ASTROLABE

Version 1.0

Building the ASTROLABE library

ABSTRACT

This document describes the prerequisites and procedure to build the ASTROLABE library.

DOCUMENT STATUS

Document title:	Building the ASTROLABE library
Document reference:	Building_the_ASTROLABE_library_1-0_NO
Document file name:	Building the ASTROLABE library_1-0_NO.odt
Nature:	<report (RP) | specification (SP) | **note (NO)** | minutes (MI) | others (OT) >
Status:	<in preparation | checked | approved | **authorized**>
Author(s):	José A. Navarro (CTTC)
Version:	1.0
Last change:	2018-11-16T10:13:00
Distribution:	<team | project | organisation | **public**>
Security classification:	<**public** | internal | confidential | secret>
Number of pages:	14

DOCUMENT CHANGE RECORD

Version	Date	Section(s) affected	Description of change, change request reference or remarks.
1.0	2018-11-16	all	First public deliverable version.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to describe the procedure to build CTTC's ASTROLABE C++ library.

This document does **not** describe ASTROLABE. For an exhaustive, rigorous description of the ASTROLABE data model, please refer to [1]. A gentler introduction to ASTROLABE may be found in [2] and [3]. All these documents should be part of the distribution of ASTROLABE.

1.2 Organization of this document

Section 2 describe the tools that have been used to build the standard distribution of ASTROLABE. These are the tools recommended to new users of this library. In section 3, the structure of ASTROLABE's distribution folder is described; additionally, some recommendations are given in order to use the open source libraries on which ASTROLABE depends. These recommendations suggest a folder hierarchy and a naming convention for such libraries. Section 4 list the dependencies (libraries) on which the ASTROLABE library and the set of tests included depend. A rule of a thumb to safely compile and link new software relying on ASTROLABE is also stated. In section 4 the actual build process is briefly described. Finally, section 5 lists the recommended readings to better understand what ASTROLABE is and what it can do.

2 THE TOOLS

The ASTROLABE C++ library may be built either for Windows or Linux-based operating systems. Obviously, the set of tools required in each of these environments differ.

For **Windows** environments, at least **Microsoft's Visual Studio 2015** must be used. There is no need to use the commercial version of this tool; the code may be compiled without problems using the freely available community edition.

The last version of Visual Studio is, at the moment of writing this document, 2017. No attempts have been made to compile ASTROLABE using this version of the tool, but no problems should arise when attempting to build the library in Visual Studio 2017. However, this is not guaranteed.

For **Linux-based** operating systems, project files for **Eclipse Oxygen** (with **CDT** installed) have been provided. The compiler used is gcc, version 7.3.0 (the one included with the distributions of Linux based on Ubuntu 18.04 LTS).

It should be possible to build the library using any other set of tools or compilers. However, no project files have been delivered besides those for Visual Studio 2015 and Eclipse Oxygen. Providing that the selected compiler is compatible with the **C++ 11 standard** and that the user prepares the project or make files by him(her)self, then it should be possible to compile ASTROLABE without problems.

The ASTROLABE source code is thoroughly commented using **doxygen**. To build the HTML documentation, please install this tool.

3 FOLDER STRUCTURE. NAMING CONVENTIONS

3.1 *The ASTROLABE (delivery) folder*

The contents of the folder containing ASTROLABE is depicted in Figure 1.

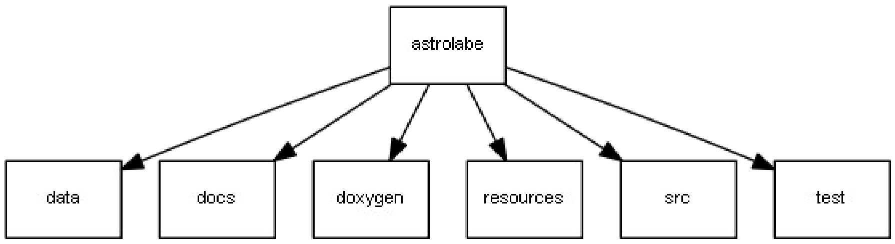

Figure 1: The structure of ASTROLABE's delivery folder.

