JSC Engineering Orbital Dynamics Planet Fixed Model

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

Package Release JEOD v5.1

Document Revision 1.0
July 2023

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC Engineering Orbital Dynamics
Planet Fixed Model

Document Revision 1.0
July 2023

A.A. Jackson and Jeff Morris

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Abstract

This model utility provides two coordinate transformations:

- A module for transformation from planet centered planet fixed Cartesian coordinates to planetospherical or planetoellipsoidal coordinates (a longitude, latitude, altitude set) and vice versa.(Note: in the model the word ellipse occurs in place of ellipsoidal coordinates. The reader should note that ellipsoidal is the standard usage for a given planetary body. See the Explanatory Supplement to the Astronomical Almanac [4].
- A module for transformation from planet centered planet fixed to a tangent plane system. This is a horizon system and here is chosen to be the North East Down specification with coordinates of altitude , latitude (or elliptical latitude) and longitude chosen as the non cartestian coordinates.(This is the reversed version of Local North East Up coordinates, where height (or altitude), right ascension and declination are chosen.)

The term planet_fix refers to the two utility modules planet_fixed_posn and north_east_down.
These coordinates and coordinate transformation apply to any body in the solar system. The reader however will find difference nomenclature in the literature for these as noted above.

Contents

Chapter 1

Introduction

1.1 Model Description

This model is a utility with JEOD to provide transformations between Cartesian , Spherical , Spheroidal and Tangent Plane coordinates. It is made up of two modules Planet Fixed and North East Down (a tangent plane system).

1.2 Document History

Author	Date	Description
A. A. Jackson and J. Morris	October 2009	Initial Version
A. A. Jackson and J. Morris	December 2009	Revision
R. O. Shelton	January 2011	Revision

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [3] and is organized into the following chapters:

1.3 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [3] and is organized into the following chapters:

Chapter 1: Introduction - JEOD has the capability to take a position vector in one coordinate system, and convert it to another coordinate system. JEOD has several coordinates systems, in the following focus is on two modules under the utility planet_fixed. There are four coordinate systems to consider, planet centered planet fixed Cartesian, planet fixed spherical, planet fixed elliptical and a tangent plane coordinate system. In this document for all planet fixed coordinates, no epoch has been specified, the fiducial reference is the prime meridian and the mean rotational axis of a planet. This module is meant to be generic and a reference body planet, satellite or other solar system body must be specified.

Chapter 2: Product Requirements - Describes requirements for the Planet Fixed Model.

Chapter 3: Product Specification - Describes the underlying theory, architecture, and design of the Planet Fixed Model in detail. It is organized in three sections: Conceptual Design, Mathematical Formulations, and Detailed Design.

Chapter 4: User Guide - Describes how to use the Planet Fixed Model in a Trick simulation. It is broken into three sections to represent the JEOD defined user types: Analysts or users of simulations (Analysis), Integrators or developers of simulations (Integration), and Model Extenders (Extension).

Chapter 5: Verification and Validation - Contains Planet Fixed Model verification and validation procedures and results.

Chapter 2

Product Requirements

Requirement PLANETFIX_1: Implement a set of planet fixed coordinates in JEOD v5.1

The requirement for the various planet fixed coordinate systems is to define the following coordinate systems and implement the ability to transform to and from each. Note: 'Fixed' is the astrodynamic specification for a coordinate system rotating with associated celestial body. The coordinate systems are:

1.1 Planet fixed Cartesian

- Planet fixed Cartesian:

    1. X-axis: The intersection of the prime meridian and the rotation equator of the named solar system object (planet, satellite, asteroid)
    2. Y-axis: Completes a standard, right-handed coordinate frame named solar system object (planet, satellite, asteroid)
    3. Z-axis: The mean rotation pole of the named solar system object (planet, satellite, asteroid)

1.2 Planet fixed spherical

- Planet fixed spherical

    1. Altitude h - The height above a given equatorial radius of the named solar system object (planet, satellite, asteroid)
    2. Longitude λ - Is angle measured along the mean equator positive east from the prime meridian of the named solar system object (planet, satellite, asteroid)
    3. Spherical Latitude θ - Is the angle measured from the mean equator positive north of the named solar system object (planet, satellite, asteroid)

