observation) and o (observation equation) records from a pure conceptual standpoint. No assumptions are made on how the items described in these tables are stored on actual files or sent through network connections.

Item	Description	Data type	Dimension / comments	Legal values
Event (record) tag	Identifies the type of event (record)	Character	1	Always an “I” (lower case letter l)
Active flag	Flag stating the status of the whole record (active or removed)	Boolean	1	True (for active events) or false (for removed or inactive ones).
Identifier	Unique identifier used to determine the kind of observation	Character	Variable number of characters. There are no limitations on such number. Must be, however, at least one character long. Its value must correspond to the value of one <lineage><id> tag found in observation metadata. See section 3.5.	Only letters, numbers and other characters that may be used in *file names*
Instance identifier	Unique value to tell apart different instances of observations sharing the same identifier	Integer	1 See section 3.5.	Any valid integer value
Time tag	Time associated to the whole observation	Real	1	Any valid real value
Tags	Ancillary data that, without been part of the observation, helps to complement it	Real	Variable. The actual number of tags depends on each type of observation. May be zero, since tags are optional.	Any valid real value
Expectations	The observed values	Real	Variable. Depends on each type of observation.	Any valid real value
Covariance matrix	Estimated error of the expectations	Real	If n is the number of expectations, 0 (omitted covariance matrix), n (only standard deviations are provided) or n*(n+1)/2 (full covariance matrix). Standard deviations come first. If correlations are provided, these must appear in left-to-right, then up-down order. See section 3.6.1.1.8 for details on how to write covariance matrices	Real values greater than 0 (standard deviations) or between -1.0 and 1.0 (correlations)

Table 6: Conceptual description of an I-record (observation)

Item	Description	Data type	Dimension / comments	Legal values
Event (record) tag	Identifies the type of event (record)	Character	1	Always an “o” (lower case letter o)
Active flag	Flag stating the status of the whole record (active or removed)	Boolean	1	True (for active events) or false (for removed or inactive ones).
Observation equation identifier (model identifier)	Unique identifier used to tell apart the model to use to relate the different kinds of data involved in the equation	Character	Variable number of characters. There are no limitations on such number. Must be, however, at least one character long. Its value must correspond to the value of one <lineage><id> tag found in model metadata. See section 3.5.	Only letters, numbers and other characters that may be used in *file names*
Time tag	Time associated to the observation equation	Real	1	Any valid real value
List of parameter (states) instance identifiers	List of instance identifiers of the parameters (states) intervening in the observation equation.	Integer	Variable. The actual number of parameter identifiers depend on each type of observation equation	Any valid integer value
List of observation instance identifiers	List of instance identifiers of the observations intervening in the observation equation.	Integer	Variable. The actual number of observation identifiers depend on each type of observation equation	Any valid integer value
List of instrument instance identifiers	List of the instance identifiers of the instruments intervening in the observation equation.	Integer	Variable. The actual number of instrument identifiers depend on each type of observation equation. May be zero, since instrument instance identifiers are optional.	Any valid integer value

Table 7: Conceptual description of an o-record (observation equation)

Note the use of the words “identifier” and “instance identifier” in tables 6 and 7. Refer to section 3.5. for a discussion on the meaning of terms as “type”, “identifier” and “instance identifier” and how these are used to cross-reference data and metadata.

All l-records must have a covariance matrix. When it is not specified in the l-record itself (see how in Table 6 it is stated that covariance matrices may be omitted) then a default one must be provided by means of metadata. See section 3.6 for a complete description on ASTROLABE metadata and section 3.6.1.1.8 for a specific description of default covariance matrices.

3.7.2.4 The organization of the <obs-e_file>

As every data file in the ASTROLABE interface, <obs-e_file>s consist of (at least) two files:

- the XML, text header and,
- an (set of) files containing actual observation-event data (that is, the actual set of l- and o-records).

Note that there exist a single exception to the statement above: data transmitted through a socket connection consists only of a header file.

Example 12 below shows an header for an observation-events text file. Note the attributes type and format in the <device> tag. There it is stated that data are observation-events and that they have been stored in a text file. The name of such file is `simulated_data.obs`.

```
<?xml version="1.0" encoding="UTF-8"?>
<astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
    <lineage version="1.0">
      <id> id1 </id>
  </lineage>
  <data>
    <device type="obs-e_file" format="text_file"> simulated_data.obs </device>
  </data>
</astrolabe-header_file>
```

Example 12: A header file pointing to observation-events data stored in a text file

Example 13 shows the raw observation-events file. Note that this is a pseudo-XML file, since in spite of using XML-like tags and attributes, no headers, root nor closing tags are used.

```
  <l s="a" id="Code1" n="0">  1.000000000000000e+000 … 3.000000000000000e+000 </l>
  <l s="a" id="Code1" n="1">  1.000000000000000e+000 … 4.000000000000000e+000 </l>
  <l s="r" id="Code1" n="2">  1.000000000000000e+000 … 5.000000000000000e+000 </l>
  <o s="a" id="ModelA"> 1.000000000000000e+000 0 1 </o>

  …

  <l s="a" id="Code5" n="872"> 2.260000000000000e+002 … 5.875000000000000e+003 </l>
  <o s="a" id="ModelB"> 2.260000000000000e+002 871 872 </o>
```

Example 13: An <obs-e_file >with actual data

Of course, the file shown in Example 13 might have been a binary one. The value of the attribute “format” in the <device> tag in Example 12 should have been changed to “binary_file”.

Section 3.7.2.5.2 describe the syntax of raw observation-events text files. The format used in raw observation-events binary files is detailed in section 3.7.2.5.3.

3.7.2.5 The available formats

The FI define two formats to store observation-events raw data. These are classified as follows:

- Binary format. Data is stored in binary form, which reduces considerably the size of the intervening files and simplifies the actual read / write operations. Data are actually stored in one or more binary files.

- Text format. Data is stored as plain text, pseudo-XML human-readable form. The main advantage of this format is that files are easily editable by human operators needing to correct the information these include. The main disadvantage is that text files need much more disk space to store they information they contain. Data are actually stored in one or more text, XML-like files.

Before explaining in detail how these two FI formats are actually defined, some considerations need to be made concerning raw data files. Section 3.7.2.5.1 takes cares of this. Sections 3.7.2.5.2 to 3.7.2.5.3 proceed then to specify the different available formats.

3.7.2.5.1 Raw data (text or binary) files. Organization. Naming conventions

Actual raw data, either in text or binary formats, may be stored in just a single file or a set of these. That is, the whole contents of the observation-events data may be split over several files. The number of split files actually used will depend on the amount of data that needs to be stored. The size and number of each of these files may be freely decided by the designers of software adhering to this ICD. Such number may be, of course, just one.

The reason to split actual data in a set of files instead of using just one is *security*. Observation-events files may be generated in real time by applications directly capturing data from sensors. An unexpected malfunctioning in such applications may lead to data corruption. Splitting the output in several files serves to guarantee that at least all files but the last one will contain correct data. We will refer to these files as *split files.*

Split files must be organized as follows:

- Data is sorted by ascending time. This means that the first split file will store the oldest data, while the last one will contain the newest information.
- Internally, each split file is also sorted by ascending time. So the beginning of the file will keep again the oldest data while its end will store the newest.
- Must always contain complete epochs. That is, epochs may not be stored across files.

Splitting data into several files has direct implications on *file naming*. The XML observation-events file header includes a base file name in its <device> tag as shown in example 14:

```
<device type="obs-e_file" format="text_file"> july_campaign_data.obs </device>
```

Example 14: A <device> tag and the base name for raw data files

But this base file name (“july_campaign_data.obs” in the example) is not, in fact the name of the different split files actually storing data. The reason are:

- there may be more than one split file, being necessary several actual file names to uniquely identify these, and
- these must be named in such a way that the time-ascending sorting rule stated above is respected.

Therefore, a file naming convention using the base file name written in the <device> XML tag is needed to derive the names of each actual split file. These are the rules that define such file naming convention for split files:

- The pattern to follow is <base_name>_<numeric_suffix>[<.extension>]
- <base_name> and <.extension> correspond to the base file name and its (optional) extension written in the XML <device> tag value.
- <.extension> is optional since the base file name written in the XML <device> tag value may not include such extension.

- <numeric_suffix> is a series of 1 up to 9 digits.
- The number of digits in a <numeric_suffix> must be exactly the same for all the split files in a set.
- If the number of digits in a <numeric_suffix> for a given set of split files is n, all suffixes must use n digits, left-padding the suffix with as much zeroes as necessary.
- The <numeric_suffix> of a series of split files must start always at 1 (not 0) and be incremented by one for each file present in the set.

The maximum number of split files that may be generated according to these rules depends directly on the number of digits in the <numeric_suffix>. If a width of only one digit is selected, then only 9 files may be created (obviously numbered from 1 to 9). Using a <numeric_suffix> 9 digits wide, will allow for 999999999 split files in the same set.

Example 15 includes several sets of file names that adhere to the naming convention described above. All file naming examples assume that the value given in the XML <device> tag for the base file name is “july_campaign_data.obs”.

```
july_campaign_data_001.obs                  july_campaign_data_000001.obs
july_campaign_data_002.obs                  july_campaign_data_000002.obs
july_campaign_data_003.obs                  july_campaign_data_000003.obs
july_campaign_data_004.obs                  july_campaign_data_000004.obs
july_campaign_data_005.obs                  july_campaign_data_000005.obs

july_campaign_data_1.obs                    july_campaign_data_000000001.obs
july_campaign_data_2.obs                    july_campaign_data_000000002.obs
july_campaign_data_3.obs                    july_campaign_data_000000003.obs
july_campaign_data_4.obs                    july_campaign_data_000000004.obs
july_campaign_data_5.obs                    july_campaign_data_000000005.obs
```

Example 15: Valid names for a set of five split external files

The file names include in example 16 are, on the other side, invalid ones. The reasons explaining why are specified along with each example.

```
(Numbering starting at 0)                   (Variable length of suffix)
july_campaign_data_000.obs                  july_campaign_data_000001.obs
july_campaign_data_001.obs                  july_campaign_data_002.obs
july_campaign_data_002.obs                  july_campaign_data_0003.obs
july_campaign_data_003.obs                  july_campaign_data_0000004.obs
july_campaign_data_004.obs                  july_campaign_data_000005.obs

(Numeric suffix beyond 9 digits)            (Missing "_" between file name / suffix)
july_campaign_data_00000000001.obs          july_campaign_data000001.obs
july_campaign_data_00000000002.obs          july_campaign_data000002.obs
july_campaign_data_00000000003.obs          july_campaign_data000003.obs
july_campaign_data_00000000004.obs          july_campaign_data000004.obs
july_campaign_data_00000000005.obs          july_campaign_data000005.obs
```

Example 16: Invalid names for a set of five split external files

Note again that the <extension> is optional since it may not be present in the <device> tag. Assuming that the base file name was “july_campaign_data” instead, example 17 shows again several sets of valid names for split files.