The contents of these folders and subfolders are:

- **astrolabe**. Parent folder. Contains also the Microsoft Visual Studio project files (Windows builds) as well as those for Eclipse Oxygen (Linux). Shell file (postbuild.bat, postbuild.sh) for both operating systems are also provided, to implement some post-build steps. As shown in Figure 1, it also contains some subfolders:
    - **data**. Contains some XML schemas defining the syntax of several ASTROLABE files. These schemas are the input to some methods in the library, since they are used as the reference to check the correctness of the syntax of some (meta)data user files.
    - **docs**. Here, the documentation related to ASTROLABE (this document, for instance) is stored. The doxygen documentation, however, is not included here (see below).
    - **doxygen**. The doxygen project file (astrolabe_doxyfile.dox) as well as other doxygen pages required to build the HTML documentation of the library are included in this subfolder. Once built the HTML files are stored in subfolder named html that is created here by the doxygen tool.
    - **resources**. This is the place where several images used to create the HTML documentation are kept.
    - **src**. The source code folder.
    - **test**. Source code including several example programs showing how to use the library.

Some of these folders have, on their side, one or more subfolders. For instance, there are as many subfolders in the test folder as examples have been provided: one subfolder per test. The docs subfolder contains an additional subfolder holding all the images inserted in the documents.

Other subfolders of the astrolabe main folder (as headers or binaries) will appear when the library is built.

3.2 *The open source libraries*

ASTROLABE depends on a number of open source libraries. These are:

- Udunits2 (https://www.unidata.ucar.edu/software/udunits/).
- Petr Beneš ziplib (https://bitbucket.org/wbenny/ziplib/wiki/Home).
- Xerces-c 3 (http://xerces.apache.org/xerces-c/).

Each of the libraries in the list above must be built using the mechanisms provided in their distribution packs. This document does not explain how to do it; please refer to the documentation of each library for such purpose. Then, proceed as indicate below.

To build ASTROLABE, the header files (.h, hpp) files for these open source libraries must be present in the system. To build executable programs or DLL / shared libraries relying on ASTROLABE, binary versions of the aforementioned libraries must be provided. Additionally, it is also required to provide with a binary version of an extra library, namely

- expat (https://libexpat.github.io/).

which is not needed at compile time but is a must at link time (since udunits2 relies on expat to perform parsing tasks).

As stated in section 2, project files have been provided to build ASTROLABE in Windows (Microsoft Visual Studio) and Linux (Eclipse Oxygen + CDT). **If the user wishes to use these project files** (which is highly recommended), the open source libraries must be organized in a specific way, so the headers corresponding to these libraries are automatically found. Figure 2 depicts the structure to adhere to.

