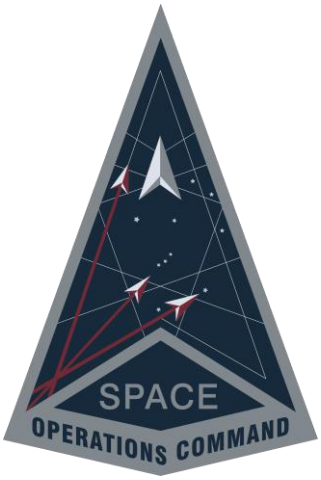
HQ Space Operations Command (SpOC)
DCG-T/S9I
Astrodynamics Standards Engineering Group

Astrodynamics Standards



Programmers & Users Guide

Version 9.4

May 2024

Contents

Figures

Tables

1. Introduction

This document is intended to serve as a programmer's guide for integrating with the Astrodynamics Standards (AstroStds) libraries. The AstroStds libraries were built in the form of shared libraries which can be accessed by various programming languages in 64-bit formats. Within this document, the term "Library" is used to refer to either Dynamic Linked Libraries (DLL or *.dll files) for Windows, Shared Object libraries (*.so files) for Linux, or Dynamic Libraries (*.dylib files) for Mac.

1.1. What are the Astrodynamics Standards Libraries?

The AstroStds is a suite of software libraries which were built based on the algorithms used in the 18th Space Control Squadron (18 SPCS). This software was created to enable interoperability between the 18 SPCS and users of its space catalog products consisting primarily of Two-Line Element sets (TLEs) and Vector Covariance Messages (VCMs). The AstroStds provide the only validated method to utilize the TLEs and VCMs to obtain a satellite position and velocity at a particular point in time. Some of the benefits of using the AstroStds include ensuring that accuracy is not lost, reduce system development costs and procurement cycle times, reduce software maintenance costs, and facilitate the development of new capabilities.

The AstroStds algorithms are designed to be compatible with systems and astrodynamics algorithms implemented into space operations and used by Warfighters and Analysts. The AstroStds are also used to Verify and Validate (V&V) equivalent algorithms of these operational space-domain systems such as those that run at the 18 SPCS at Vandenberg AFB, and other operational locations critical to the National defense.

1.2. Background

The AstroStds have a long history which can still be seen in their present implementation. Over the years they have been modernized while still maintaining the capabilities of previous versions. There are many different factors which have shaped the current AstroStds and knowledge of them will increase understanding of why various decisions were made in their creation.

1.2.1. Historic Motivation for Standardized Astrodynamics Algorithms

Numerous astrodynamics tools and algorithms have been developed by the United States Space Force (USSF) HQ Space Operations Command (SpOC), formerly Air Force Space Command (AFSPC), to fulfill its Space Surveillance mission. These algorithms were tested, reliable, and compatible with the data gathered and distributed for use by the Space Surveillance Network (SSN). However, as network users updated old systems and brought new ones online, site-specific software was often developed. Many times, it fell on the site contractor to create or obtain software that would process and produce SSN data. Since it was often difficult to obtain the desired software from AFSPC, and because there was no clear guidance, as new systems were built contractors would develop a solution. The result was a proliferation of redundant algorithms that were all maintained separately at increased cost and risk to AFSPC. A viable solution was to release existing software to qualified users. The problem was that SSN-compatible software was typically part of an integrated system (like SPADOC or 427M). The individual modules needed by a user of SSN products could not be easily separated from the rest of the system for distribution.

USSF SpOC and USAF AFSPC Role in Astrodynamics Standards

1.2.2. In 1991 USSPACECOM/J3 tasked HQ AFSPC/DO to be the technical lead for AstroStds. During the early 90's AFSPC undertook the task of extracting desired algorithms from the larger programs in the Space Defense Operations Center (SPADOC) and officially recognized them as the AFSPC Astrodynamics Standards. They were created to provide tested, trusted, and centrally maintained SSN compatible code to ensure interoperability, and to avoid spending money reinventing and separately maintaining code whose capability was already available. These algorithms are currently released as no cost GOTS (Government off the shelf) products to qualified DOD contractors and Government users for installation in programs that use or provide data to the SSN. With the creation of the United States Space Force (USSF), the maintenance and distribution of the software has been transferred to USSF SpOC.

Evolution of the Astrodynamics Standards Library

1.2.3. The AstroStds were originally a collection of separate executable programs which had defined input and output file formats and performed a specific predetermined task. Starting in 2008, the AstroStds interfaces have been transformed to better meet the needs of the large and diverse user base. They were completely redesigned to meet a new vision of how to access them. The most important change is that the AstroStds are no longer required to pass structure (user-defined) data types. With just simple (primitive) data types, the interfaces are guaranteed to work with any programming language that supports DLL, SO, or DyLib invocation. Another big change is that the AstroStds are modularized. These modularized Libraries are much easier for the programmer to learn and integrate them into the users' current or new systems. The modularized libraries also offer benefits to the USSF developers, including ease of distribution, ease of releasing upgrades/changes, and reduced maintenance costs. The updated library has been designed to work, if desired, without the use of any input text files. Now there is much greater flexibility which allows the programmer to directly access many of the intermediate calculations and receive the output data directly versus a text file. A multitude of astrodynamics calculations, time and coordinate system conversions, and other utilities are available to programmers. While many changes were made, AstroStds kept its algorithms written in Fortran. AstroStds libraries are heavy in mathematical formulas, which is appropriately implemented in a programming language tailored to math.

1.2.4. Nevertheless, users of the AstroStds libraries can use the libraries regardless what language they use.

Utilization of Shared Libraries

A shared library of functions (procedures or subroutines) are compiled, linked, and stored separately from the applications using them. Because of this separation, libraries can be shared and replaced easily without recompiling the applications that call them, an illustration of this is shown in **Figure 1**. The AstroStds delivery provides wrappers and drivers in a variety of languages. The term "**Wrapper(s)**" pertain to the source files in each of the supported languages that provide the Application Programming Interface (API) to the functions in the libraries. All functions available in the libraries are provided in the **Wrappers**. The term "**Driver**" or "**Driver Example**" pertains to example code that uses the library function calls.

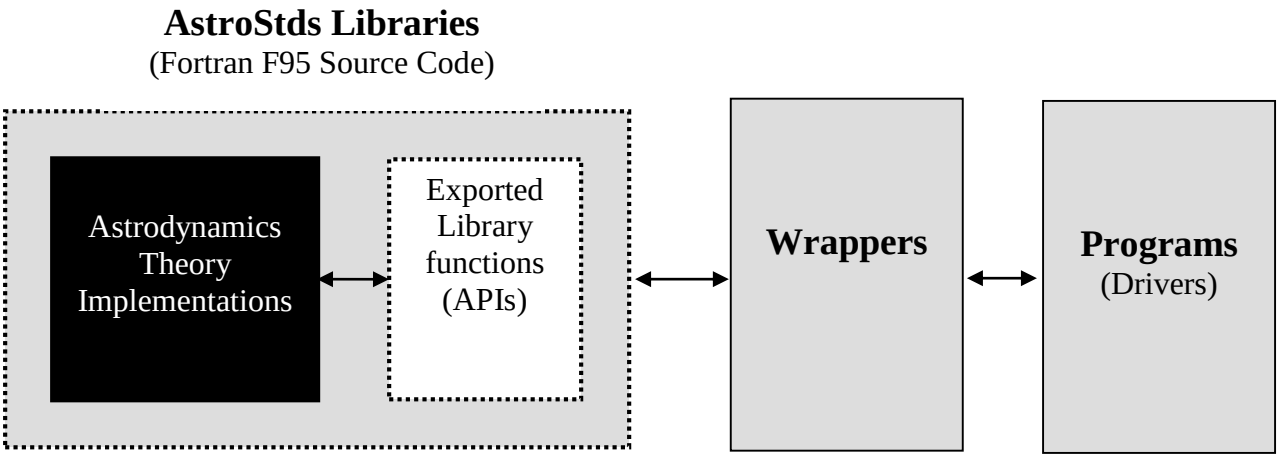


Figure 1. Astrodynamics Standards Library Structure

Some of the advantages of using Libraries:

- Saves memory: Only one copy of a library is stored in memory at any given time. If multiple applications need code from the same library, all of them can use the same shared copy of that code. In addition, the AstroStds libraries are designed to be thread safe.
- Easier upgrades: Whenever the library is upgraded and its interfaces do not change, the applications that use the AstroStds can simply replace the old version with the new without being modified or recompiled.
- Support for multiple programming languages: The libraries can be used by any programming language that supports DLL, SO, or DyLib integration.
- Separates the interface from the implementation: Makes upgrades/changes to the library transparent to the users.
- Flexibility: Designed to work easily with both GUI (Graphical User Interface) and non-GUI applications and to facilitate the development of new capabilities.