july_campaign_data_001 july_campaign_data_002 july_campaign_data_003 july_campaign_data_004 july_campaign_data_005 july_campaign_data_1 july_campaign_data_2 july_campaign_data_3 july_campaign_data_4 july_campaign_data_5	july_campaign_data_000001 july_campaign_data_000002 july_campaign_data_000003 july_campaign_data_000004 july_campaign_data_000005 july_campaign_data_000000001 july_campaign_data_000000002 july_campaign_data_000000003 july_campaign_data_000000004 july_campaign_data_000000005

Example 17: Valid names for a set of five split external files. No extension in original file base name

3.7.2.5.2 The format of raw text data files

Raw text observation-events data files are pseudo-XML files. Although these adhere to the principles of the XML syntax, neither headers nor the tag closing the root one appear. Example 18 shows one of such files. Note how the aforementioned elements are missing in the example.

```
<l s="a" id="Code1" n="0">  1.000000000000000e+000 … 3.000000000000000e+000 </l>
<l s="a" id="Code1" n="1">  1.000000000000000e+000 … 4.000000000000000e+000 </l>
<l s="r" id="Code1" n="2">  1.000000000000000e+000 … 5.000000000000000e+000 </l>
<o s="a" id="ModelA"> 1.000000000000000e+000 0 1 </o>

…

<l s="a" id="Code5" n="872"> 2.260000000000000e+002 … 5.875000000000000e+003 </l>
<o s="a" id="ModelB"> 2.260000000000000e+002 871 872 </o>
```

Example 18: A pseudo-XML (text) obs-e_file showing some l- and o-records

In example 18:

- Data for l- and o-records are fictitious and incomplete
- The XML format, which be described below, is shown.

Section 3.7.2.3 describes l- and o-records from the *conceptual* standpoint, When these records are stored in an XML text file the right (XML-compliant) syntax must be used.

An **l-record** must be written according to the following syntax:

<l s="*the_active_flag*" id="*the_observation_id*" n="*the_instance_identifier*"> *the_time_tag the_tags the_expectations the_covariance_matrix* </l>

where

- Text in italics (as "*the_observation_id*") is used as placeholders for actual data. The name of the placeholder is used to refer synoptically to the items described in Table 6.
- The event (record) tag (lowercase letter "l") is directly given as the name of the opening tag, which is "<l>".
- Attribute "s" (meaning "status") serves to provide with the value of the active flag (placeholder: *the_active_flag*). Legal values are either "a" (for active) or "r" (for removed, inactive).
- Attribute "id" is used to specify the observation identifier (placeholder: *the_observation_id*). This

identifier must match the value of an <lineage><id> tag included in the description of one observation in the navigation metadata (see section 3.5).

- Attribute “n” introduces the instance identifier for the l-record (see section 3.5). The placeholder is *the_instance_identifier*.
- The values of the observation's instance identifier (*the_instance_identifier*), time tag (*the_time_tag*), tag values (*the_tags*), expectations (*the_expectations*) and their covariance matrix (*the_covariance_matrix*) are written at this point (see section 3.5 for a discussion on identifiers and instance identifiers). Note that the covariance matrix is optional.
- The l-record is closed by means of the character sequence “</l>”.

The three attributes included in the <l> tag (“s”, “id” and “n”) may appear in any order. The remaining data (time tag, tags, expectations and covariance matrix) must appear exactly in the order specified here.

Note that the number of tag values, expectations and elements in their covariance matrix depend on each kind of observation (l-record). Note also that covariance matrices may be omitted or written either as standard deviations only or standard deviations plus correlation values. When covariance matrix are not present, there must exist a default covariance defined by means of a <c> tag in the corresponding metadata definition (see section 3.6.1.1.8 or Table 6).

Real values (as tags or expectations) must adhere to the syntax of the xs:double element as defined by the XML standard, which is equivalent to the definition of floating point values stated by the IEEE in its standard IEEE 754. Examples of valid values are: "123.456", "+1234.456", "-1.2344e56", "-.45E-6", "INF", "-INF", or "NaN".

The following would be a valid example of an l-record where the number of tags is 1 and the number of expectations and elements in the covariance matrix (including *only* standard deviations) is 2:

```
<l s=”a” id=”myobs” n=”23”> 7.02e-1 0.00003 1.20003 88.49343 1.0005 5.0002 </l>
```

In this l-record example,

- the active flag (attribute “s”) has been set to “active” (“a”),
- the observation identifier (attribute “id”) is “myobs”,
- the instance identifier (attribute “n”) of the observation is 23,
- the time tag is 7.02e-1,
- the only tag is 0.00003,
- the values for the expectations are 1.20003 and 88.49343 and
- those for the covariance matrix standard deviations (there are no correlations) are 1.0005 and 5.0002.

An **o-record** must adhere to the syntax below:

<o s=”*the_active_flag*” id=”*the_observation_equation_id*”> *the_time_tag the_list_of_p_instance_ids the_list_of_o_instance_ids the_list_of_i_instance_ids* </o>

where

- Text in italics (as “*the_observation_equation_identifier*”) is used as placeholders for actual data. The name of the placeholder is used to refer synoptically to the items described in Table 7.
- The event (record) tag (lowercase letter “o”) is directly given as the name of the opening tag, which is “<o>”.
- Attribute “s” (meaning “status”) serves to provide with the value of the active flag (placeholder: *the_active_flag*). Legal values are either “a” (for active) or “r” (for removed, inactive).
- Attribute “id” is used to specify the observation identifier (placeholder: *the_observation_equation_id*). See section 3.5.
- The values of the observation equation's time tag (*the_time_tag*), parameter instance identifiers (*the_list_of_p_instance_ids*), observation instance identifiers (*the_list_of_o_ids*) and instrument

instance identifiers (*the_list_of_i_instance_ids*) are written at this point. Note that while the list of parameter and observation instance identifiers are mandatory, that of instrument instance identifiers is optional.

- The o-record is closed by means of the character sequence </o“>”.

The two attributes included in the <o> tag (“s”, and “id”) may appear in any order. The remaining data (the time tag and three lists of identifiers) must appear exactly in the order specified here.

The number of elements in the lists of parameter, observation and instrument instance identifiers depends on each kind of observation equation. Note that there may be no instrument instance identifiers at all, since these are optional.

Real numbers, exactly as in the case of l-records, must be written according to the IEEE 754 standard.

Assuming an observation equation where the number of parameter, observation and instrument instance identifiers were respectively 2, 1 and 1, the following would be a valid example of an o-record written in XML syntax:

```
<o s=”r” id=”myobsequation”> 234.567 45 18 33 18 </o>
```

In this o-record example,

- the value of the active flag (attribute “s”) is “removed” (inactive, “r”),
- the identifier of the observation equation (attribute “id”) is “myobsequation”,
- the time tag has a value of 234.567,
- the parameter instance identifiers are 45 and 18,
- the only observation instance identifier is 33 and
- the only instrument instance identifier is 18.

3.7.2.5.3 The format of binary files

This format is the binary equivalent to the raw text files described in section 3.7.2.5.2.

Each split raw binary file contains a series of l- and o-records grouped into epochs, each of these separated by *backtracking records*. Figure 5 depicts schematically such structure.

Note how the successive epochs' time tags are sorted in ascending order, that all records in an epoch share the same time tag and that all epochs always start with an l-record – thus adhering to the requisites set to observation-events files.

l- and o-records contain the data described in sections 3.7.2.3. However, this section explains the different data items making l- and o-records from a conceptual standpoint. Tables 8 and 9 describe the *actual* representation used to store this information *when using binary files*.

A *backtracking (b-) record* is composed of two items as shown in Table 10. The first one is the record tag (a lowercase letter 'b' in this case) used to tell apart this record. The second item contains the amount of bytes needed to store the epoch located just before the b-record itself, that is, to store the preceding full epoch. Note that this amount does **not** include the bytes used by b-records themselves.

A binary files always ends with a b-record. There is no b-record at the beginning of a binary file, since this would mean that another epoch, preceding such b-record, should have been written to the file.

Figure 7 is a reinterpretation of Figure 5, where epochs are shown as blocks (which, of course, include a series of l- and o-records) and b-records are depicted explicitly.

Figure 5: Overall organization of a binary file

3.7.2.5.3.1 What are backtracking records and why are these necessary?

I- and o-records have variable lengths because the actual number of items these store depend on the actual kind of observation or observation equation being represented (see section 3.7.2.3).

In spite of these variability in length, I- and o-records stored in XML are easily identifiable because of the syntax used; opening and closing tags (i.e. <I> and </I>) are the markers delimiting the starting and ending points for such records. No other elements in the file use such markers, so no ambiguity exists.

This means that, even when working in backwards mode, a reader has the necessary information to analyse the contents of an observations-event file. The aforementioned markers provide the required reference points to split text into meaningful units (records). And this is true for both forward and backwards readers.

On the contrary, such markers do no exist in binary files. These store information in a much more compact form; for instance, a double value may take up to 22 characters / bytes in text form, while 8 bytes are enough to store the same information in a binary file.

When reading in forward direction, even with binary files, it is possible to identify the successive elements integrating a record. The reader receives the extra information needed to learn how many elements need to be read (as for instance, the number of items composing the set of expectations). So, in spite of the variable length of these records, all the necessary information to read the next record is always at hand.

On the contrary, backwards readers face the problem that it is necessary to jump to the beginning of an epoch and then read it in forward direction until all the records that integrates such epoch are exhausted. This means that it is necessary to jump back an arbitrary number of bytes that cannot be determined if no extra metadata is included in the file. Such metadata are the backtracking (b-) records. Thanks to these, the reader may actually jump back the necessary amount of bytes to travel to the beginning of the epoch and, then, proceed as a regular forward reader would do: receiving the extra (external) information needed to ascertain the number of individual items in each of the components whose length is variable. When the epoch is exhausted, the process is repeated to travel to the previous epoch and the read in forward direction until the beginning of the file is reached.

To summarize:

- Backtracking (b-) records are actual markers *delimiting* **full epochs**, so there is no doubt about where these start or end.
- Since the amount of bytes needed by the preceding epoch is stored, backwards binary readers are able to compute the length of the backwards jump to perform to locate the beginning of such epoch.
- Forward, binary readers need not b-records to proceed, so these must simply ignore such markers.