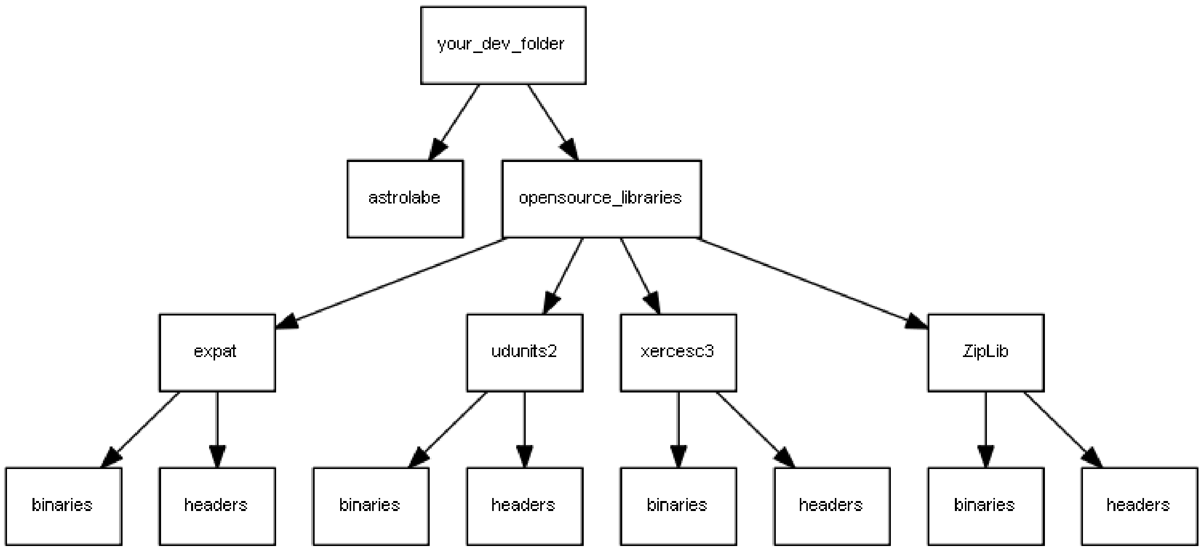

Figure 2: The structure of the folders with the open source libraries.

In Figure 2 it is possible to see that:

- A folder named `your_dev_folder` is used to store both the astrolabe folder depicted in Figure 1 and a new one whose name is `opensource_libraries`. Both `astrolabe` and `opensource_libraries` coexist at the same level. Note that `your_dev_folder` is a name chosen as a placeholder only. Your development folder may be named as you wish.
- The folder `opensource_libraries` is the parent where all the open source libraries must reside (expat, udunits2, xercesc3 and ziplib).
- For each open source library, two subfolders must be created, namely `headers` and `binaries`.

The contents of the headers and binary subfolders for each open source library will be, respectively, (1) the whole set of header (.h, .hpp) and (2) library (.lib, .a) files. This means that after downloading and building each of the aforementioned libraries, the user must copy such header and library files to the respective directories in the structure just depicted.

Note that in the case of the xerces-c 3 and ziplib libraries, it is much easier to copy **the whole source code structure** under the corresponding `headers` folders, since these are pretty complex. For xerces-c, copy the contents of the `src` directory; for ziplib, copy the contents of `source/ziplib`.

The case of the remaining libraries (expat and udunits2) is much simpler; you do not need to copy the whole source structure but a small number of files. For expat, copy `expat.h` and `expat_externals.h`; for udunits2 copy `coverter.h` and `udunits2.h`.

However, if you plan to use your own project files or modify the ones included in the distribution, this organization of the open source libraries may be ignored. Obviously, the locations of the several opensource libraries must be properly set by yourself, pointing to the places where the open source libraries reside.

The `binaries` folders for each open source library are not strictly needed to build the ASTROLABE library itself; it is a library, so no linking takes place! However, the project files for the tests located in the `test` folder of the ASTROLABE distribution (see Figure 1) do rely on them. These produce executables that need to link all the intervening libraries, not only ASTROLABE but also its dependencies.

Again, if you intend to use the supplied project files (either for Visual Studio 2015 or for Eclipse Oxygen) then it is mandatory to copy the binary files with the libraries (either .lib or .a, depending on your operating system) in the respective `binaries` folders. Note that in this case, it is also mandatory to rename the .lib or .a files according to a specific naming convention, which is described in the next section.

3.3 Naming conventions for the libraries

This section should be read only if the user decides to use the project files provided to build the ASTROLABE library and the different tests included in the delivery folder. This is so because these project files rely on a specific naming convention for the library (.lib / .a) files intervening in the process. If the user creates his / her own project files, then it is possible to skip this section (and ignore the naming conventions defined here as well), but the project files must be built then manually.

Note that such conventions affect the ASTROLABE library or the test programs in different ways; in the case of ASTROLABE, these control the name of the output libraries, while when talking about the test programs, the conventions affect the names of the input libraries required to link the executables (that is, ASTROLABE itself as well as the open source ones).

The conventions differ depending on the operating system used to build the software, that is, Windows or Linux.