1.3. Platform and Language Availability

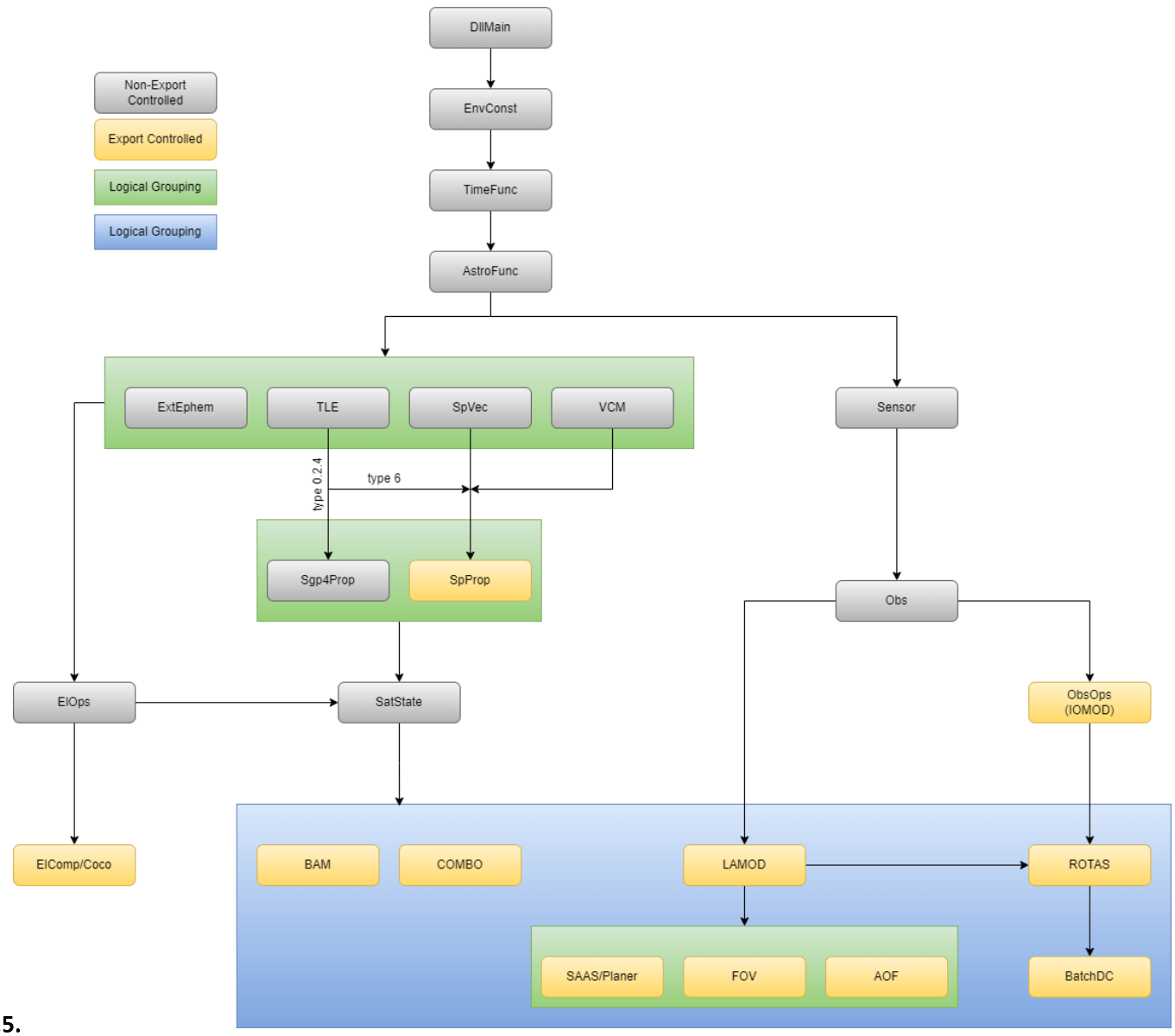
The AstroStds libraries are available on 64-bit platforms for both Windows, Linux, and Mac operating systems (see **Appendix C - Mac OS**). Version releases include **wrappers** and **drivers** to support a variety of customer/user environments which are shown in **Table 1**. These programming languages were tested and proven to work with the AstroStds libraries. Due to resource constraints, some languages have limited driver examples. The most complete set of drivers is provided for the C language, and they can be consulted to better understand the steps necessary to utilize a specific library with some adaptations for your desired language. All languages have a driver for SGP4 which demonstrates how to call libraries from that specific language.

Table 1: Programming Languages with Example Code

Language	Windows	Linux/Mac	Sample Drivers
C/C++	Yes	Yes	16
C#	Yes	No	3
Fortran	Yes	Yes	3
Go	Yes	Yes	2
Java (JNA & JNI)	Yes	Yes	14
Julia	Yes	Yes	2
Matlab	Yes	Yes	4
Octave	Yes	Yes	2
Python	Yes	Yes	4
Rust	No	Yes	3
Swift	Yes	Yes	5
Tcl	Yes	Yes	2
VB.Net	Yes	No	3

1.4. Architecture

AstroStds currently has 24 libraries and about 639 functions and subroutines available for users to call. They are designed to provide users with flexibility and ease of use. Each library is designed to handle a specific task. However, each library may have certain dependencies on other libraries which the user needs to understand and follow. As shown in **Figure 2**, the arrows depict the dependency structure of the AstroStds. For example, the users need to load and initialize EnvConst before using TimeFunc, and to use EnvConst, the user needs to load and initialize the DllMain.



1.5.

Figure 2. Astrodynamic Standards Library Dependency Diagram

Record Storage & Identification

When a record (satellite, sensor, or observation data) is loaded into the AstroStds, it is entered into the corresponding binary tree and can be accessed by all the dependent libraries. It is given a unique identifier (key) to allow for easy access throughout the libraries. **Table 2** shows the record types.

Table 2: Record Identification

Identifier	Input Record Types	Data Type	Record Library Location(s)
satKey	TLE, VCM, SpVector, ExtEphem	64-bit Integer (19 digits)	ExtEphem, TLE, SpVec, VCM
senKey	Sensor Record	Integer	Sensor
obsKey	Observation Record	64-bit Integer (19 digits)	Obs

Selecting the SatKey Mode

Any applications that work with ELSETs (TLEs, SpVecs, or VCMs) need to call SetElsetKeyMode() method/function from the DllMain API to select the appropriate mode. (This is not applicable to External Ephemeris). Multithreading on the same satKey is not allowed by design, instead each thread should utilize a unique satKey for its processing even if using an identical ELSET.

1.5.

Table 3: SatKey Loading Guide

Return Value	Intepretation	ELSET KeyModes
satKey > 0	Record loaded successfully	NODUP, DMA
satKey = 0	Duplication detected, not loaded	NODUP
satKey < 0	Invalid input data, not loaded	NODUP, DMA

1.5.1.1. ELSET_KEYMODE_NODUP (default mode)

AstroStds will check to see if there is an identical ELSET (same ELSET type with the identical orbital values and epoch time) already loaded in memory. If there isn't, a new unique satKey is generated and returned indicating this ELSET is loaded successfully in memory.