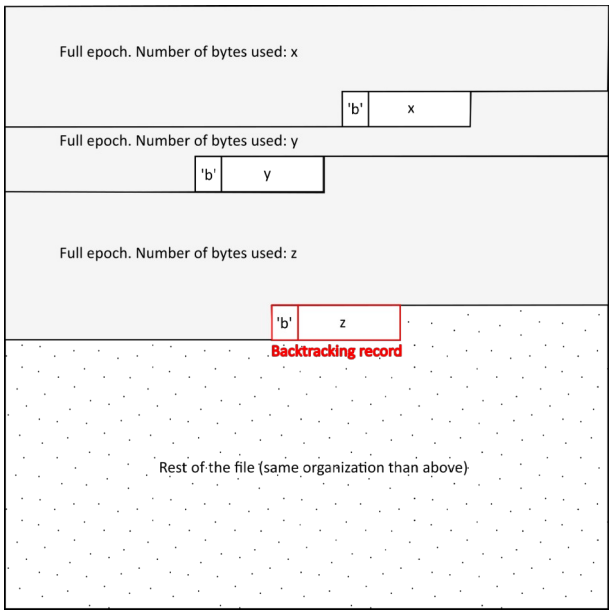

Figure 6: Epochs and backtracking records

3.7.3 The correlation matrix file format: <r-matrix_file>

As it happens with the <obs-e_file> format (section 3.7.2), the <r-matrix_file> format is used by several types of input and output files included in the ASTROLABE FI. These are the correlation matrix for trajectory observations file (<tr_obs_correlation_Rll>), the correlation matrix for trajectory parameters file (<tr_states_correlation_Rxx>) and the correlation matrix for residuals file (<tr_res_correlation_Rvv>).

Since there are several kind of files relying on the <r-matrix_file> format, no default extension is suggested. These extensions are proposed individually for each kind of <obs-e_file> based file in later sections of this document.

The <r-matrix_file> format is similar in many aspects to the <obs-e_file> format. Therefore, this section will constantly refer the reader to the appropriate subsections of section 3.7.2 when needed in order to avoid unnecessary repetition.

Item	Description	C++ type	Dimension / comments	Legal values
Event (record) tag	Identifies the type of event (record)	char	1	Always an “l” (lower case letter l)
Active flag	Flag stating the status of the whole record (active or removed)	char	1	1 for active events, 0 for removed ones. Note that these are *numeric* values, not string ones.
Length of the identifier	Number of characters used to store the identifier	4-byte int	1	Strictly greater than zero.
Identifier	Unique identifier used to determine the kind of observation	char	As many char items as specified by the length of the identifier above	Only letters, numbers and other characters that may be used in *file names*
Instance identifier	Unique value to tell apart different instances of observations sharing the same identifier	4-byte int	1	Any valid integer value
Time tag	Time associated to the whole observation	8-byte double (IEEE 754)	1	Any valid real value
Tags	Ancillary data that, without been part of the observation, helps to complement it	8-byte double (IEEE 754)	Variable. The actual number of tag values depends on each type of observation. May be zero, since tags are optional.	Any valid real value
Expectations	The observed values	8-byte double (IEEE 754)	Variable. Depends on each type of observation	Any valid real value
Covariance matrix	Estimated error of the expectations	8-byte double (IEEE 754)	If n is the number of expectations, either 0 (omitted covariance matrix), n (only standard deviations are provided) or n*(n+1)/2 (full covariance matrix). Standard deviations come first. If correlations are provided, these must appear in left-to-right, then up-down order. See section 3.6.1.1.8 for details on how to write covariance matrices	Real values greater than 0 (standard deviations) or between -1.0 and 1.0 (correlations)

Table 8: Format of an observation event record stored in binary format

Item	Description	C++ type	Dimension / comments	Legal values
Event (record) tag	Identifies the type of event (record)	char	1	Always an “o” (lower case letter o)
Active flag	Flag stating the status of the whole record (active or removed)	char	1	1 for active events, 0 for removed ones. Note that these are *numeric* values, not string ones.
Length of the observation equation identifier	Number of characters used to store the observation equation identifier	4-byte int	1	Strictly greater than zero.
Observation equation identifier (model identifier)	Unique identifier used to tell apart the model to use to relate the different kinds of data involved in the equation	char	As many char items as specified by the length of the observation equation identifier above	Only letters, numbers and other characters that may be used in *file names*
Time tag	Time associated to the observation equation	8-byte double (IEEE 754)	1	Any valid real value
List of parameter (states) instance identifiers	List of instance identifiers of the parameters (states) intervening in the observation equation	4-byte int	Variable. The actual number of parameter identifiers depend on each type of observation equation	Any valid integer value
List of observation instance identifiers	List of instance identifiers of the observations intervening in the observation equation	4-byte int	Variable. The actual number of observation identifiers depend on each type of observation equation	Any valid integer value
List of instrument instance identifiers	List of instance identifiers of the instruments intervening in the observation equation	4-byte int	Variable. The actual number of instrument identifiers depend on each type of observation equation. May be zero, since instruments are optional	Any valid integer value

Table 9: Format of an observation equation event record stored in binary format

Item	Description	C++ type	Dimension / comments	Legal values
Event (record) tag	Identifies the type of event (record)	char	1	Always a “b” (lower case letter b)
Bytes in preceding epoch	The total number of bytes used to store all the l- and o-records in the epoch located just *before* the b-record itself	4-byte int	1	Strictly greater than zero

Table 10: Format of a backtracking record stored in binary format

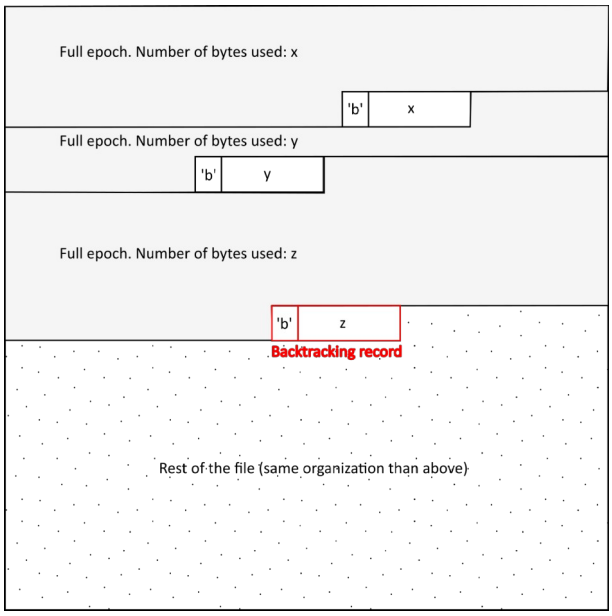

Figure 7: Epochs and backtracking records

3.7.3.1 An overview

The <r-matrix_file> family of files contain an (ascending) time-ordered set of correlation matrices. Each correlation matrix is tagged with a unique time stamp or tag. It is not possible to store two correlation matrices with the same time tag in a <r-matrix_file>.

Each of these correlation matrices and their corresponding time tag are called “r-record”. Either of these terms (correlation matrix, r-record) will be used in this document when referring to correlation matrices in the context of <r-matrix_file> family of files.

In short, an <r-matrix_file> may be described as follows:

r-record (t_0)

r-record (t_1)

…

r-record (t_n) where $t_0 < t_1 < \ldots < t_n$

3.7.3.2 Active versus removed records

As it happens with the l- and o-records in the <obs-e_file> format, r-records includes an item named “Active flag”. The purpose of this flag is exactly the same than for l- and o-records and works in the same way. See section 3.7.2.2 for details.

3.7.3.3 The r record type

Table 11 describes the contents of r-records from a pure conceptual standpoint. No assumptions are made on how the items described in these tables are stored on actual files.

Item	Description	Data type	Dimension / comments	Legal values
Record tag	Identifies the type of record	Character	1	Always an “r” (lower case)
Active flag	Flag stating the status of the whole record (active or removed)	Boolean	1	True (for active records) or false (for removed or inactive ones)
Time tag	Time associated to the whole correlation matrix	Real	1	Any valid real value
Correlation matrix	Correlation matrix for the data entity being described.	Real	If n is the number of elements whose correlation matrix is stored, then the number of elements is n(n-1)/2, Only the elements below the diagonal are stored. These are stored first left-to-right and then top to bottom.	Real values between -1.0 and 1.0.

Table 11: Conceptual description of an r-record (correlation matrix)

The time tags in r-records must correspond to those in the epochs existing either in observation or parameter files (both adhering to the <obs-e_file> format.

A correlation matrix relate the instances of the observations (or parameters, or observations residuals) found in one epoch of the corresponding <obs-e_file> files, in the same order these appear. There exist a correlation matrix for every epoch in the observations (parameters) file.

For example, assuming that the following are the instance identifiers (see section 3.5) - appearing in the order below - in some epoch in an observations (or parameters) file:

```
45 28 22 19 30
```

then the correlation matrix would be the next one:

	45	28	22	19	30
45	1	a	b	d	g
28	a	1	c	e	h
22	b	c	1	f	i
19	d	e	f	1	j
30	g	h	i	j	1

The numbers at the first row and columns represent the aforementioned identifiers. The letters to the right and below the rulers are the actual values of the correlation matrix.

Since correlation matrices are symmetric, and the values in the diagonal are always 1, only the values

below the diagonal need to be stored in the correlation matrix field of the r-record. In the example above, these would be

```
a b c d e f g h i j
```

These values are stored from left to right and then from top to bottom.

Since the number of observation (or parameter) instances in an epoch is variable, the dimension of successive correlations matrices may vary; the actual observation (or parameter) instances involved in the matrix may also change. For instance the next one could be the sequence of instance identifiers found in a different epoch:

```
28 45 33
```

Then, the correlation matrix would be:

	28	45	33
28	1	k	l
45	k	1	m
33	l	m	1

and the values actually stored in the correlation matrix of the r-record would be:

```
k l m
```

3.7.3.4 The organization of the <r-matrix_file>. Text and binary formats

The correlation matrix files are organized in the same way that <obs-e_files> (header + raw data files). Please refer to section 3.7.2.4 for details.

Obviously, the value of the type attribute in the header file will be different: instead of “obs-e_file” it will state that the raw data file is of type “r-matrix_file”. See example 19.

```
<?xml version="1.0" encoding="UTF-8"?>
<astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
    <lineage version="1.0">
      <id> julycampaign </id>
  </lineage>
  <data>
    <device type="r-matrix_file" format="text_file"> july_campaign.rll </device>
  </data>
</astrolabe-header_file>
```

Example 19: A header file describing an <r-matrix_file>

Once more, the formats available to store correlation matrix files in ASTROLABE's FI are the same than those available for <obs-e_files>, that is, binary and text formats.

Example 21.depicts a raw, text r-matrix file including just two r-records. Note the absence of headers and footers. This file, together with the one shown in example 20 constitute a complete r-matrix file (raw text data version).

```
<r s=”a”>
     124.88
     0.7
     0.7 0.7
     0.9 0.3 0.3
     0.3 0.9 0.3 0.5
     0.3 0.3 0.9 0.5 0.5
     0.4 0.4 0.1 0.2 0.2 0.2
     0.4 0.4 0.1 0.2 0.2 0.2 0.8
     0.1 0.1 0.1 0.2 0.2 0.2 0.8 0.7
     0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.5
     0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.5 0.1 0.0
     0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.1 0.1 0.0 0.0
     0.0 0.0 0.0 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0
     0.0 0.0 0.0 0.2 0.3 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0
     0.0 0.0 0.0 0.1 0.1 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
     0.0 0.0 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
</r>

<r s=”a”>
     124.90
     0.6
     0.6 0.8
     0.8 0.3 0.2
     0.4 0.9 0.2 0.4
     0.4 0.2 0.8 0.5 0.5
     0.3 0.3 0.2 0.1 0.1 0.2
     0.3 0.5 0.2 0.1 0.3 0.2 0.7
</r>

…
```

Example 21: An <r-matrix_file >with actual data (r-records)

Again, binary and text raw data files may be split in several fragments. In such cases, the naming convention for split data files defined in section 3.7.2.5.1 apply here.