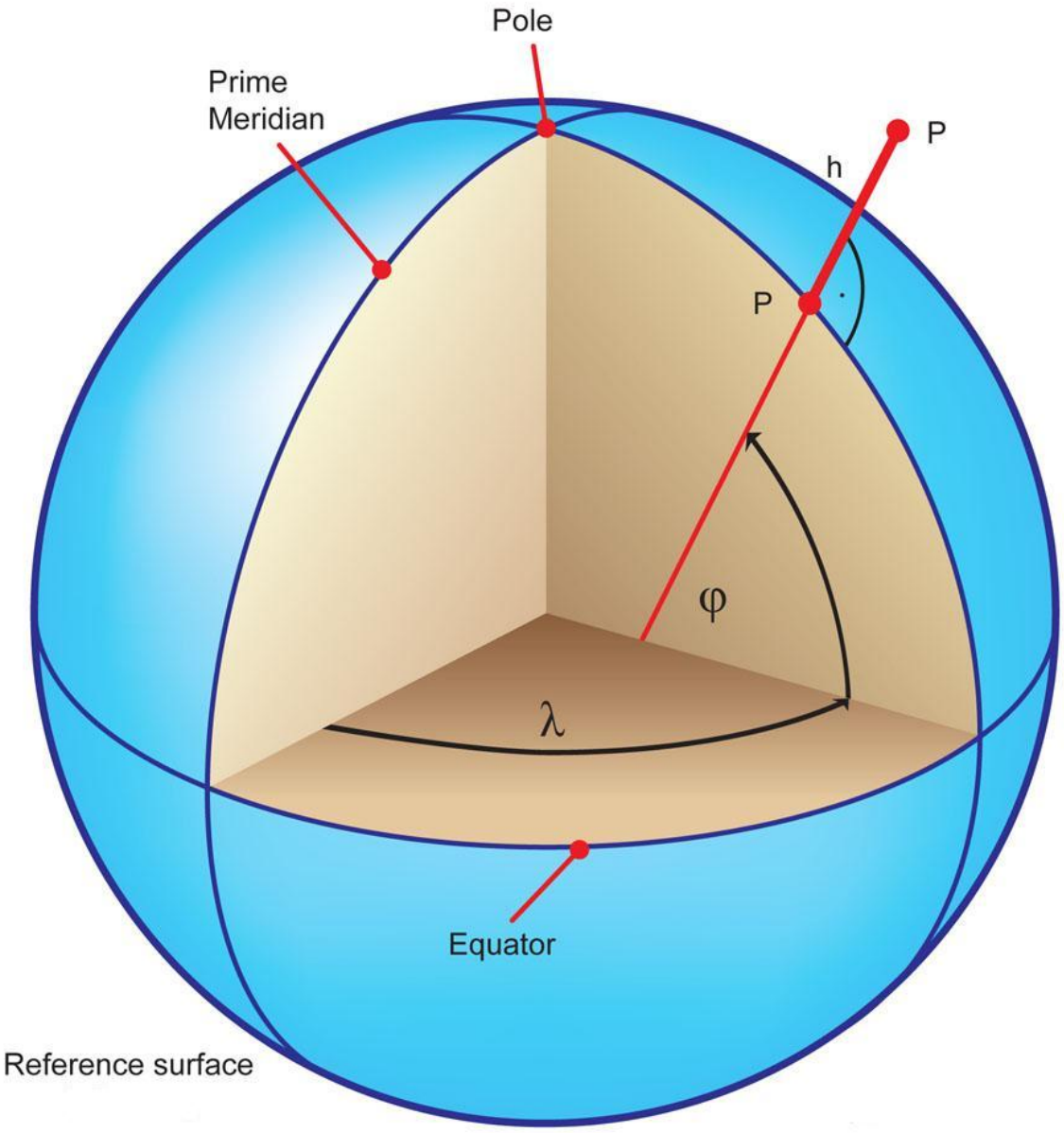

Figure 2.1: Prime meridian planet fixed

1.3 Planet fixed Elliptical

- Planet fixed elliptical

  1. Height h - At a point is the distance from the reference ellipsoid*3, see figure 2.2 to the point in a direction normal of the reference ellipsoid of the named solar system object (planet, satellite, and asteroid)
  2. Latitude elliptical θ - Is the angle between the equatorial plane and a line that is normal to the reference ellipsoid of the named solar system object (planet, satellite, and asteroid)
  3. Longitude λ - Is angle measured along the mean equator positive east from the prime meridian of the named solar system object (planet, satellite, asteroid) * A reference ellipsoid is defined by semi-major (equatorial radius) and flattening (the relationship between equatorial and polar radii of the named solar system object (planet, satellite, and asteroid).