1.5.1.2. ELSET_KEYMODE_DMA (Direct Memory Access)

In this mode, a satKey which associates with the actual memory address allocated for the ELSET is returned. There is no check to see if this ELSET is already loaded. This means that one satellite record can have more than one satKey. Using this mode is highly recommended in multi-thread applications to prevent multiple threads from accessing the same satKey at the same time which could cause the AstroStds to crash or provide incorrect results.

Description of Astrodynamics Standards Library Components

The following subsections are listed alphabetically by each of the 24 AstroStds libraries for quick reference. The precise library name is shown before the hyphen ("-") in the section heading and corresponds directly to the library names used in **Figure 2**. If the component is subject to the International Traffic in Arms Regulations (ITAR), which also refers to "Export Control", then "(ITAR)" will be listed after the library name description.

3.

AOF - Area Overflight

AOF computes when overhead satellites have potential visibility to a geographic location or area on the surface of the Earth. Visibility is defined as a nominal Field of View (FOV), defined by a user-specified half-angle around the satellite's sub-point intersecting the defined points or areas on the surface.

3.1

AstroFunc - Astrodynamics Functions

AstroFunc provides functions for converting between orbital elements and for transforming between coordinate systems. It also provides a variety of other useful astrodynamics utility functions.

3.2

BAM – Breakup Analysis Module (ITAR)

3.3

BAM uses the last good element set of the parent object along with user-selected best-quality element sets of a few of the breakup pieces. The post-breakup piece element sets are propagated backward in time, and the pre-breakup parent element set is propagated forward in time, to identify the "pinch-point" where the piece and parent element sets converge positionally in time. This convergence point corresponds to the time the breakup of the parent most likely occurred.

3.4.

BatchDC - Batch Differential Correction (DC) Orbit Determination (ITAR)

BatchDC performs a least-squares batch differential correction of orbital elements using empirical tracking data (sensor observations). It updates either SGP4 Keplerian elements (18th SPCS TLE) or SP state vectors (from an 18th SPCS VCM) using the appropriate propagator theory.

3.5

Combo - Computation of Miss Between Orbits (ITAR)

Combo computes close approaches between satellite orbits based on user-specified screening volume, exclusion volume, and warning and alert thresholds expressed in either standoff radius or asset-centered UVW (radial, in-track, and cross-track) miss-distance criteria. Precomputed SGP4, SGP4-XP, or SP-based ephemerides are used in the evaluation. ***Note:*** SGP4 GP-based Combo results are appropriate for general understanding of the frequency of approaches between objects and coarse assessment of expected miss distances between objects, but *not* appropriate for collision-avoidance decisions. Only high-accuracy SP ephemerides that include propagated covariance resulting in computed Probability of Collision P_c should be used for collision-avoidance decisions.

3.6.

DllMain - Main Library

DllMain is the root component of the AstroStds libraries. It creates a static global data set containing all of the data pointers and function pointers needed by the rest of the libraries. This global data set also allows the libraries to communicate with each other. This library also provides logging capability.

ElComp - Element Comparison, including Computation of Co-Orbital (ITAR)

ElComp combines capabilities of Computation of Coplanar Orbits (COCO) into the Element Comparison library (ElComp), with the ElOps library dependency.

The COCO algorithm calculates the difference in orbital plane between a pair of orbits defined by element sets. COCO includes evaluation of relative nodal rates between the two objects and reports when (how many days hence) the two object's ascending nodes will align.

3.7.

The ElComp algorithm is used to determine the degree to which two orbits are Same, Close, Similar, or None. Determination of these conditions is by user-specified comparison thresholds for the primary orbit versus the secondary for which differences are evaluated in inclination, right ascension of ascending node, perigee height, eccentricity, orbital period, and argument of perigee.

ElOps - Element Operations

ElOps includes various orbital element operations that do not require a propagator. This includes estimating the decay time, the approximate longitude east subpoint, whether it is a GEO satellite, and the satellite's orbital elements and other parameters.

3.8.

EnvConst - Environmental Constants Utility

3.9. EnvConst is used for loading and manipulating various Earth constants, FK data, geomagnetic model, and specifying the shape of the earth.

3.10. ExtEphem - External Ephemeris

ExtEphem allows the user to use their own ephemeris data as a substitute for data propagated by the SGP4/SP propagators. It can load external ephemerides from many supported file formats (ASCII and/or binary). It uses its own interpolator to interpolate ephemeris to the desired time.

3.11.

FOV - Field of View (ITAR)

FOV determines times in which orbiting satellites fly through a ground-based observer's conical field of view. The field of view can be defined by a constant azimuth and elevation boresight, a constant right ascension and declination boresight, or as a line-of-site to an orbiting satellite. The input orbit descriptions may be either a SGP4 TLE, a SP VCM, or an externally generated ephemeris file.

3.12.

Lamod - Look Angle Module (ITAR)

3.13. Lamod computes sensor (ground-based or space-based) viewing opportunities ("look angles") for Earth-centered satellite orbits. The input orbit description may be either a SGP4 TLE, a SP Vector, a SP VCM, or an externally generated ephemeris file.

Obs – Observations

Obs is used for loading and manipulating observations including observations in B3 format, external TTY ASCII format, or Comma-Separated Value (CSV) format.

ObsOps - Observation Operations (ITAR)

ObsOps includes the Initial Orbit Module (IOMOD) capability to compute an initial set of orbital elements (18th SPCS TLE) from as few as three sensor observations, or four observations to include solution for the drag model parameter. ObsOps also allows users to manipulate observations and derive useful information such as Latitude and Longitude from Right Ascension, Declination and Height. Notably, the
3.14. ObsOps library now has the capability to compute satellite range through a triangulation technique using two simultaneous angles-only tracks.

Rotas - Report Association, Observation/Element-Set Association (ITAR)

Rotas compares sensor observations against satellite orbital data using the same algorithms used by the 18th SPCS. It computes observation residuals, compares the residuals against delta-height, delta-in-track,
3.15. and delta-beta residuals and assigns the ASTAT (association status) value to be either 1, 2, 3, or 4. It also provides residuals in terms of the observables for telescopes (Right Ascension, Declination) or radars (Azimuth, Elevation, Range, Range-Rate). A delta-position (vector magnitude distance) is also calculated using an approximated range for telescope observations.

SAAS - Space Attack Assessment Software (ITAR)

3.16. For a specified launch location, profile, and maximum kill altitude, SAAS computes the kill ring and identifies all object's orbits that will penetrate the kill ring (entry or exit or both and the corresponding satellite numbers and times). Identified satellites only include those that orbit less than or equal to the maximum altitude of the launch capability for a specified launch forecast period of interest. SAAS will also compute the launch orbital plane, missile time of flight, auto cone size, and planar intersection time and associated data.

3.17.

SatState - Satellite State

SatState is used to have a single call to retrieve propagated position and other information about a satellite regardless of the input satellite data type (TLE, VCM, SP Vector, and Ephemeris). It extracts derived information such as semi-major axis, minimum perigee and maximum apogee, or computation of Earth-Centered Inertial (ECI) positions from a ground sensor location (either Latitude Longitude and
3.18. Height, or Earth-Centered Rotating (ECR) position). SatState will also compare the position and velocity of two satellite orbits at a requested time.

3.19. Sensor – Sensor Processing

Sensor is used for loading sensors, defining sigma, bias and the sensor limits that define the coverage type, Field of Regard and Field of View.

Sgp4Prop - SGP4 Propagator, extended to include SGP4-XP Capabilities

Sgp4Prop includes analytic propagation methods based on General Perturbations (GP) theory. Sgp4Prop is used for generating ephemerides for satellites in Earth-centered orbits. It is the appropriate means for propagating orbital state using input TLEs for sensor-tasking purposes. SGP4 perturbations account for non-sphericity of Earth, Sun and Moon perturbative accelerations, and atmospheric drag according to Jacchia 1970 static density tables. The new Extended Perturbations (XP) version of SGP4 is appropriate for applications that require SP-level accuracy. This new version of SGP4 is referred to as “SGP4-XP”.