The XML syntax for r-records (for XML-like text files) is the following:

<r s=”*the_active_flag*”> *the_time_tag the_correlation_matrix* </r>

where

- Text in italics (as “*the_active_flag*”) is used as placeholders for actual data. The name of the placeholder is used to refer synoptically to the items described in Table 11.
- The record tag (lowercase letter “r”) is directly given as the name of the opening tag (“< r>”).
- Attribute “s” (meaning “status”) serves to provide with the value of the active flag (placeholder: *the_active_flag*). Legal values are either “a” (for active) or “r” (for removed, inactive).
- The value of the time tag (placeholder: *the_time_tag*) comes next.
- The values of the correlation matrix are the next ones in the record (placeholder: the_correlation_matix). See section 3.7.3.3 for a detailed description on how correlation matrices are specified.
- The r-record is closed by means of the character sequence “</r>”.

In spite of all the similarities between <obs-e_files> and <r-matrix_files>, there are noticeable differences between the format used to store binary data in these two cases.

Although binary correlation matrix file adhere to the same schema seen in <obs-e_files>, there are

some differences that deserve a detailed description.

There are almost no changes in the header files. The type attribute in example 19 must be changed to “binary_file”.

The format of the binary correlation matrix files is much simpler than the one used by <obs-e_files>.*Binary <r-matrix_file> files are a series of r-records stored in the format described in Table 12.*

Item	Description	C++ type	Dimension / comments	Legal values
Record tag	Identifies the type of record	char	1	Always an “r” (lower case)
Active flag	Flag stating the status of the whole record (active or removed)	char	1	1 for active records, 0 for removed. Note that these are *numeric* values, not string ones
Time tag	Time associated to the whole correlation matrix	8-byte double (IEEE 754)	1	Any valid real value
Correlation matrix	Correlation matrix for the data entity being described.	8-byte double (IEEE 754)	If n is the number of elements whose correlation matrix is stored, then the number of elements is n(n-1)/2, Only the elements below the diagonal are stored. These are stored first left-to-right and then top to bottom. See section 3.7.3.3 for a detailed description.	Real values between -1.0 and 1.0.

Table 12: Format of a covariance matrix record stored in binary format

These records must be stored sorted by ascending time (that is, using the value of the time tag field).

Note that there are no backtracking records, These are used by binary <obs-e_file> files to enable backwards reading operations. In the case of correlation matrix files, there is no need to read these backwards, making backtracking records unnecessary.

3.7.4 Observation (measurement) files

Observation (or measurement) input files contain data coming from sensors. These completely adhere to the <obs-e_file> format defined in section 3.7.2.

The recommended file extension of a <obs-e_file>-based observation file is [obs].

3.7.4.1 Example

Examples 22 and 23 below show an observations file (header plus text raw data files). The line numbers in the left part of the raw text data file example are not part of the file itself, but shown for easy cross-referencing. The colours in the line numbers are used to tell apart different epochs. The syntax of

l- and o-records (<l> and <o> tags) is defined in section 3.7.2.5.2. Note that no “s” (active flag) attributes have been written in this example; therefore, all records are assumed active.

```
<?xml version="1.0" encoding="UTF-8"?>
<astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
    <lineage version="1.0">
      <id> julycampaign </id>
  </lineage>
  <data>
    <device type="obs-e_file" format="text_file"> example_23.obs </device>
  </data>
</astrolabe-header_file>
```

Example 22: Header file for data in example 23

```
01    <l id=”baro1” n=”32”>        124.88 23.44 1023.44
02                                              0.3                                 </l>
03    <l id=”imu1” n=”41”>         124.88       0.01 0.02 0.015 0.32 0.43 9.95
04                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
05    <l id=”imu1_bias_pn” n=”17”> 124.88       0.0  0.0  0.0   0.0  0.0  0.0  </l>
06    <o id=”pva1_d”>              124.88       27   11   41                   </o>
07    <o id=”imu1_bias_d”>         124.88       11        17                  </o>
08    <o id=”height_update”>       124.88       27   34   32    51             </o>
09
10    <l id=”baro1” n=”33”>        124.90 23.45 1023.45
11                                              0.3                                 </l>
12    <l id=”imu1” n=”41”>         124.90       0.01 0.01 0.014 0.33 0.44 9.95
13                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
14    <l id=”imu1_bias_pn” n=”17”> 124.90       0.0  0.0  0.0   0.0  0.0  0.0  </l>
15    <o id=”pva1_d”>              124.90       27   11   41                   </o>
16    <o id=”imu1_bias_d”>         124.90       11        17                  </o>
17    <o id=”height_update”>       124.90       27   35   33    52             </o>
18
19    <l id=”baro1” n=”32”>        124.92 23.45 1023.45
20                                              0.3                                 </l>
21    <l id=”imu1” n=”41”>         124.92       0.01 0.01 0.013 0.30 0.42 9.95
22                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
23    <l id=”imu1_bias_pn”> 17 124.92           0.0  0.0  0.0   0.0  0.0  0.0  </l>
24    <o id=”pva1_d”>              124.92       27   11   41                   </o>
25    <o id=”imu1_bias_d”>         124.92       11        17                  </o>
26    <o id=”height_update”>       124.92       27   34   32    51             </o>
27
28    …
```

Example 23: Observations (measurements) file in text (XML) format

Data in example 23 match the definitions of the observations, parameters, instruments and models included in the navigation metadata file shown in example 8. The key data presented in that navigation metadata file is summarized in tables 13 and 14 to help to understand example 23. **Note that the last column in tables 13 and 14 refer to the line numbers in example 8.**

Entity	Type	Identifier	Instance identifier	Line numbers
Observation	baro	baro1	32,33	4 – 33
Observation	imu	imu1	41	34 – 56
Observation	RW_6_PN	imu_bias_pn	17	57 – 79
Parameter	pva	pva1	27	80 – 106
Parameter	IMU_bias	imu1_bias	11	107 – 127
Parameter	baro_cal	baro1_cal	34, 35	128 – 148
Instrument	baro_p0_h0	p0_h0	51, 52	149 – 187

Table 13: Summary of key metadata (observations, parameters, instruments) from example 23

Type	Identifier	Parameter ids.	Observation ids.	Instrument ids.	Line numbers
local_mec_eq_bias_d	pva1_d	pva1 imu1_bias	imu1	-	188 – 220
RW_6_d	imu1_bias_d	imu1_bias	imu1_bias_pn		221 – 243
PVA_baro_U	height_update	pva1 pva1_baro_cal	baro1	p0_h0	244 – 276

Table 14: Summary of key metadata (models) from example 23

In these tables, columns “Type” and “Identifier” point to the corresponding <type> tags in the navigation metadata file depicted in example 23. It is worth to insist in the fact that the line numbers in these two tables refer to example 8, where the respective objects are defined.

Table 13 also include a column with title “Instance identifier”. It contains **arbitrarily** generated, unique instance numbers for the different occurrences of the objects they correspond to as shown in example 23 Section 3.5 describe the meaning of the type, identifier and instance identifiers. There, it is stated that instance identifiers may take any value, and it is recommended that the same instance identifier be used across a complete data set to refer to a specific type / identifier pair. The instance identifiers shown in Table 13 might have been any others, providing the conditions stated in section 3.5 were satisfied.

From this column in table 13 it is possible to deduce that **two** different instances of the observation with type “baro” and identifier “baro1” are shown in example 23, these labelled 32 and 33. Correspondingly, two instances of instrument objects exist (baro_p0_h0 / p0_h0) with instance identifiers 51 and 52 respectively. The remaining objects in table 13 have only one instance identifier because there exist single instances of these only.

The interpretation of the double occurrence of baro / baro1 is that in example 23, data from **two** barometers are collected. These are identical (since these share the same type and identifier) but two devices were set up (instance identifiers 32 and 33) to collect data. Concerning instrument objects, the double appearance of these is shown in lines 8, 17 and 26 of the example, where the instance identifiers 51 and 52 are used as part of three o-records.

All l-records (observation (measurement)) in example 23 must include such instance identifier. According to the values shown in Table 13, all observations related to the inertial measuring unit imu / imu1 must be tagged with the instance identifier 41. Barometers (there are two), must show either of the values 32 or 33 depending on the actual barometer data come from.

Table 14 include three new columns, labelled respectively as “Parameter identifiers”, “Observation identifiers” and “Instrument identifiers”. These columns show the identifiers of the parameters / observation / instruments involved in the respective models. This will help to understand the o-records shown in example 23.

For instance, the third model in Table 14 states that the identifiers of the parameters, observations, and

instruments are, in this order, pva1, pva1_baro_cal (parameters), baro1 (observations) and p0_h0 (instruments). This means that the observation equations gathering information to be processed by means of this model (PVA_baro_U / height_update) must include *instance identifiers* for these objects in the order specified.