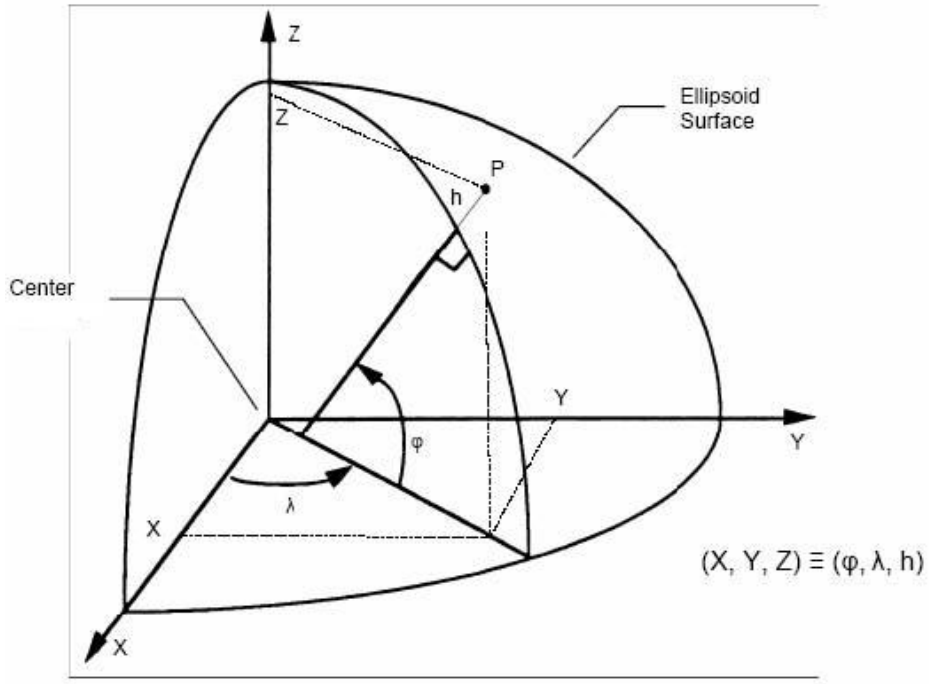

Figure 2.2: Planet fixed ellipsoid system

1.4 Planet fixed tangent plane

- Local Planet fixed tangent plane can be associated with either a spherical or an elliptical coordinate system. A tangent plane is attached to a fixed point on an elliptical (or spherical) surface a named solar system object (planet, satellite, and asteroid)see figure 2.3.

    1. x component of the coordinate (NORTH)
    2. y component of the coordinate (EAST)
    3. z component of the coordinate (UP)

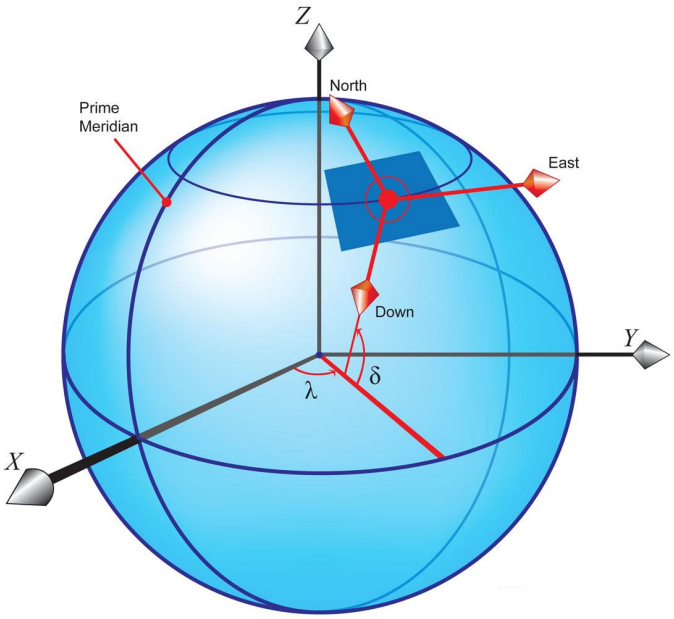

Figure 2.3: Local planet fixed tangent plane

Chapter 3

Product Specification

3.1 Conceptual Design

JEOD coordinate systems are generic in that any given solar system body (or bodies) one or several can be specified. For the Planet Fixed modules a set of four coordinate systems are specified. For solar system bodies the IAU differentiates between Planet Centered and planetographic body-fixed coordinates : Planet Centered latitude refers to the equatorial plane and the polar axis, planetographic latitude is defined as the angle between equatorial plane and a vector through the point of interest that is normal to the biaxial ellipsoid reference surface of the body. Both latitudes are identical for a spherical body. Planet Centered longitude is measured eastwards (i.e. positive in the sense of rotation) from the prime meridian. Planetographic longitude of the sub-observation point increases with time, i.e. to the west for prograde rotators and to the east for retrograde rotators.

Note in the following the terms spherical will be used for Planet Centered and elliptical for planetographic. The planet fixed tangent plane is the nominal 'local horizon's system defined here in 'aeronautic' terms of north east and down.