SpProp - SP Propagator (ITAR)

SpProp uses a Special Perturbations (SP) theory to generate ephemerides from ECI state vectors produced by the BatchDC orbit determination library or VCMs. SP theory includes perturbations accelerations due to Sun and Moon and other third-bodies (planets); non-sphericity of Earth including up to 70x70 zonals and tesserials in the EGM-96 geopotential model; dynamic calibration of the

3.20. atmosphere (DCA); Earth-tides and ocean tides; and other techniques that improve orbit determination and prediction accuracy.

SpVec - SP Vector

SpVec is used for loading SP Vector, which are an alternative to the VCM. SP Vectors do not contain covariance and are the preferred method to load vectors which do not originate from AstroStds or the

3.21. 18 SPCS.

TimeFunc – Time Functions

TimeFunc manages various time types and conversions among various time types such as Time Atomic

3.22. International (TAI), Universal Time Corrected, TAI minus UTC offset (leap second), UT1 Rate, and determination of Greenwich hour angle, also known as Theta Greenwich.

TLE - Two-Line Element Set Processing

3.23. TLE is used for loading operational Two-Line Element sets (TLEs) and CSV formatted orbital elements.

3.24. VCM - Vector Covariance Message Processing

The VCM library is used for parsing and loading information from VCMs, which are produced by the BatchDC library, or the 18 SPCS.

4. What's Provided in the Delivery Package

The exact content within your delivery varies based upon what has been specified in your AstroStds user agreement; however, all deliveries follow the same basic structure regardless of which subsets of software you may receive. You will find four main folders: Astrodynamics Standards Library (Lib), documentation (Documentation), sample code for various programming languages (SampleCode), basic testing code (UnitTests), some useful utilities (Utilities), and a method to verify that the software is working correctly (Verify). There's also a README.txt file with information on how to get started.

4.1 Astrodynamic Standards Library "Lib" Folder

This folder contains separate folders for each operating system (Windows, Linux, and Mac). The Linux folders contain the libraries compiled with "ifort" and "gfortran". The Windows folder contains libraries for the Windows operating systems. The MacOS folder contains libraries for the Intel based Mac, compiled in Intel oneAPI and GNU gfortran compilers. The MacOS folder also contains libraries for the M1 based Mac, compiled with GNU gfortran.

All "Lib" sub-folders contain the "SGP4_Open_License.txt" file which is required to run SGP4. It must be kept with the same directory as the libraries.

4.2 "Documentation" Folder

The documents folder contains the supporting documentation for the library.

4.2.1 APIDocs

This contains a complete listing of all the functions available within each library. It shows all the parameters and return types for each function. To access it, double click on "index.html" (or open a browser to this file). This has been created using javadoc but is applicable to all software languages. While the HTML documents were designed to accommodate all programming languages as much as possible, there are remnants that may not pertain to a specific language, such as "class" or "classes".

4.2.2 Librarydocuments

This contains PDF documents for each library which provide a description, prerequisites for use, input and output file formats. For some libraries, this also includes additional details regarding definitions and theoretical basis of the calculations.

4.2.3 Verdict

This includes the documentation for the Verdict program which allows users to verify that their output data matches what is expected.

4.2.4 Individual Files

In addition to the folders, some individual files may also be provided at this top-level. These include "Release Notes" for the most recent version which details exactly what has been changed from the previous version. It also contains diagrams of the V7 and V8 architectures which provides valuable information on the dependency of the libraries.

"SampleCode" Folder

There is source code for C drivers that emulate all the legacy AstroStds executable functionality which take the legacy input files as input and create standard output files. These can be used as examples of how to utilize each individual library. Each example comes with "runExample.bat" and "runExample.sh" scripts, which should run out-of-the-box and will prompt you for anything that is missing.

4.3.

Many programming languages can use the AstroStds API, and there is sample code provided for many of the most common languages but this is not inclusive of all the languages that can access the libraries. Due to resource constraints, many languages only have examples for the SGP4 and SP libraries (if provided). If you will be using a library that does not have examples provided in your language, it is recommended that you consult the examples that are provided to understand the language-specific methods for calling the library and then consult the example for that application provided by the C drivers to better understand how to call that application.

There is also a README.md file under each language. It may contain specific instructions for that programming language.

4.4. "UnitTests" Folder

The UnitTests folder contains a Microsoft Visual Studio project, including a "solution" file (AFSPC.AstroStds.UnitTests.sln) that can be used as sample code for developing your project.

4.5. "Verify" Folder

The Verify folder contains the resources necessary to validate that your delivery is producing the expected results for your particular operating system and processor type. This is done using VERDICT (VERsatile Difference and Compare Tool), which is a numerical data file comparison utility. Refer to VERDICT.pdf located in the "/Documentation/verdict" folder for additional information. This includes pre-made test cases which can be used as examples of how to correctly compose input card types.

5. Input and Output Parameters Conventions

- Parameters to the API calls are either inputs or outputs. Whether the parameter is an input or output is documented in the APIDocs. The output parameters are how results are returned by the API call (i.e.. pass-by-reference). How pass-by-reference is implemented differs by language, which is shown in the "Driver" examples
- The default values of certain input options are placed in square brackets "[]". For example, when calling SpSetApCtrlAll in SpProp, “RUN_MODE = 0, [1]” means 1 is the default value for RUN_MODE.

5.1. Named Constants

Named Constants are used extensively in the AstroStds library to refer to an integer that is used to get/set a field or as an array index. **Table 4** shows naming conventions for these constants. It should be noted that Astro Standards uses zero-based arrays. Languages with one-based arrays should take this into account by adding one to the named constant or starting the indexing at zero if possible.

Table 4: Named Constants Naming Convention

Name	Description
XF_xxx	Named constants for a field get/set
XA_xxx	Named constants for array index
XS_xxx	Named constants for string
XAI_xxx	Named constants for array index of an integer number array
XAR_xxx	Named constants for array index of a real number array

5.1.1. Example 1: Field Named Constant

Assume a Named Constant called XF_EXAMPLE is defined to be 256. Instead of using 256, you can use the name "XF_EXAMPLE". This is highly encouraged because oftentimes, the code may change (e.g.. XF_EXAMPLE in a new release is 512). If you used the named constant, your code would not need to change to accommodate changes made in the libraries.

5.1.2. Example 2: Field Named Constant

An example of an API call that is using a Named Constant for a field get/set, is for the following API:

```
public static int TleGetField(long satKey, int xf_Tle, byte[] valueStr)
```

The inputs to the TLEGetField API are a satellite key (satKey) and an integer to specify which field should be returned in the output byte array (valueStr). The values that can be used for xf_Tle are specified by the Named Constants with the naming convention xf_TLE_yyy. The user can either pass the integer or the Named Constant when calling the API.

```
Int errCode = TleGetField( satKey, XF_TLE_SATNUM, valueStr)
Int errCode = TleGetField( satKey, 1, valueStr)
```

Example 3: Named Constant for Array Index

An example of using Named Constants to access an array index, is for the following API:

public static void PosVelToKep(double[] pos, double[] vel, double[] metricKep)

The values of the metricKep array can be accessed using the XA_KEP_yyy Named Constants which are shown below:

5.1.3.

```
Keplerian elements order in an array (zero-based array)
XA_KEP_A    =  0
XA_KEP_E    =  1
XA_KEP_INCLI =  2
XA_KEP_MA   =  3
XA_KEP_NODE  =  4
XA_KEP_OMEGA =  5
```

If you wanted to store the inclination as incli, the following command would be used to access inclination in the ‘2’ position:

```
Double inclination = metricKep[XA_KEP_INCLI]
```

Using the Astrodynamics Standards Library

In order for your application to use the AstroStds libraries, you first need to setup an environment variable, so your application can find the location of the libraries:

- PATH: Add the directory to the AstroStds libraries to this environment variable on Windows
- LD_LIBRARY_PATH: Add the directory to the AstroStds libraries to this environment variable on Linux or Mac (see **Appendix C** for additional comments for the Mac)
- May require both of these environment variables for some programming languages.