Mapping these identifiers to the instance identifiers included in Table 13, it may be seen that an o-record (observation equation) should have a list of instance identifiers like this:

```
27      34              32       51
(pva1) (pva1_baro_cal) (baro1) (p0_h0)
```

or, since there are two different barometers (baro / baro1),

```
27      34              33       52
(pva1) (pva1_baro_cal) (baro1) (p0_h0)
```

In example 23:

- Lines 1 – 2 show an l-record for a “baro / baro1” observation. The identifier is “baro1” (attribute id = “baro1”) and its type “baro” (see the metadata corresponding to this identifier). The “n” attribute states that the instance identifier is 32. Immediately comes the time tag (124.88) the single tag value related to this kind of observation (temperature, 23.44) and then, the unique value in the observation vector, 1023.44. In line 2 the covariance matrix (standard deviations only) is shown, which is 0.3.
- Lines 3- 4 depict a “imu / imu1” observation (id=”imu1”). The instance identifier (n=”41”) comes next and after that the time tag (124,88) is shown. There are no tag values (see the metadata for this kind of observation in the corresponding lines in the navigation metadata file (in example 8) so the next six values (0.01 0.02 0.015 0.32 0.43 9.95) correspond to the observation vector (the measurement). Line 4 includes again a standard deviation-only covariance matrix (1e-3 1e-3 1e-3 1e-2 1e-2 1e-2).
- Line 5 includes another l-record, in this case for a RW_6_PN / imu_bias_pn observation (id=”imu_bias_pn”). Its instance identifier is 17 (attribute “n”), the time tag 124.88, there are no tag values, and the observation vector is (`0.0 0.0 0.0 0.0 0.0 0.0`). There is no covariance matrix (the default one provided in the corresponding metadata object will be used).
- Lines 6 – 8 show three observation equations (o-records) providing data for the three models included in the navigation metadata file shown in example 8. Only the last one (line 8) will be described here, since all these adhere to the same structure defined by o-records. The observation identifier is “height_update” (so its type, see Table 14, is PVA_baro_U). The time tag is again 124.88. This observation equation includes the instance identifiers for two parameters (pva1, pva1_baro_cal), one observation (baro1) and one instrument (p0_h0). The actual values for these instances identifiers are, in this order, 27 34 32 51.
- Note that lines 1 – 8 include both l- and o-records with the same time tag. That is, these define a full epoch.
- Data shown in lines 10 – 17, and 19 - 26 correspond to two new epochs (see the ascending time tags, 124.90 and 124.92 respectively. There is only one difference in these data deserving an extra description. Lines 10 – 11 show again a “baro / baro1” observation like the one included in lines 1 – 2. However, its instance identifier (compare specifically lines 1 and 10) is 33 instead of 32. and the instance identifier for the instrument (baro_p0_h0 / p0_h0) is 52 instead of 51. This is so because this last observation corresponds to a second barometer unit.
- Line 28 includes an ellipsis, showing that more data might appear here.

3.7.5 Instrument files

From the ASTROLABE standpoint, instruments are not random variables; that is, time does not affect instrument data. The approach of ASTROLABE is, therefore, consider instrument data as constant information. Typical examples of instruments constants would be the focal length of a camera or the position of the center of a GNSS antenna.

Of course, instruments constants must be made available to the software responsible for estimating a trajectory. To do so, ASTROLABE introduces the concept of instrument data (not to be confused with

observations; instrument data are not observations). The information related to an instrument is a list of values of variable length and their covariances.

The dimension of this list depends directly on the kind of instrument and is defined by the <i_spec> object in the navigation metadata files (see section 3.6.1). Note that the covariances related to the constants are *a mere indication of their quality, since instruments are not interpreted as random variables.*

Instrument data is stored in (input) instrument files that adhere to the <obs-e_file> format defined in section 3.7.2. Nonetheless, some differences must be taken into account:

- There are no observation equations (o-records) in instrument files. This means that only l-records are present in this kind of files (of course, in external, binary implementations, b-records will still exist. See section 3.7.2.5.3).
- The meaning of the different fields in an l-record must be reinterpreted. Every reference to “observation” in Tables 6 and 8 must be replaced by “instrument” to correctly interpret it.
- The value of the time tag is purely informative. The value for this field must use the same coordinate reference frame that those used for time in the observations files. Additionally, the value itself should reflect the time when the constants defining the instrument were obtained or, in the worse case, a moment in time no later that the first time tag found in the observations file (time of the first epoch).
- The data included in the covariance matrix field is merely informative and should not be used for computational purposes.

The software implementing the read or write operations in these files must be aware of these differences (in this particular case, the absence of o-records).

Since instrument files will usually contain a few l-records, **ASTROLABE does not impose the need to implement the binary format for this kind of files**. This has direct implications on the software components dealing with instrument files, since it will not be necessary to provide with mechanisms to read instrument data in binary format.

The recommended file extension of a <obs-e_file>-based instrument file is [ins].

3.7.5.1 Example

Example 24 below shows an raw instruments text data file. No header file is provided since it would match any of those included for obs-e files – as the one in example 22. The line numbers in the left part of the example are not part of the file itself, but shown for easy cross-referencing. The syntax of l-records and (<l> tags) is defined in section 3.7.2.5.2. Data in example 24 match the definition of the instrument included in the navigation metadata file shown in example 8, more specifically, in lines 111 – 141. No active flag attributes (“s”) have been included, so all records are active.

```
01    <l id="p0_h0"> 51 124.88 1024.01 0.0 </l>
02    <l id="p0_h0"> 52 124.88 1023.99 0.0 </l>
```

Example 24: Instruments file in text (XML) format

In example 24:

- Lines 1 – 2 depict two l-records with data for two instances of barometers (type baro_p0_h0, identifier p0_h0). These are told apart by means of their instance identifiers (51 and 52, see Table 13, column “Instance identifiers”).
- The values of the time tags (124.88 in both l-records) are purely informative. As stated in section 3.7.5, these should contain either a value with the approximate calibration time for the

instruments or, at least, a value not exceeding the time tag of the oldest observation found in observation files. In this example, the value of the time tag for the oldest observation in the observations file (see example 23, line 1) is used.

- The two constants (reference pressure and height) comes after the time tag. Each instrument has its own set of values.

3.7.6 Parameter (state) files

The result of estimating a trajectory is stored in an (output) parameter (state) file. These adhere to the <obs-e_file> format described in section 3.7.2. It must be noted, however, that parameter (state) files do not contain o-records (observation equations) so only l-records are present in these. In binary implementations b-records still exist. See section 3.7.2.5.3.

l-records are defined in Table 6 from a conceptual standpoint. Table 8 offer an implementation-oriented view of the same concept. To correctly understand how parameter (state) l-records are, change all references to “observation” by “parameter (state)” in these tables.

The software implementing the read / write operations for parameter (state) files must be aware that these do not incorporate o-records.

The recommended file extension of a <obs-e_file> based parameter (state) file is [par].

3.7.6.1 Example

Examples 25 and 26 below show a parameter file where the raw data files are stored in text format. These show:

1. the header file (example 25) and
2. the actual raw data stored, as stated above, in a text file (example 26).

The line numbers in the left part of these two examples are not part of the files themselves, but shown for easy cross-referencing. The colours in the line numbers in example 26 are used to tell apart different epochs. The syntax of l-records (<l> tag) is defined in section 3.7.2.5.2. No “s” (active flag) attributes have been ncluded, so all records are active.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03   version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
04     <lineage version="1.0">
05       <id> julycampaign </id>
06   </lineage>
07   <data>
08     <device type="obs-e_file" format="text_file"> output_parameters.par </device>
09   </data>
10 </astrolabe-header_file>
```

Example 25: Parameters file in text format: the header

```
01 <l id="pva1" n="27">      124.88 0.82 0.03 3.23 2.1   3.4   2.0   3.14 0   0.02
02                                  1e-6 1e-6 1e1  0.3   0.3   0.3    1.2 0.5 0.3    </l>
03 <l id="imu1_bias" n="11"> 124.88 0.04 0.02 0.03 0.005 0.004 0.003
04                                  1e-2 1e-2 1e-2 1e-3  1e-3  1e-3                  </l>
05 <l id="baro1_cal" n="34"> 124.88 1.4  0.1                                          </l>
06
07 <l id="pva1" n="27">      124.90 0.82 0.03 3.23 2.1   3.4   2.0   3.14 0   0.02
08                                  1e-6 1e-6 1e1  0.3   0.3   0.3    1.2  0.5 0.3   </l>
09 <l id="imu1_bias" n="11"> 124.90 0.04 0.02 0.03 0.005 0.004 0.003
10                                  1e-2 1e-2 1e-2 1e-3  1e-3  1e-3                  </l>
11 <l id="baro1_cal" n="35"> 124.90  1.41  0.1  </l>
12
13 <l id="pva1" n="27">      124.92 0.82 0.03 3.23 2.1   3.4   2.0   3.14 0   0.02
14                                  1e-6 1e-6 1e1  0.3   0.3   0.3    1.2 0.5 0.3    </l>
15 <l id="imu1_bias" n="11"> 124.92 0.04 0.02 0.03 0.005 0.004 0.003
16                                  1e-2 1e-2 1e-2 1e-3  1e-3  1e-3                  </l>
17 <l id="baro1_cal" n="34"> 124.92 1.39  0.1                                          </l>
```

Example 26: Parameters file in text format: the raw data file

In example 25:

- Line 8, attribute “type” states that this header points to an <obs-e_file>. This is so because parameter files are, in fact, a specific case of observation-events files.
- Line 8, attribute format states the way data is handled: a text file.
- Line 8, <device> value defines, consequently, the name of the raw text data file containing the parameters (output_parameters.par).

According to section 3.7.2.5.1, the data files accompanying the header file shown in example 25 may be split in several fragments. Example 26 shows a complete (non-split, that is, **unique**) data file.

Moreover, and again according to section 3.7.2.5.1, the actual file name of the raw data files must adhere to a naming convention based on the base file name included in the header file; such naming convention states that a numeric suffix whose number of digits may vary between 1 and 9. This means that the unique external text file shown in example might have up to nine different file names, being

`output_parameters_1.par`

the shortest one (only one digit in the numeric suffix) and

`output_parameters_000000001.par`

the longest (9-digit suffix). Any of these names would be correct.

In example 26:

- The l-records contain the values of the adjusted parameters (see Table 14 or example 8) pva / pva1 (instance identifier:27), IMU_bias / imu1_bias (instance identifier: 11) and baro_cal / baro1_cal (instance identifiers: 34 and 35).
- All the l-records shown adhere to the syntax defined in section 3.7.2.5.2. This means that, taking the first l-record shown in lines 1 – 2 as a reference,
  - id= “pval” sets the identifier for the parameter,
  - n=”27” states the parameter instance identifier,
  - 124.88 is the time tag,
  - 0.82 0.03 3.23 2.1 3.4 2.0 3.14 0 0.02 are the adjusted values for the parameter vector and
  - now in line 2, 1e-6 1e-6 1e1 0.3 0.3 0.3 1.2 0.5 0.3 is the covariance matrix (including standard deviations only).

3.7.7 Observation's residuals files

The files containing the residuals for the observations adhere to the <obs-e_file> format defined in section 3.7.2. Nonetheless, these files do not contain o-records (observation equations) so only l-

records are present. In binary implementations b-records still exist. See section 3.7.2.5.3.

l-records are defined in Table 6 from a conceptual standpoint. Table 8 offer an implementation-oriented view of the same concept. To correctly understand how observation's residuals l-records are, change all references to “observation” by “observation's residuals” in these tables.

The software implementing the read / write operations for this kind of files must be aware that these do not incorporate o-records.

The recommended file extension of a <obs-e_file>-based observation's residuals file is [res].

3.7.7.1 Example

Example 27 shows an observation's residuals file. In this case, the header is not shown, but just the single raw text data file (example 25, however, might be a valid header file, just changing the name of the data file in the value of the <device> tag). The line numbers in the left part of the example are not part of the file itself, but are shown for easy cross-referencing. The colours in the line numbers are used to tell apart different epochs. The syntax of l-records (<l> tags) is defined in section 3.7.2.5.2.