3.3.1 Naming conventions for Windows development environments

In the case of the Windows operating system (and therefore, Microsoft Visual Studio), three variables are used to build the name of the libraries. These are:

1. the base name of the library (that is, **astrolabe, expat, udunits2, or ZipLib**),
2. whether these are built using a debug or a release configuration and,
3. the number of bits (32 or 64) of the target processor.

The convention itself is defined by the following pattern:

```
<base_name_of_the_library><infix><suffix>.lib
```

where:

- `<base_name_of_the_library>` stands for what it states, that is the base name of the library. See the list above.
- `<infix>` is optional. When present, its value is always `_x64`, denoting a library built for 64-bit systems. 32-bit libraries include no prefix in their names.

- `<suffix>` is mandatory and may take two values:
  - `_mtd`, which implies a release multithreaded DLL build or
  - `_dmtd`, denoting a debug multithreaded DLL build.

According to the conventions just stated, the four possible names of the expat library would be:

Base name	Debug	Release	32-bit	64-bit	Name
expat	×		×		expat_dmtd.lib
expat	×			×	expat_x64_dmtd.lib
expat		×	×		expat_mtd.lib
expat		×		×	expat_x64_mtd.lib

Table 1: The naming convention for libraries (Windows) - an example.

The naming convention above does not apply to the xerces-c library. Note that building this library produces two files (for each combination of release / debug and 32 / 64 bits): the library itself plus a DLL that will be used in run time. The name of the DLL is hardwired in the .lib file, so it is better not to change these names, that is, those produced by the building process.

The names of the libraries (and DLLs) of the xercesc-3 library are, therefore, the following:

Base name	Debug	Release	32-bit	64-bit	Name
xerces-c_3	×		×		xerces-c_3D.lib
xerces-c_3	×			×	xerces-c_3_1D_x64.lib
xerces-c_3		×	×		xerces-c_3.lib
xerces-c_3		×		×	xerces-c_3_1_x64.lib

Table 2: The specific naming convention for the xerces-c_3 library (Windows)

3.3.2 Naming conventions for Linux development environments

The project files for the ASTROLABE library and test programs are provided for 64-bit only. Debug and release configurations are taken into account.

Therefore, the general naming convention is:

```
lib<base_name_of_the_library>_x64[suffix].a
```

where [suffix] is present only in debug configurations and stand for the single letter “d”.

The following table shows the names of all the libraries either produced or used by ASTROLABE according to the naming convention described above.

Base name	Debug	Release	Name
astrolabe	×		libexpat_x64d.a
astrolabe		×	libexpat_x64.a
expat	×		libastrolabe_x64d.a
expat		×	libastrolabe_x64.a
udunits2	×		libudunits2_x64d.a
udunits2		×	libudunits2_x64.a
ZipLib	×		libZipLib_x64d.a
ZipLib		×	libZipLib_x64.a

Table 3: Naming conventions applied to all the libraries involved in ASTROLABE (Linux).

4 DEPENDENCIES

Table 4 below list the dependencies of both the ASTROLABE library as well as the set of tests included in the distribution. It is very important to know these dependencies when a user decides to create his / her own project files to build the software using a different development tool.

Note that dependencies are of two kinds: header & binary ones. Header dependencies imply that the header files for a given library are needed at compile time. Binary dependencies state that the library on which some other component depends must be provided to the linker.

Header and binary dependencies are indicated with the letters “h” and “b” respectively in table 4.

Module	astrolabe	expat	udunits2	xercesc3	ZipLib
astrolabe			h	h	h
test_astrolabe_header_file_parser	h+b	b	b	h+b	
test_bin_to_reverse_bin	h+b				
test_bin_to_txt	h+b				
test_instrument_reader (1)	h+b	b	b	h+b	
test_instrument_writer (1)	h+b				
test_metadata_parser	h+b	b	h+b	h+b	
test_navdir_parser	h+b	b	b	h+b	
test_navfile_manager (2)	h+b	b	b	h+b	h+b
test_observation_reader (1)	h+b	b	b	h+b	
test_observation_writer (1)	h+b				
test_parameter_reader (1)	h+b	b	b	h+b	
test_parameter_writer (1)	h+b				
test_rmatrix_reader (1)	h+b	b	b	h+b	
test_rmatrix_writer (1)	h+b				
test_socket_to_txt (1)	h+b				
test_synthbinsplit	h+b				
test_synthtxt	h+b				
test_txt_to_bin	h+b				
test_txt_to_reverse_txt	h+b				
test_txt_to_socket (1)	h+b				

Table 4: Header and binary dependencies for all the modules in the delivery folder.