6.

Different languages have different ways of calling library functions. Besides a library file itself, there are several associated files the users need to have for their programs to work. These are usually referred as "wrappers". These wrapper files are created by SpOC and provided to the users. For example, if the programmer wanted to use AstroFunc.dll, here is the list of wrapper files he would need to include in his program:

- C/C++: AstroFuncDll.h and AstroFuncDll.c
- C#/VB.Net: AstroFuncWrapper.cs (VB.Net users can compile the file to create a DLL and then add a reference to it in their programs.)
- Fortran: AstroFuncDll.f90 and AstroFunc.lib
- Java: JnaAstroFunc.java, JniAstroFunc.java
- Julia: AstroFuncDll.jl
- Matlab: M_AstroFuncDll.h
- Python: AstroFuncWrapper.py

There is a **README.md** file in each of the specific languages under the SampleCode directory. This file will contain notes on using AstroStds for that specific language.

6.1

32-bit vs. 64-bit Operating System Considerations

The 32-bit version of AstroStds has been discontinued as of 9.0. AstroStds will generally run on platforms running a 64-bit version of the same operating system. Care must be taken to ensure that the program and all the necessary runtime components are of the same architecture. Reminder: Be sure to use the correct 64-bit compiler option when compiling the driver programs with AstroStds.

6.2.

Calling the AstroStds Library from C/C++

SpOC provides the following files for use with C/C++:

- DllUtils.h and DllUtils.c: Provide template routines to load/unload the AstroStds libraries. They also provide macros to make function pointer assigning easier.
- Associated header ".h" files: Define the path to the AstroStds Libraries, declare function prototypes and function pointers for the AstroStds libraries, and define constant variables.
- Associated source code ".c" files: Load/unload specific AstroStds libraries and assign function pointers to the appropriate library functions.

Notes:

C and C++ use NULL-terminated strings. The strings used in the AstroStds library do not follow this convention. You must add the NULL character to strings passed to your program by the AstroStds library routines. The NULL character will not be added automatically.

When using the Driver Example sample projects in Microsoft Visual Studio, it is important to set the active platform to match the libraries being used (64-bit versions). This is done by right clicking the solution name in the Solution Explorer window, selecting *Properties* from the popup menu, selecting *Configuration Properties* from the left side of the Properties window and using the *Configuration Manager* to set the active platform.

For Linux, depending on the system and path settings, the libifcoremd.dll and libmmd.dll files provided in the /Lib directory may need to be copied to the application directory.

Two example solutions for C included in the DriverExamples folder:

- Sgp4Prop_Simple – demonstrates propagation of a hard-coded TLE. Output is displayed on the console, so it can be run from the command line with the output piped to a file using a command. For example*:*
  Sgp4Prop_Simple.exe > test_output.txt
- Sgp4Prop – demonstrates propagation using an input file specified in the command line parameters. See the source code comments for an explanation of the complete parameters which include specifying the input file and output file.

Note that the executable filenames in the examples above (e.g. *Sgp4Prop.exe*) may vary depending on the setup in Visual Studio.

6.3. Calling the AstroStds Library from C# or VB.Net

SpOC provides the following files for use with C#/VB.Net:

- Associated wrapper ".cs" files: Create static classes that wrap the libraries, define paths to the libraries, define static variables, and implement DllImports/public static extern to map .Net wrapper functions to the AstroStds library functions.
- C# developers can include the associated wrapper ".cs" files directly in their projects. VB.Net developers need to compile the associated wrapper ".cs" files to .Net libraries first, and then refer to them within their application.

Notes:

- C#/VB.Net developers do not need to worry about loading/initializing the required libraries. The wrappers have been implemented to handle these tasks automatically in their static constructors.

When using the Driver Example projects in Microsoft Visual Studio, it is important to set the active platform to match the libraries being used. This is done by right-clicking the solution name in the Solution Explorer window, selecting *Properties* from the popup menu, selecting *Configuration Properties* from the left side of the Properties window and using the *Configuration Manager* to set the active platform.

Two example solutions for both C# and VB.Net included in the DriverExamples folder:

- Sgp4Prop_Simple– demonstrates propagation of a hard-coded TLE. Output is displayed on the console.
- Sgp4Prop – uses command-line arguments for the input and output file names. To run the samples within Visual Studio, the command-line arguments can be specified using the menu *Project > Properties > Debug > Command line arguments* and should appear like "*rel14.inp sgp4prop_out*". See the comments in the sample code for a full explanation of the input parameters.

Note that the Microsoft Visual Studio projects contain references to *AFSPC.AstroStds.Utilities.dll* and *AFSPC.AstroStds.Wrappers.dll* in the /Lib directory of the examples. The path to the AstroStds libraries is specified in the property *AFSPC.AstroStds.Utilities.Utility.ASTROSTDDLLPATH*. The source code for the Utilities and Wrappers are included in the */LibSource* folder so the two libraries can be recompiled if necessary to change the value in *ASTROSTDDLLPATH*. If they are recompiled, the developer should ensure that the new versions are properly referenced in the Microsoft Visual Studio project.

6.4 Calling the AstroStds Library from Java

Java wrappers provided by AstroStds use either Java Native Access (JNA) or Java Native Interface (JNI). JNA uses pure Java, so it is easier to code. Thus, the Java wrappers were first implemented in JNA. JNI requires significant complexity. It requires the addition of C code, where the syntax had to be tightly coupled with Java with specific JNI method calls to process parameters. JNA uses a technology called "Reflection", which does marshalling/un-marshalling and string lookups behind-the-scene, so although it is easier to code, it causes the process to be noticeably slower than JNI. So, regardless of the added complexity of JNI, it was added to the AstroStds libraries to satisfy users' performance needs. To the users, the only difference they would see is the method names start with either "Jna" or "Jni". All coding complexity is hidden in the libraries. All JNA functionality remains the same for legacy support.

- SpOC provides the following files for use with Java:
    - Associated JNA wrapper classes: JnaXXX.java, where XXX is the Library names previously mentioned. For example JnaAof, JnaAstroFunc, JnaExtEphem, JnaSgp4Prop, etc.
    - Associated JNI wrapper classes: JniXXX.java, where XXX, again, pertains to the Library names.
- Notes:
    - JNA:
        - JNA requires users to use specific "xxxByReference" classes for all output parameters. E.g.. DoubleByReference for output parameters of type "double".
        - JNA requires "jna.jar" and "platform.jar" files to be in the Classpath. These files are provided by SpOC in the distribution.
        - Methods in JNA that require 2-dimensional arrays must be converted to 1-dimensional arrays. There are utility methods in the Utility class that can convert 1-to-2 or 2-to-1 dimensional arrays. The Utility class is provided in the distribution.
        - JNA methods do not provide any validation checks. Users will need to ensure all parameters are not "null" and all arrays are, at least, the minimum size.
    - JNI:
        - JNI requires users to create an array for **output** parameters. E.g... For a method that has a single "double" returned, you would need to create a "double" instance like this: "double[] xxx = new double [1];", then pass "xxx".

- JNI methods provide validation for “null” and checks to make sure arrays are the correct size or larger. An error message is provided.
- Unlike JNA, JNI handles 2-dimensional arrays as 2-dimensional arrays.
- No additional libraries (e.g... jna.jar) are required.

Calling the AstroStds Library from Matlab

SpOC provides the following files for use with Matlab:

- Associated header "M_xxx.h" files: Declare function prototypes of the AstroStds library.

6.5.

Notes:

- The AstroStds library is written such that no Mex-Files are necessary within Matlab.
- Useful knowledge and Matlab commands: header ".h" file (this is from C language), loadlibrary, libfunctionsview, libpointer, calllib, get, unloadlibrary

Calling the AstroStds Library from Fortran

6.6. SpOC provides the following files for use with Fortran:

- Associated interface "xxxDll.f90" files: Define explicit interfaces for the AstroStds libraries.
- Associated library ".lib" files: These are import libraries which need to be included when you build your project.