```
01    <l id="baro1" n="32">        124.88  0.02
02                                         0.01                                    </l>
03    <l id="imu1" n="41">         124.88  1e-4  2e-4 -1e-4  1e-3  6e-3  2e-3
04                                         1e-3  1e-3  1e-3  1e-2  1e-2  1e-2 </l>
05    <l id="imu1_bias_pn" n="17"> 124.88  4e-3 -2e-3  2e-3 -2e-2  1e-2 -4e-2
06                                         1e-2  1e-2  1e-2  1e-1  1e-1  1e-1 </l>
07
08    <l id="baro1" n="33">        124.90  0.01
09                                         0.01                                    </l>
10    <l id="imu1" n="41">         124.90  2e-4  2e-4  1e-4  2e-3 -4e-3  1e-3
11                                         1e-3  1e-3  1e-3  1e-2  1e-2  1e-2 </l>
12    <l id="imu1_bias_pn" n="17"> 124.90  2e-3  2e-3 -1e-3 -4e-2  2e-2  8e-2
13                                         1e-2  1e-2  1e-2  1e-1  1e-1  1e-1 </l>
14
15    <l id="baro1" n="32">        124.92  0.02
16                                         0.01                                    </l>
17    <l id="imu1" n="41">         124.92  4e-4  5e-4 -2e-4 -3e-3  2e-3 -1e-3
18                                         1e-3  1e-3  1e-3  1e-2  1e-2  1e-2 </l>
19    <l id="imu1_bias_pn" n="17"> 124.92 -5e-3  2e-3 -2e-3  4e-2 -2e-2 -1e-2
20                                         1e-2  1e-2  1e-2  1e-1  1e-1  1e-1 </l>
21
22    …
```

Example 27: Observation's residuals file in text (XML) format

Note that this example shows the residuals that might have been obtained for the observations depicted in example 23. See how the identifiers (id attribute) and instance identifiers of the successive l-records *match* with those found in example 23. The same happens to time tags. Note that there are no observation equation (o-records), since these are never included in observation's residuals <obs-e_file> files.

In this example:

- Lines 1 – 6, 8 – 13 and 15 – 20 contain the l-records for three different epochs.
- Lines 3 – 4, for instance, contain a typical l-record for observations residuals.
- In line 3:
  - The l-record is identified by the opening <l> tag. Its attribute “id” states its identifier (imu1).

- Attribute “n” states that the instance identifier is 41, which is referring to the actual observation for which the residuals have been computed.
- Value 124.88 is the time tag.
- The next 6 values, (1e-4 2e-4 -1e-4 1e-3 6e-3 2e-3), are the observation's residual vector.

- In line 4:
  - The covariance matrix (standard deviations only) of the observation's residual vector is provided (1e-3 1e-3 1e-3 1e-2 1e-2 1e-2).
  - The l-record is closed by means of the tag </l>.

3.7.8 Correlation matrix files: <tr_obs_correlation_Rll>, <tr_states_correlation_Rxx> and <tr_res_correlation_Rvv>

The ASTROLABE FI includes three different correlation matrix (<r_matrix-file>) files. These are the correlation matrix for trajectory observations file (<tr_obs_correlation_Rll>), the correlation matrix for trajectory parameters file (<tr_states_correlation_Rxx>) and the correlation matrix for (observation) residuals file (<tr_res_correlation_Rvv>).

From the format standpoint, all these files are implemented by means of the correlation matrix files (<r_matrix_file>) described in section 3.7.3. Refer to that section for details.

The recommended file extension of a <r-matrix_file> based <tr_obs_correlation_Rll> file is [rll].

The recommended file extension of a <r-matrix_file> based <tr_states_correlation_Rxx> file is [rxx].

The recommended file extension of a <r-matrix_file> based <tr_res_correlation_Rvv> file is [rvv].

3.7.8.1 Example

Examples 28 and 29 below show a parameter correlation matrix file (header plus a single, non-split raw text data file). In the last example, only one r-record is shown, but there should be as many as different epochs in the corresponding observations <obs-e_file> file.

Note that examples for observations / residuals correlation matrix files would be equivalent, so these are not shown here. No “s” (active flag) attributes have been included, so all records are active.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03   version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
04     <lineage version="1.0">
05       <id> julycampaign </id>
06   </lineage>
07   <data>
08     <device type="r-matrix_file" format="text_file"> july_campaign.rll </device>
09   </data>
10 </astrolabe-header_file>
```

Example 28: Header file for a parameters correlation matrix stored in a raw text data file

```
01 <r>
02      124.88
03      0.7
04      0.7 0.7
05      0.9 0.3 0.3
06      0.3 0.9 0.3 0.5
07      0.3 0.3 0.9 0.5 0.5
08      0.4 0.4 0.1 0.2 0.2 0.2
09      0.4 0.4 0.1 0.2 0.2 0.2 0.8
10      0.1 0.1 0.1 0.2 0.2 0.2 0.8 0.7
11      0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.5
12      0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.5 0.1 0.0
13      0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.1 0.1 0.0 0.0
14      0.0 0.0 0.0 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0
15      0.0 0.0 0.0 0.2 0.3 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0
16      0.0 0.0 0.0 0.1 0.1 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
17      0.0 0.0 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
18 </r>
19
20  ...
```

Example 29: Raw text data file containing r-records for parameters correlation matrices

In example 28, the relevant things happen in line 8; there, the kind of data is defined (attribute type, “r-matrix_file”), the storage is selected (attribute format, “text_file”) and the base name of the raw text data file is set (“ parameter_correlations.rxx”).

In example 29:

Only one r-record is shown (lines 1 – 18). However, correlation files include one correlation matrix per epoch (in time-tag ascending order). The ellipsis in line 20 is there to show that other r-records should follow.

- Line 1: the r-record starts by means of the <r> tag.
- Line 2: the value there (124.88) is the time tag.
- Lines 3 – 17: the correlation matrix, written as described in section 3.7.3.3.
- Line 18: the closing tag </r> states that this is the end of the r-record.

3.7.9 The options file: <op_file>

The <nav_file> file may include an options file (See section 3.7.11),

Note that (1) the inclusion of an options file is *optional* and (2) when included, *its format is free* that is, *any kind of options file may be included.*

The rationale behind the inclusion of an options file in the <nav_file> file is to keep a record of the set of options used to compute the output trajectory. Different applications may use different approaches to compute trajectories. This means that some options present in some application's options file may be meaningless for other applications and conversely. This is the reason why the format of the options file to include in a <nav_file> file has been left as a choice.

In this way, the format of the <nav_file> is both *closed* – an options file may be included if needed – and *open* – the format of such file is free – at the same time.

The specific format – and version – of the options file may be indicated in the <nav_directory> file (see section 3.6.2).

3.7.10 The log file: <log_file>

Note that (1) the inclusion of a <log_file> is *optional* and (2) when included, its format is *not* fixed, that

is, *any kind of log file may be included.* When present, it contains the history of the trajectory determination process.

3.7.11 The navigation file: <nav_file>

The navigation <nav_file> persistent serialized object file is the most important file of the ASTROLABE specification. A <nav_file> file contains all the files involved in the estimation of the output trajectory; i.e., observations, instruments, estimated parameters, estimated residuals, metadata and any other ancillary information that might be of interest.

In short, the <nav_file> file contains the information necessary to reconstruct the trajectory. At the implementation level, it is a **compressed –zipped– file**; containing other files. Table 15 summarizes the files that may be contained in a navigation file, where the # column corresponds to the expected number of files of each particular type and where the file extension [any] indicates that any file referred by other files may be included.

The recommended extension for a navigation <nav_file> file is [nf].

3.7.12 Header files and the Communications Interface (CI)

The Communications Interface (see section 4) details how ASTROLABE data (observations, parameters, instruments and correlation matrices) are exchanged between processes in run time using TCP / IP sockets; apparently, no files should be involved in the process of transmitting information between cooperating (producer and consumer) tasks processing ASTROLABE data.

Nonetheless, and for the sake of homogeneity, the header files included in the file interface (see section 3.7.1) include the definition of the parameters required to characterize a connection. See, for instance, example 1 on page 30.

The existence of such header files defining how a socket-based transmission should be parametrized, implies that a software component using ASTROLABE data may *always* behave in the same way thanks to the information stored in the files defining its input and output. Using the information stated in these headers, it will be able to ascertain where actual data is stored: either in text or binary files or, like in the situation described here, “reading” (or “writing”) through a socked connection. Of course, when using sockets, data are stored nowhere, but transmitted over a communications line. Obviously, the ability to “behave the same way” no matter what kind of data support is being used is closely related to the availability of a generic input / output library. ASTROLABE is just the definition of an interface and not a software library; nonetheless, the inclusion of the socket input / output channel as one extra case in the definition of a data channels opens the way to a generic implementation of such library.

File extension	File / serialized object	#	Low-level format	Contents and description
.op	<options_file>	0 or 1	text	An options file defining how the trajectory was computed. Free format. Optional
.nmd	<nav_metadata_file>	>=1	text	Metadata (observation, parameter, instrument and model) files
.obs	<obs-e_file> (observations)	1	text + text / text + binary	Input observations (measurements)
.ins	<obs-e_file> (instruments)	0 or 1	text + text / text + binary	Input instruments Optional.
.par	<obs-e_file> (parameters)	1	text + text / text + binary	Output parameters (states)
.res	<obs-e_file> (residuals)	0 or 1	text + text / text + binary	Residuals (for observations). Optional
.rll	<tr_obs_correlation_Rll>	0, 1 or 3	text + text / text + binary	Correlation matrix for trajectory observations. Optional. Only one file when processing either in forward or backward mode. Three in combined mode.
.rxx	<tr_states_correlation_Rxx>	0, 1 or 3	text + text / text + binary	Correlation matrix for trajectory parameters. Optional. Only one file when processing either in forward or backward mode. Three in combined mode.
.rvv	<tr_res_correlation_Rvv>	0, 1 or 3	text + text / text + binary	Correlation matrix for trajectory residuals. Optional. Only one file when processing either in forward or backward mode. Three in combined mode.
.*any*		>=0	text / binary	Any file referred to by the .nmd file. Optional.
.ndf	<nav-directory_file>	1 or 3	text	List of files included in the process of estimating the output trajectory.
.log	<log_file>	0 or 1	text	A log file –trajectory solution report. Optional.

Table 15: Files available in a navigation serialized object

4 CI – THE COMMUNICATIONS INTERFACE: PROTOCOLS

Chapter 3 defines the file interface (FI) of ASTROLABE. The FI will be used normally in post-processing environments, once all the data have been collected. However, in some situations, observation-event (and related) data may need to be processed in real time; this means that information must be *sent* and *received* by several software modules using a different mechanism, not implying the use of files.

The approach adopted by ASTROLABE for such situations is to define a real time communications channel and protocol to achieve the real time trajectory determination.

In this chapter the communication interface used for this real time mode is defined.

4.1 General overview and approach of the ASTROLABE CI

SOCKETS is the technology selected to implement the communication channel. Future versions of ASTROLABE may include other communications technologies.