Notes to table 4 above:

1. In Windows environments only, the library ws2:32.lib must also be input to the linker. Its name does not change depending on the configuration (debug or release) or the number of bits (32 / 64) selected.
2. In Windows environments only, the ZipLib library is compiled as a set of independent libraries. Therefore, when using ZipLib, a set of extra libraries must be input to the linker. Their base names are: bzip2, lzma and zlib. These base names must be mutated to adhere to the naming conventions described in section 3.3.1 (so, for instance, the 32-bit, debug version of bzip2 should be named bzip2_dmtd.lib)

It is possible, however, to ignore table 4 above and provide always with the same set of directories to include headers and libraries. This apply not only to the tests in that table but also to the new software created by a user relying on the ASTROLABE library.

To guarantee that all the header and binary folders are set, specify the paths to the `headers` and `binaries` for the following libraries:

- **All operating systems:** astrolabe, xercesc_3, udunits2, expat, ZipLib.

In the same line, provide the following list of libraries (with their names conveniently modified according to the respective naming conventions) to the linker:

- **Windows:** astrolabe, xercesc_3, udunits2, expat, ZipLib, bzip2, lzma, zlib, ws2_32
- **Linux**: astrolabe, xercesc_3, udunits2, expat, ZipLib

In Windows environments the order in which libraries are input to the linker does not matter. In **Linux**, however (at least when using Eclipse) the order is relevant, so **use the ordering stated in the list above.**

5 BUILDING. POSTBUILDING

If the user uses the project files included in the distribution folder of ASTROLABE, building the library or the tests is a simple task: open the corresponding development environment for the platform has to be built (that is, Visual Studio in Windows, Eclipse in Linux) and build the project.

Of course, the libraries on which ASTROLABE depends must have been compiled before, and their headers and binary files arranged in the way described in section 3.2.

The project files included in the original distribution (either for Windows or Linux operating systems) execute a postbuild step once the library has been compiled. The postbuild step takes care of:

- Creating (only the first time) a new folder, located at the same level than the `src` folder, named `headers`.
- Creating (only the first time, same place) a new folder named `binaries`.
- Copying all the .h and .hpp files in the `src` folder to `headers`.
- Copying the libraries to `binaries`.

In this way, the software built by the user relying on ASTROLABE may be used just referring to these headers and binaries folders (in the same way that ASTROLABE makes use of the open source libraries (see section 3.2 and Figure 2).

To build the HTML documentation of the library, please run the `doxygen` tool using the file `ASTROLABE_doxyfile.dox` located in the `doxygen` folder of the distribution as project file. This will build the documentation in a new folder, named `html`, which will be located inside the `doxygen` directory.

6 BIBLIOGRAPHY

[1] Parés, M. E., Navarro, J. A., Colomina, I. 2017. “ASTROLABE version 1.0 Interface Control Document”.

[2] Navarro, J.A.; Parés, M.E.; Colomina, I.; Blázquez, M. "A generic, extensible data model for trajectory determination systems", in Proceedings of the 3rd International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM 2017), pp. 17-25, 27-28 April 2017, Porto (Portugal). doi:10.5220/0006258400170025

[3] Navarro, J.A.; Parés, M.E.; Colomina, I.; Blázquez, M. “ASTROLABE: A Rigorous, Geodetic-Oriented Data Model for Trajectory Determination Systems”. ISPRS Int. J. Geo-Inf. 2017, 6, 98.