Notes:

You need to have a Fortran compiler that complies with Fortran 2003 (even though the file extensions are .f90) or later to work with the AstroStds library.

If Microsoft Visual Studio is being used to build your project, you need to include the associated “.lib” files. The following steps will accomplish this.

1. Go to Project/Properties/Linker/General. In Additional Library Directories option, type the path to the folder that contains the ".lib" files.
2. Go to Project/Properties/Linker/Input. In Additional Dependencies option, type all the associated ".lib" file names in there. For example, enter "DllMain.lib EnvConst.lib AstroFunc.lib TimeFunc.lib

6.7. Tle.lib SpVec.lib Vcm.lib SpProp.lib" to include all the libraries needed to implement the Sp propagator

Calling the AstroStds Library from Python

SpOC provides the following files for use with Python:

- AstroUtils.py: Provides utility functions useful when calling the AstroStds library functions from Python.
- Wrapper files: These are named using the form <library name>Wrapper.py. For example, the wrapper for DllMain is called DllMainWrapper.py. The wrapper files provide a function to load the library in question and retrieve an object that can be used to access it. They also set things up so that parameters are checked before function calls are made in order to avoid problems resulting from incorrect parameter types.

Notes:

The AstroStds libraries are called from Python via the ctypes module. Because of this, there are a few things to keep in mind.

1. You must ensure that parameters passed to functions in the AstroStds library are of the correct ctype. For instance, if the function you are calling requires an int, you must pass a c_int and not a standard Python int.
2. When passing values by reference, use the byref() function. This function is provided by the ctypes module. Arrays and strings will automatically be passed by reference; do not use byref() with them.
3. C strings can be created using the create_string_buffer() function, provided by the ctypes module. Make sure to allocate one extra character for the NULL character, just as you would in C.
4. C arrays can be created in the proper way using the createCArray() function. This function is located in the AstroUtils.py file, provided by SpOC. It is recommended that you use this function instead of trying to create and initialize C arrays yourself.
5. Values are sometimes converted between ctypes data types and Python data types in surprising ways. For example, if you get a value from a ctypes array, you will get a value using a Python data type back even though the array contains values that are stored using ctypes data types. Conversions were made explicit and obvious to avoid this confusion.

Calling the Astro Standards from Ada2012

6.8 Ada is a supported language of Astro Standards as of v9.4. Ada prefers the Interface.C module to interface with C libraries. However, the native types seem to work just as well, so they were used instead. Astro Standards also included a service called AstroAdaTypes that will need to be included in the main program. These are for defining the 1D and 2D arrays and the pointers. Array sizes didn't need to be defined in the wrapper, but the wrappers include them anyways in the comments below the function definition. Arrays of unknown size at compile time must have the size defined at runtime. This is done with pointers. The gnat compiler complains that there is no equivalent C type for Ada pointers, but seems to run anyway.

The runExample.sh script uses gnatmake to compile the program. If any of the locations of the wrappers or services change, this line will need to be updated. As with the other languages, LD_LIBRARY_PATH on Linux/Mac or PATH on Windows will need to be set before running. The script also uses this to find the location of the Astro Standards libraries when running gnatmake.

Appendix A – Glossary of Common Terms used in AstroStds

Term	Definition
API	Application Programming Interface
Apogee (Q)	The point of the orbit farthest from the primary focus (Earth).
Argument of perigee (ω)	An angle measured in the orbit plane in the direction of motion from the ascending node to perigee.
ASTAT	Association Status for an observation compared to a satellite record.
Classical Elements	A set of six orbital elements needed to define an orbit which include: eccentricity, inclination (deg), mean anomaly (deg), mean motion (revs/day), Right Ascension of the Ascending Node (deg), Argument of perigee (deg)
Drivers	Example code that executes functions in the AstroStds libraries.
DTG	Date Time Group
DLL	Dynamic Link Library
Eccentricity (e)	The shape of the obit with a perfect circle having a value of 0.0.
ECI	Earth Centered Inertial Coordinate System defined by the True Equator Mean Equinox of Date (Epoch)
Epoch	A point in time that defines a vector, TLE, VCM, etc.
Equinoctial Elements (Eqnx)	A set of six orbital elements needed to define an orbit which are specially defined to avoid singularities when eccentricity is zero or the orbit is equatorial: Af, Ag, Chi, L: mean longitude (deg), N: mean motion (revs/day), Psi
FK4	Basic Fourth Fundamental Catalog
FK5	Basic Fifth Fundamental Catalog
GEOFolder	Directory containing the files which define the various geopotentials that are used in the AstroStds. Also called GEO_FILES
J2000	Earth Centered Inertial Coordinate system defined by the Mean Equator and Mean Equinox at Noon on Jan 1, 2000:

Keplerian Elements (Kep)	A set of six orbital elements needed to define an orbit which include:
Inclination (i)	The angle between the satellite's orbit plane and the earth's equator plane.
IOMOD	(Initial Orbit Generation) Computes an initial set of orbital elements (18 SPCS TLE) from three sensor observations. DLL name is ObsOps.
LLH	Latitude, Longitude, Height
Longitude East Subpoint	
Mean Anomaly (M)	A uniformly varying angle that identifies the position of the satellite in the orbit.
Mean Motion (n)	The average angular speed of an obit measured over an entire orbit. This is typically expressed in radians per minute or revolutions per day.
Mean Longitude (L)	The calculation of L avoids division by sin(i) and sin(e) by using a combination of the Mean Anomaly, Argument of Perigee, and Right Ascension of Ascending Node.
Right Ascension of the Ascending Node (Ω)[1]	Measured counterclockwise in the equator plane from the direction of the vernal equinox to the point where the satellite makes its south-to-north crossing of the equator. Also referred to as RAAN.
Perigee (q)	The point of the orbit closest to the primary focus (Earth).
Semi-major Axis (a)	Defines the size of the orbit, half the distance between the apogee and perigee.
SGP4	(Simplified General Perturbations #4) Is an analytic method based on a general perturbation theory for generating ephemerides for satellites in earth-centered orbits. It is the proper means for correctly propagating an 18th SPCS TLE.
SGP4-XP	(Simplified General Perturbations #4 - Extended) A new orbit prediction model which replaces the 50-year-old SGP4 model. Prediction accuracy compared to the SGP4 model is 1 to 2 orders of magnitude improvement with comparable run times. It provides additional and improved perturbations modeling including solar radiation effects, atmospheric drag includes effect of solar activity, and extends applicability to earth bound orbits reaching cis-lunar altitudes.
SP and SPEPH	(Special Perturbations) Is a high accuracy special perturbations theory which uses numerical integration to calculate

[1] TP008 incorrectly uses the terminology "Longitude of the Ascending node" for this definition.

	ephemerides for satellites in earth-centered orbits. It is the proper means for correctly propagating an 18 SPCS VCM.
SSN	Space Surveillance Network detects, tracks, catalogs and identifies artificial objects orbiting Earth, i.e. active/inactive satellites, spent rocket bodies, or fragmentation debris.
TEME	True Equator Mean Equinox of Date
TLE (Two Line Element)	Data format that contains orbital elements of an Earth-orbiting object for a given point in time, the *epoch*. TLEs are used in the SGP4 propagator to predict the position and velocity of the object at any point in the past or future can be estimated to some accuracy. The format uses two lines of 80-column ASCII text to store the data. The TLE format is a *de facto* standard for distribution of an Earth-orbiting object's orbital elements. TLEs can describe the trajectories only of earth orbiting objects.
True Anomaly	Measured from the principle focus in the direction of motion from perigee to the satellite's position vector.
UVW Coordinate System	An inertial coordinate system defined by placing the origin at the current satellite location and defining the axes as: U: Aligned with the radial vector V: Cross-product of W and U W: Aligned with the angular momentum vector In this definition, the UVW system is non-rotating compared to the ECI J2000 coordinate system
VCM (Vector Covariance Message)	A data format that contains information about earth-orbiting objects. The message consists of 28 or more lines that describe such parameters as position and velocity, geopotential models, solar flux values, and epoch and nutation values. VCMs are used in the SP propagator to predict the position and velocity of the object at any point in the past or future.
Wrappers	The source files in each of the supported languages that provide the Application Programming Interface (API) to the functions in the libraries. All functions available in the libraries are provided in the Wrappers.
18 SPCS	The 18th Space Control Squadron (SPCS) at Vandenberg Air Force Base, CA, is located 160 miles northwest of Los Angeles, CA. Its mission is to "Defend freedom of action in space for the Joint Force, multinational partners and humanity". The squadron is tasked with providing 24/7 support to the space surveillance network (SSN), maintaining the space catalog and managing United States Space Command's (USSPACECOM)

space situational awareness (SSA) sharing program to United States, foreign government, and commercial entities. The squadron also conducts advanced analysis, sensor optimization, conjunction assessment, human spaceflight support, reentry/break-up assessment, and launch analysis. In addition, 18 SPCS also oversees 18 SPCS Detachment 1, located in Dahlgren, VA.