A socket is a software endpoint that establishes bidirectional communication between a server programme/process and one or more client programs. The socket binds the server process to a specific port on the device where it runs so any client program anywhere in the network (even in the same device) using a socket bound to the same port can communicate with the server program.

One of the most important characteristics of sockets, and one of the main reasons to select this technology, is that it is present in (virtually) every operating system and may used in (almost) every programming language. This guarantees *general availability.*

Of course, socket libraries may expose slightly different interfaces to developers when used in different operating systems or languages, but the underlying functionality remains the same. This guarantees *interoperability*.

The aforementioned set of reasons was the keystone to select sockets as the most suitable option for the ASTROLABE CI.

The idea of sockets is to define a virtual path between two processes. Then, it is possible to send information (*messages*) through the communication channel. Section 4.2 describes the communication protocol defined by ASTROLABE, that is, the format and meaning of the messages that different components using the ASTROLABE CI may send and receive.

Note however, that software components using the ASTROLABE CI must define the parameters characterizing sockets connections by means of header files. See sections 3.7.1 and 3.7.12 for further details.

4.2 The ASTROLABE communications protocol

Once the underlying communication channel has been selected, it is necessary to define the protocol to use. In other words, the rules to follow in order to communicate sender and receiver processes have to be defined. Without such a protocol, processes would never achieve a correct communication.

Note that in the context of this document, the word “protocol” does not refer to the underlying *TCP / IP protocol* used (normally) by sockets technology, since it is thoroughly described in the specialized literature. On the contrary, here, “protocol” stands for “application protocol”, that is, the set of messages (and their format) used by the different components using the ASTROLABE CI to exchange information.

Nonetheless, it is important to note that thanks to TCP / IP, sockets have the following characteristics:

- The communication process is connection-oriented
- Data transmission is guaranteed, without errors and omissions.
- Data will arrive in the same order that it was transmitted.

The application protocol defines how data is represented. This definition is twofold:

- Syntax: how the messages are, how these are built.
- Semantics: the meaning of each command and the accompanying data.

Several types of messages are used by the application protocol. These are:

- Data records, comprising
  - *Observation* or *measurement* records. This record is used to transmit observations, instruments and parameters, since these data entities are structurally equivalent.
  - *Observation equation* records.
  - *Correlation matrix* record.
- Command records, including
  - *Acknowledgement of reception* records.
  - *End of transmission* (end of data) records.

The protocol defines as well the correct flow of message to be exchanged between the sender and listener (receiver) processes.

4.2.1 Data representation

All data sent or received through the CI interface is structured in *messages*. A message (also known as a *record*) is a stream of bytes of arbitrary, variable length.

All messages have at least a command header defining their type. A data section following the command header is present only in data messages.

Commands are represented by one single character:

- “e”: *End of data (or end of transmission)* command message.
- “I” : *Observation* (*measurement)* data message. Used also for parameter and instrument data.
- “r”: *Correlation matrix* data message.
- “o”: *Observation equation* data message.
- Any character, *even those listed for all other records above*: Acknowledgement of reception command message.

Note the apparent anomaly concerning the value of the command header for *Acknowledgement of reception* command records, since it may be represented by any available character. These records need not to carry any specific information; they play a confirmation (acknowledgement) role in the *flow* of messages travelling between sender and listener processes and may be sent or received in very specific moments, so it is *not* the information they transmit, but the fact that the message *has been received* what really matters.

The data section is included only in data messages – that is, *Observation / Measurement* (including parameters and instruments), *Correlation matrix* and *Observation equation* data messages. The contents of the data sections for these two messages is defined in sections 4.2.2.1 to 4.2.2.3.

All the information sent and received during the communication between the sender and the listener must be encoded to and decoded from the XDR format. **The only exception to this rule is the *Acknowledgment of reception* command message**, since it conveys no information at all.

XDR (External Data Representation) is a standard data serialization format, for uses such as computer network protocols. It allows data to be transferred between different kinds of computer systems. Converting from the local representation to XDR is called encoding. Converting from XDR to the local representation is called decoding. XDR is implemented as a software library of functions which is portable between different operating systems and is also independent of the transport layer.

The need to use XDR to encode / decode the information is explained by the different representations used to store data in different computer architectures. Examples of such differences may be the endiannes (the ordering or sequencing of bytes of a word of digital data in computer memory storage or during transmission) that may be either big-endian or little-endian or the kind of representation used to store floating point data (IEEE double, for instance).

If a normalized data encoding / decoding scheme is not adopted, it is not possible to guarantee that the information sent from some computer will be correctly interpreted by another one, since its architectures may differ. Using the XDR encoding standard avoids this problem.

4.2.2 CI messages

This section describes the contents of the messages used by the CI protocol. Note that, unless the contrary is indicated, all the information conveyed by a message *must be encoded* (and afterwards decoded) *using the XDR standard* (see section 4.2.1.)

The subsections below use several *C / C++ built-in types* to describe the size of the different kinds of items constituting a message. The default sizes of C / C++ types char, int and double assumed in these subsections are 1, 4 and 8 bytes respectively.

Refer to section 3.6.1.1.8 or Table 6 for details on how covariance matrices are represented.

4.2.2.1 Observation or Measurement data message

Item	Description	C / C++ type	Dimension	Values
Command	Kind of event message	char	1	Always an “l” (lower case letter l)
Active flag	Flag stating the status of the whole event message (active or removed)	char	1	“1” (for active events) or “0” (for removed or inactive ones). Note that these values are characters, not the numbers 1 and 0
Length of the identifier	Length in characters of the identifier	int	1	> 0
Identifier	Unique identifier used to tell apart the kind of observation or measurement	char	The amount stated by item “length of the identifier”	Only letters, numbers and other characters that may be used in *file names*
Instance identifier	Unique value to tell apart different instances of observations sharing the same identifier	int	1	Any valid int value
Time tag	Time associated to the whole observation or measurement	double	1	Any valid double value
Overall dimension of the tags, expectations and covariance matrix arrays	Number of total elements, including tags, expectations and their covariance matrix, that constitute the actual observation	int	1	> 0
Tags, expectations and covariance matrix arrays	An array including, *in this order*, the tags, expectations and their covariance matrix. See section 3.6.1.1.8 for details on how to write covariance matrices	double	The amount stated by item “Overall dimension of the auxiliary values, expectations and covariance matrix arrays	Any valid double value for tags and expectations data, double values greater than zero for standard deviations. Double values between -1.0 and 1.0 for correlations

Table 16: Observation (measurement) data message

4.2.2.2 Correlation matrix data message

Item	Description	C / C++ type	Dimension	Values
Command	Kind of event message	char	1	Always an “r” (lower case letter r)
Active flag	Flag stating the status of the whole message (active or removed)	char	1	“1” (for active records) or “0” (for removed or inactive ones). Note that these values are characters, not the numbers 1 and 0
Time tag	Time associated to the whole correlation matrix	double	1	Any valid double value
Dimension of the correlation matrix	Number of elements in the correlation matrix	int	1	> 0
Tags, expectations and covariance matrix arrays	An array including, *the correlation* matrix. See section 3.7.3.3 for details on how to write correlation matrices	double	The amount stated by item “Dimension of the correlation matrix”	Double values between -1.0 and 1.0.

Table 17: Observation (measurement) data message

4.2.2.3 Observation equation data message

Item	Description	C / C++ type	Dimension	Values
Command	Kind of event message	char	1	Always an “o” (lower case letter o.)
Active flag	Flag stating the status of the whole event message (active or removed.)	char	1	“1” (for active events) or “0” (for removed or inactive ones). Note that these values are characters, not the numbers 1 and 0
Length of the observation equation identifier	Length in characters of the observation equation identifier	int	1	> 0
Observation equation identifier (model identifier)	Unique identifier used to tell apart the kind of model to use to relate the different observations involved in the equation	char	The amount stated by item “length of the observation equation identifier”	Only letters, numbers and other characters that may be used in *file names*
Time tag	Time associated to the observation equation	double	1	Any valid double value
Dimension of the identifiers array	Number of elements included in the identifiers array below	int	1	> 0
Identifiers array	An array including the identifiers of the different parameters, observations and instruments (in this order) that take part in the observation equation	int	The amount stated by item “Dimension of the identifiers array”	Any valid int value

Table 18: Observation equation data message

4.2.2.4 End of transmission (end of data) command message

Item	Description	C / C++ type	Dimension	Values
Command	Kind of event message	char	1	Always an “e” (lower case letter e.)

Table 19: End of transmission (end of data) command message

4.2.2.5 Acknowledgement of reception command message

Item	Description	C / C++ type	Dimension	Values
Command	Kind of event message	char	1	Any, even those listed in the remaining messages

Table 20: Acknowledgement of reception command message

This contents of this message **must never** be encoded using the XDR standard, and it must be sent “as is”. As explained above, the single character making the command of this message may be any value.

4.2.3 Full message cycles

The *full message cycle* is the basic procedure defined by ASTROLABE to exchange command or data messages. All information transmitted between processes must use full message cycles to send or receive data.

A full message cycle consists of four steps:

1. The sender sends the command or data message, whatever it is,
2. the listener reads the command or data message.
3. the listener sends the acknowledgement of reception command message and, finally,
4. the sender reads the acknowledgement of reception command message.

Very schematically, a full message transmission cycle may be represented as follows:

```
(1) Sender sends a message
                                                  (2) Listener reads a message (blocking)
                                                  (3) Listener sends the acknowledgement
(4) Sender reads the acknowledgement (blocking)
```

In the former schema, “(blocking)“ means that read actions block the process until a send command is executed by the counterpart.

Another way to describe a full message cycle would be:

1. Each time a sender needs to send a command or data message, it will:
   - send the command or data message itself and
   - immediately will read an acknowledgement of reception command message (blocking).
2. Each time a listener expects a new message, it will
   - read the message it is expecting (blocking) and

- immediately, once the message is received, it will send an acknowledgement of reception command message.

The use of full message cycles as defined above as well as the fact that some of the operations in this cycle are blocking guarantee that both processes are correctly synchronized and that these are notified about the correct transmission of the command or data message.

All information exchanged by processes adhering to the ASTROLABE protocol must be sent / received using full message cycles.

4.2.4 The CI protocol

When exchanging information between processes (sender and listener) data must be sent in several *blocks*. Each block will be used to transmit (receive) information of some specific kind, as instrument data (one block), parameter data (one more block) or observation data (another block), for instance. Each block will be responsible of exchanging a whole dataset of some kind of information; for instance, all the instrument data will be sent in a block, observation data in another one.

The current set of CI data messages makes possible to transmit four different blocks of data (those for instruments, observations, parameters and correlation matrices).

Each block starts and ends a communication *session*. This means that all the steps required to set up the connection between the two counterparts, the actual transmission of data, and closing the aforementioned connection, are the parts integrating a session.