Appendix B - Coordinate Systems, Reference Frames, Time

8. Coordinate Systems

8.1

The AstroStds support a variety of coordinate systems which include TEME of Date (ECI), MEME of Noon on January 1st of 2000 (J2000), Earth Fixed Geocentric (EFG), Earth-Centered Rotating (ECR), Latitude-Longitude-Height (LLH), Satellite-Centered Radial Fixed (UVW) and Satellite-Centered Velocity Fixed (PTW). The abbreviations and terminology within the AstroStds have historically been used for the space surveillance mission but may have more precise or different names in the civil space world. The AstroFunc library performs coordinate transformations between these coordinate systems.

There are three different categories of coordinate systems used:

1. Earth-Centered Inertial are fixed against the star background and include J2000 and ECI.
2. Earth-Centered Rotating are fixed according to the Earth's rotation and include EFG, ECR & LLH.
3. Satellite fixed coordinate systems are centered on the satellite and include UVW and PTW.

A particular celestial equator and vernal equinox direction are the basis for all ECI frames used by AstroStds. Several types of ECI frames are used depending on the application. An ECI frame has a strictly fixed orientation relative to the stars if its orientation for a particular date/time is specified. The ECI frame most closely related to the EFG frame is the true equator/mean equinox frame ECI TEME for the same date/time as the associated EFG frame. The only difference between this frame and the EFG frame is the orientation of the Greenwich meridian about the CEP axis at the time of interest. This orientation is specified by the Greenwich Mean Sidereal Time (GMST) which is a function of UT1 time.

A diagram showing the relationship between the various Earth-Centered Inertial and Rotating frames is shown in Figure 3. A more detailed explanation of these coordinate systems and how transformations are performed between them is available in the SpProp.doc found in the "Documentation\librarydocuments" folder of your delivery.

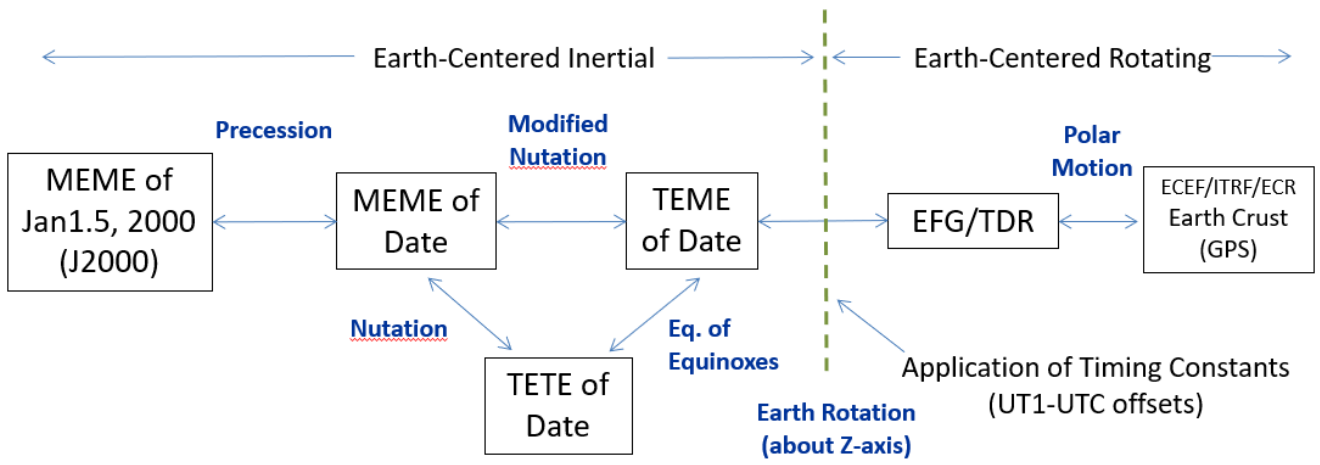


Figure 3: Relationship between Earth-Centered Inertial and Rotating Coordinate Frames

ECI

8.1.1. Within the AstroStds, ECI refers to Earth-Centered Inertial coordinate system referenced to the True Equator Mean Equinox (TEME) of Date (epoch). Generally, an ECI frame can have various references; however, within the AstroStds ECI is always in reference to TEME of Date. ECI is the default output coordinate frame within the AstroStds library. Within the library, it is also referred to simply as "XYZ" and is the coordinate frame when the output is defined simply as position (POS) and/or velocity (VEL).

8.1.2.

ECR (ITR)

The International Terrestrial Reference (ITR) frame is fixed relative to Earth's crust and has a single unambiguous definition. This non-inertial frame is also called the Earth-Centered Earth-Fixed (ECEF) frame since it is fixed to Earth's crust. Other jargon for this frame includes the term, Earth-Centered Rotating (ECR) since this frame rotates in inertial space along with Earth. While ITR is the more well-

8.1.3. known terminology, due ECR historically being used within the SSN it is the terminology used within the AstroStds.

EFG

Earth-Fixed Greenwich (EFG) differs from the ECR/ITR frame in that EFG does not incorporate polar motion. Differences between the EFG and ECR/ITR frames are small, on the order of 15 meters or less. The EFG frame is similar to the ECR/ITR frame except that the pole of this frame is the Celestial Ephemeris Pole (CEP) at the time of interest, instead of the ITR Pole, also known as the Conventional International Origin (CIO). The CEP is normal to the true equator and represents the instantaneous rotational pole after diurnal oscillations are removed. The CEP drifts slowly relative to the ITR frame. This phenomenon, known as polar wander (or polar motion), determines the orientation of the EFG frame relative to the ITR frame. The z-axis of the EFG coordinate frame points from the geocenter to the CEP, the x-axis points from the geocenter to the intersection of the Greenwich meridian and Earth's true equator. The y-axis completes this right-hand coordinate frame.

J2000

This is defined as the Earth-Centered Inertial coordinate system referenced to the Mean Equator and Mean Equinox of January 1.5 of 2000 (Noon of January 1st, 2000). This is used in the Owner/Operator ephemeris files that are supplied to the 18th SPCS for use as input for SuperCOMBO in the collision avoidance processing.

8.1.4. PTW

Position and velocity covariance is represented in a satellite-centered frame. In the PTW frame, T ("tangential") is the unit vector in the velocity direction, W ("cross-track") is the unit vector normal to the satellite inertial orbit plane, and P is the unit vector which completes the right-handed coordinate system. (P is only coincident with the radial direction when the satellite orbit is circular or for an elliptical orbit at apogee and perigee.)

8.1.5. UVW

Position and velocity covariance is represented in a satellite-centered frame. In the UVW frame, U ("radial") is the unit vector in the radial direction, W ("cross-track") is the unit vector normal to the satellite inertial orbit plane, and V ("in-track") is the unit vector which completes the right-handed coordinate system. (Despite the "in-track" descriptor, V is only coincident with the velocity when the satellite orbit is circular or for an elliptical orbit at apogee and perigee.)

8.1.6. 8.2. Fundamental Star Catalogs FK4 & FK5

Fundamental Catalogs define the star background and allow identification of the position of the vernal equinox. They are used to perform the transformation between Earth-Centered-Rotating and Earth-Centered-Inertial coordinate frames.

8.2.1. Fourth Fundamental Catalog (FK4)

The Fourth Fundamental Catalog (FK4) was an attempt to establish a fundamental system of stellar positions and proper motions for the 1950 equinox using as much position data as were available. Most of the stars are brighter than 7.0 mag. To increase the star density in some regions of the sky, positions and proper motions for additional stars were established on the same system. This catalog contains seven data files, six for different equinoxes (1950, 1955, 1960, 1965, 1970, and 1975) and one for the supplemental stars. The 1950 and 1975 files contain the complete FK4 catalog (1535 stars); the others contain only 52 polar stars. (Fricke and Kopff, Fricke 1963 1963)

8.2.2. Fifth Fundamental Catalog (FK5)

The Basic Fifth Fundamental Catalogue (FK5) Part I provides improved mean positions and proper motions for the 1535 classical fundamental stars that had been included in the FK3 and FK4 catalogs. The machine version of the catalog contains the positions and proper motions of the Basic FK5 stars for the epochs and equinoxes J2000.0 and B1950.0, the mean epochs of individual observed right ascensions and declinations used to determine the final positions, the mean errors of the final positions and proper motions for the reported epochs, and ancillary data such as magnitudes, spectral types, parallaxes, radial velocities, and cross identifications to other catalog designations.

The Basic FK5 is the successor to the FK4 (Fricke & Kopff 1963) and contains the 1535 classical fundamental stars used to define the latter system. It represents a revision of the FK4 and results from the determination of systematic and individual corrections to the mean positions and proper motions of the FK4, the elimination of the error in the FK4 equinox, and the introduction of the IAU (1976) system of astronomical constants. About 300 catalogs providing star positions obtained from throughout the world are included in the FK5. This document should be used only to supplement the information contained in the source reference. The latter should be consulted for more detailed information regarding the motivation for construction of the FK5, the determination of its equator and equinox, the expressions for general precession, a discussion of the FK5 system, systematic differences between the FK4 and FK5, the transformation of observational catalogs to the FK5 system and to the reference system J2000.0, and more thorough descriptions of the data contained in the FK5 catalog. (Fricke , Schwan and Lederle 1988)

Time

8.3. Date and Time Abbreviations – The following abbreviations are used in this documentation:

DS50: days since 1950. This date starts from 31/12/1949 00:00:00.000 (which means DS50 = 1.0 is 1/1/1950 00:00:00.00)
UTC or Ds50UTC: days since 1950, UTC (Coordinated Universal Time)
UT1 or Ds50UT1: days since 1950, UT1 (Universal Time)
TAI or Ds50TAI: days since 1950, TAI (International Atomic Time)

DTG: date time group string formats:
DTG20: YYYY/DDD HHMM SS.SSS
DTG19: YYYYMonDDHHMMSS.SSS
DTG17: YYYY/DDD.DDDDDDDD
DTG15: YYDDDHHMMSS.SSS
DTG13: YYDDD.DDDDDDDD

Appendix C - Mac OS

AstroStds provides libraries for *Apple's Mac Operating System (OS)* for both Intel and M1 based chipsets.

9. Setting Library Path

By default, the Mac will look for shared libraries in the following directories:

9.1.

- /usr/local/lib
- /usr/lib

You can either have your system administrator put the libraries in one of those locations or run the following command:

install_name_tool -add_rpath "<path to AstroStds libraries>" "<path to your executable>"

The "**install_name_tool**" is an application that comes with the Mac. Use the "-add_rpath" argument to add the path to the AstroStds libraries, then pass the path to your actual executable. If you created an LD_LIBRARY_PATH environment variable, you can also use that in the command like this:

install_name_tool -add_path "$LD_LIBRARY_PATH" "<path to your executable>"

Note: The "-add_path" argument, will cause the tool to add the path reference to your executable, so if you have multiple paths and you only want one, you need to use the "-delete_rpath" argument to remove the unwanted paths. Also, this path applies to the executable, so if you rebuild the executable the install_name_tool needs to be run again.

9.2.

Code Signing

The astrostandards libraries have to be "code-signed" in order to use them as part of a distributable application, for the Mac.
(https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html)

The burden of "code signing" is on the developer who's integrating the Astro Standards libraries.

When downloading Sgp4Prop_xx.zip from space-track.org, the *.dylib files come unsigned. You can see this using the "**codesign**" tool:

For example:

codesign -v libdllmain.dylib
libdllmain.dylib: code object is not signed at all

To sign the code, run the “codesign” tool, for example:
sudo codesign --force --verify --verbose --sign "Developer ID Application: John Doe (ABCDEFGHIJ)" libdllmain.dylib

Where “John Doe” is your developer name and “ABCDEFGHIJ” is your developer identifier code.

Quarantine Extended Attribute

9.3. The "**quarantine**" extended attribute (or bit or flag) may prevent a library from being executed at runtime. This bit is added by the MacOS automatically for files downloaded from the internet (e.g.. space-track.org). This bit tells the OS to show the user a warning, advising them that the file was downloaded from the internet and to be careful of running malicious code. This bit is cleared when the user clicks "OK" and is never set again. For dynamic libraries (i.e... *dylib files), it’s different because the parent application is calling the library, not the user directly. The parent application will fail to run if one of the dependent libraries has the quarantine bit set. So, if the file was obtained from the internet, the quarantine bit will need to be cleared before using the application.

To check if a file has extended attributes, run "**ls -l**" and you will see a "**@**" symbol after the permissions:

```
-rwxrwxrwx@ 1 user  user    266888 Jan 17 11:19 libastrofunc.dylib
-rwxrwxrwx@ 1 user  user     99272 Jan 17 11:19 libdllmain.dylib
-rwxrwxrwx@ 1 user  user     95576 Jan 17 11:19 libelops.dylib
```

To see the extended attributes, run "**ls -l@ ***"

```
-rwxrwxrwx@ 1 user user    266888 Jan 17 11:19 libastrofunc.dylib
    com.apple.quarantine       57
-rwxrwxrwx@ 1 user user     99272 Jan 17 11:19 libdllmain.dylib
    com.apple.quarantine       57
-rwxrwxrwx@ 1 user user     95576 Jan 17 11:19 libelops.dylib
    com.apple.quarantine       57
```

If the libraries were copied from a server, the quarantine flag is NOT set. It is only set for files downloaded from the internet.

To remove the "quarantine" bit, run the "**xattr -d <attribute> <file>**" command.

For example:

xattr -d com.apple.quarantine libastrofunc.dylib

xattr -d com.apple.quarantine libdllmain.dylib
xattr -d com.apple.quarantine libelops.dylib

Docker

It is also possible to use AstroStds in a Docker container via "*Docker Desktop for Mac*": https://hub.docker.com/editions/community/docker-ce-desktop-mac

9.4.

Docker runs on a Linux machine. "*Docker Desktop for Mac*" installs a Linux virtual machine onto a Mac OS. Once installed and ports are opened, you should be able to copy AstroStds libraries (as well as all other files) to your Docker instance(s). You can use a *Dockerfile* file to customize the contents of your Docker image and specify the executable to run upon instantiation. For debugging/maintenance, you can ssh into the instance like any other Linux machine. For launching a bash shell into the instance, you can run the "docker –it <docker ID> exec /bin/bash" command. To access a webpage within a Docker container, you will need to expose your public IP via network translation. This and other information is available in the Docker documentation.

See the Docker link shown above for further instructions on how to use Docker on the Mac OS.

References

2013. "2013 SSN Handbook."

Fricke , W., H. Schwan, and T. Lederle. 1988. *Fifth Fundamental Catalogue, Part I.* Heidelberg: Veröff. Astron. Rechen Inst.

Fricke, W., and A. Kopff. 1963. *Fourth Fundamental Katalog (FK4).* Heidelburg: Veroeff. Astron. Rechen-Institute.

Redding, Kevin J. 2000. *Look Angles Generation (LAMOD) Technical Report.* Science Applications International Corporation.