Both sender and listener processes *must agree* on which will be the blocks of data that will be exchanged and in which order these will be transmitted. The point here is that ASTROLABE sets no conditions on how information is exchanged between a producer and a consumer but one: how each individual block is transmitted (see below). This is so because the kind of information to transmit and the order in which is transmitted depends on each application willing to use the ASTROLABE CI.

For instance, an application might need, for whatever reason, to send several blocks of observations in different batches instead of sending all the data at once. Providing that each block is sent according to the protocol defined below, it should not be a problem for ASTROLABE.

Each *block* must be sent / received according to the protocol described in example 30. In this (pseudocode) example,

- Lines with a light blue background represent the specific procedures to set up and close socket connections.
- The loop shows how to send (receive) data using the full message cycles described in section 4.2.3. Note the sequence send / receive / send acknowledgement / receive acknowledgement commands implementing such cycles.
- The end of the loop is decided by the sender upon the availability of data. When data are exhausted, the sender process sends an end of data (end of transmission, see section 4.2.2.4) message to notify the listener that no more information will be sent. This makes the listener to break the loop too.
- Note how even that sending the end of data (end of transmission) message adheres to the full message cycle procedure; that is, the listener acknowledges the reception of such message sending an acknowledgement, which is read by the sender.
- The word *Blocked* shows the moments when either the sender or the listener are stopped (waiting) because receiving a message is a synchronous operation. The word *Released* implies that such waiting is over.

CI – Protocol to send a full data block	
Sender	**Listener**
	Initiate a server socket connection (port), accept connections through this socket. **Blocked**
Initiate a client socket connection (host, port)	
READY_TO_BREAK is now FALSE	
DO	**DO**
	Read a message. **Blocked**

IF *there are still data to send*
Send a message containing the data
ELSE
Send the end-of data command.
READY_TO_BREAK is now TRUE
END IF	

Read message (acknowledgement) **Blocked**	
	Released. (the message has been read) Send the acknowledgement of reception
Released. (the acknowledgement message has been read).	
IF *READY_TO_BREAK* **BREAK THE LOOP**	
	IF *message is end-of.data* **BREAK THE LOOP**
END DO	**END DO**
Close the socket connection	Close the socket connection

Example 30: CI protocol to transmit / receive a full data block

5 ACRONYMS AND INITIALISMS

API	Applications Programming Interface
ASTROLABE	A STandard Representation Of time-tagged LocAtion-Based data for Exchange purposes
CI	Communications Interface
CM	Computer Model
CRF	Coordinate Reference Frame
CRS	Coordinate Reference System
CS	Coordinate System
CTTC	Centre Tecnològic de Telecomunicacions de Catalunya
FI	File Interface
FM	Functional Model
GM	Geomatic Model
GNSS	Global Navigation Satellite System
GPS	Global Positioning System (USA)
ICD	Interface Control Document
IMU	Inertial Measurement Unit
IP	Internet Protocol
ITRF96	International Terrestrial Reference Frame 1996
ITRS	International Terrestrial Reference System
MM	Math Model
RF	Reference Frame
RS	Reference System
SM	Static model
SM	Stochastic Model
SM	Software Model
SW	Software
TCP	Transmission Control Protocol

TM	Technical Model
UTM	Universal Transverse Mercator
XDR	External Data Representation
XML	eXtensible Markup Language

6 DEFINITIONS

Computer Model (CM): a computer programme that simulates an abstract model of a particular system. (In NAVEGA, for instance, the network computer model simulates the [abstract] network geomatic model.)

Coordinate Reference Frame (CRF): the pair that consists of a reference frame and a coordinate system. In the NAVEGA notation, ITRF96 / UTM(31,N) stands for the combination of the ITRF96 geodetic reference frame and the UTM(31,N) map projection.

Coordinate Reference System (CRS): the pair that consists of a reference system and a coordinate system. In the NAVEGA notation, ITRS / UTM(50,S) stands for the combination of the ITRS geodetic reference system and the UTM(50,S) map projection.

Coordinate System (CS): a parametrization of a mathematical space. Or, equivalently, a set of mathematical rules to specify the coordinates of an element of a mathematical space. For a 2 dimensional space well known parametrizations are the cartesian coordinate system and the polar coordinate system. In geomatics, a map projection is a [global or local] coordinate system for the reference geodetic ellipsoid.

Dynamic model: a particular type of mathematical model. More specifically, its is a mathematical model whose functional model includes derivatives of the parameters. In other words, a dynamic model is a stochastic differential equation.

Endiannes: the ordering or sequencing of bytes of a word of digital data in computer memory storage or during transmission. Words may be represented in big-endian or little-endian manner. Big-endian systems store the most-significant byte of a word at the smallest memory address and the least significant byte at the largest. A little-endian system, in contrast, stores the least-significant byte at the smallest address.

External Data Representation (XDR): a standard data serialization format, for uses such as computer network protocols. It allows data to be transferred between different kinds of computer systems. Converting from the local representation to XDR is called encoding. Converting from XDR to the local representation is called decoding. XDR is implemented as a software library of functions which is portable between different operating systems and is also independent of the transport layer.

Functional Model (FM): a [mathematical] equation or system of equations.

Geomatic Model (GM): an extension of a mathematical model that includes coordinate reference frames, units of measurements, thresholds and any other information required to describe the geomatic properties of the mathematical model entities.

Geomatics is the art, the science and the technology of acquiring, storing, processing, delivering and managing of spatially referenced information. Geomatics entails sub-disciplines as geodesy, surveying, positioning and navigation, remote sensing and photogrammetry, cartography, mapping and geographic information systems.

Instrument: a device used to measure. In NAVEGA, an instrument is a software entity which contains the constants that characterize an actual instrument. It is one of the four fundamental modelling NAVEGA classes (instrument, observation, parameter or state and model).

Map Projection (MP): a 2 dimensional coordinate system for a geodetic reference ellipsoid. The UTM family of 120 local Transverse Mercator maps is an example of a set of local map projections.

Mathematical Model (MM): an abstract model that uses mathematical language to describe the behaviour of a system. In the context of this document, a mathematical model is a functional model plus and stochastic model.

Measure. A measure or measurement is a particular type of observation. It is an observation that directly results from the act or process of measuring.

Model: an abstract (or actual) representation of an object or system from a particular viewpoint. In NAVEGA, there are functional, stochastic, mathematical, technical, geomatic, computer and SW models. If not otherwise specified, in the NAVEGA jargon, model refers to one of the four fundamental modelling classes (instrument, observation, parameter and model).

Object-Oriented Programming (OOP) is a programming paradigm that uses abstraction to create models based on the real world. Its main techniques and features are encapsulation, inheritance and polymorphism.

Observable. (noun) A numerical property of a physical system that can be determined by a sequence of physical or mathematical operations. Technically, it is a random variable.

Observation. An observation is one of the values that an observable or random variable may take; i.e., it is a random sample of a random variable. (For example, the various repeated measurements between A and B of a distance meter instrument –the measured distances– are observations and the abstract concept of distance between A and B is the observable.) It is one of the four fundamental modelling NAVEGA classes (instrument, observation, parameter and model).

Parameter: a random variable whose expectation and covariance have to be estimated from known observations, instruments and models. It is one of the four fundamental modelling ASTROLABE classes (instrument, observation, parameter or state, and model).

Random Error. Random errors of an observation or estimated parameter are scattered about the true value and such that their mean, after repeated observations or estimations, tends to zero. Random errors lead to the situation where the mean of many separate observations does not differ significantly from the true value of the observed feature. (Random errors are always present in observations.)

Random Variable: a measurable function from a sample space to the measurable space of possible values of the variable.

Reference Frame (RF): a realization of a reference system. For geodetic reference systems the most common realization is a list of points, whose coordinates are known and whose physical location is well defined, usually through marks or targets. ITRF96 is, for example, a realization of ITRS. A more recent way to realize a reference system is through the orbits of a GNSS satellite constellation, like the GPS.

Reference Model: an abstract model or template for the consistent development of more specificc models.

Reference System (RS): a definition; a set of prescriptions and conventions together with the modelling required to define the origin, axes and other required properties of the base of a space; e.g.

the ITRS of the IERS, or the reference system of photographic camera.

Sensor: a device used to measure. In the NAVEGA context, sensor and instrument mean the same.

Sensor Calibration (SC): Is the determination of the parameters xp of a sensor model f'(l-e; x; xp) = 0 that extends the sensor nominal model f(l-e; x) = 0.

Sensor Systematic Error: Is the difference between a sensor's given nominal model f and a new [higher fidelity] model f'. The new model extends f(l-e; x) = 0 to f'(l-e; x; xp) = 0 where xp are the calibration parameters.

Software Component (SC) is an object written to a specification, offering a predefined service and able to communicate with other components. Software components often take the form of objects or collections of objects.

Software Framework is a software support infrastructure –i.e., a set of software and data items– in which another software project can be organized and developed. A framework may include support programs, code libraries, data, specialized scripting or interface languages, or any other software to help develop and glue together the different components of a software project.

Software Model: an abstract model that describes a computer programme or software system, usually with a special graphical modelling language.

Software Platform is a particular type of software framework which allows other software to run.

State. See parameter.

Static Model: a particular type of mathematical model. More specifically, its is a mathematical model whose functional model does not include derivatives of the parameters. In other words, a static model is a stochastic equation.

Stochastic Model (SM): a mathematical model that describes the probability distribution of a random variable or stochastic process.

Stochastic Process: a collection, discrete or continuous, of random variables associated with a deterministic parameter, usually a time or space coordinate.

Systematic Error: a bias of an observation or of a parameter estimate. Systematic errors lead to the situation where the mean of many separate observations differs significantly from the true value of the observed feature. (A systematic error is not necessarily constant.)

Technical Model (TM): In the NAVEGA context, a technical model is the difference between a geomatic model and a mathematical model.

Toolbox. A NAVEGA toolbox is a set of classes that inherit from the NAVEGA four fundamental modelling abstract [C++] classes instrument, observation, parameter or state and model.

Trajectory. A trajectory is a path of an stochastic process. That is, is one of the many time series realizations of an stochastic process.

7 BIBLIOGRAPHY

[1] Colomina, I., Navarro, J. A., Térmens, A., *"GeoTeX: a general point determination system"*. In Prodeedings of the XVIIth International Congress of the ISPRS (International Society for Photogrammetry and Remote Sensing), 2 – 14 August 1992, Washington DC (USA).

[2] Colomina I., Blázquez, M., Navarro, J. A., Sastre J. *"The need and keys for a new generation network adjustment software"*. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2012.

[3] Colomina, I. *"Sensor Orientation (2): precise trajectory & attitude determination with INS"*. Lecture notes of the International Executive Master in Airborne Photogrammetry and Remote Sensing (2009).

[4] Parés, M. E., Colomina, I. *"On software architercture concepts for a unified, generic and extensible trajectory determination system"*, in Proceedings of the ION-GNSS+, 14 – 18 September 2015, Tampa, Florida (USA).