3.2 Mathematical Formulations

We describe the transformations between planet-fixed Cartesian , and the spherical and elliptical coordinate systems.

3.2.1 Basic Equations

We denote the Cartesian coordinates of an arbitrary point as the vector $\mathbf{R_0} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$, and the distance $r_0 = |\mathbf{R_0}| = \sqrt{\mathbf{x_0^2} + \mathbf{y_0^2} + \mathbf{z_0^2}}$ We assume that the planetary surface is modeled by a biaxial

ellipsoid described by the equation

$$\frac{x^2+y^2}{a^2}+\frac{z^2}{b^2}=1 \tag{3.1}$$

where a is the equatorial radius and b is the polar radius. The meridians of equal longitude for such a body are ellipses having semi-major axis a and semi-minor axis b. The parallels of equal latitude are circles.

We introduce $\rho=\sqrt{x^2+y^2}$ which allows us to rewrite 3.1 as the familiar equation of a meridian ellipse expressed in ρ and z.

$$\frac{\rho^2}{a^2}+\frac{z^2}{b^2}=1 \tag{3.2}$$

The eccentricity ϵ is defined by

$$\epsilon=\sqrt{1-\left(\frac{b}{a}\right)^2} \tag{3.3}$$

For both spherical and elliptical systems, longitude is defined as the angle between the projection of $\mathbf{R_0}$ in the X-Y plane and the positive X-axis and is given by the familiar equation

$$\lambda=\arctan\frac{y_0}{x_0} \tag{3.4}$$

For both spherical and elliptical systems, our coordinates will be triples of the form <altitude, latitude, longitude> and we will use the variables h, ϕ, and λ respectively to denote these quantities.

3.2.2 Spherical Coordinates

Here, we assume that the planet is a sphere, i.e. $a=b$ in 3.1. The altitude h in a spherical system is just the planet-centered distance r_0 less the radius a of the planet

$$h=\sqrt{x_0^2+y_0^2+z_0^2}-a \tag{3.5}$$

We already have a formula for λ given by 3.4.

The latitude ϕ is defined to be the angle between the local normal to the planetary surface and the equatorial $(X-Y)$ plane. When the planetary surface is modeled as a sphere, the surface normal coincides with the position vector $\mathbf{R_0}$, thus latitude is given by the simple equation

$$\begin{gathered}\phi=\arcsin(\frac{z_0}{r_0})= \\ \arctan(\frac{z_0}{\sqrt{r_0^2-z_0^2}})=\arctan(\frac{z_0}{\rho_0})\end{gathered} \tag{3.6}$$

where

$$\rho_0=\sqrt{x_0^2+y_0^2} \tag{3.7}$$

The transformation from spherical alt-lat-long to Cartesian is

$$\mathbf{R_0}=\begin{bmatrix}(a+h)\cos\phi\cos\lambda\\(a+h)\cos\phi\sin\lambda\\(a+h)\sin\phi\end{bmatrix} \tag{3.8}$$

3.2.3 Elliptical Coordinates

Borkowski [1] presents an efficient iterative method for determining elliptical latitude and altitude which provides exact solutions in very few iterations. The algorithm is implemented in PlanetFixedPosition::get_elliptic_parameters. For the sake of completeness, we provide a brief derivation based on Borkowski's paper.

Recall that the meridian of the Cartesian point $\mathbf{R_0}$ is represented by the equation 3.2 and can be parameterized as follows:

$$\begin{aligned} \rho &= a\cos\theta \\ z &= b\sin\theta \end{aligned} \tag{3.9}$$

The normal $\mathbf{N}$ to the ellipse meridian can be found by taking the gradient of the left-hand side of 3.2 and is given by

$$\mathbf{N} = \begin{bmatrix} 2\rho/a^2 \\ 2z/b^2 \end{bmatrix} \tag{3.10}$$

thus the tangent of the elliptical latitude ϕ must satisfy

$$\tan\phi = \frac{a^2}{b^2}\frac{z}{\rho} \tag{3.11}$$

which when combined with 3.9 yields the identity

$$\tan\phi = \frac{a}{b}\tan\theta \tag{3.12}$$

We can express the coordinates ρ_0 and z_0 in terms of θ, ϕ, and the altitude h as

$$\rho_0 = a\cos\theta + h\cos\phi \tag{3.13}$$

$$z_0 = b\sin\theta + h\sin\phi \tag{3.14}$$

If we multiply 3.13 by $\tan\phi$ and use 3.12, we obtain

$$\frac{a^2}{b}\sin\theta + h\sin\theta = \frac{a\rho_0\sin\theta}{b\cos\theta} \tag{3.15}$$

We eliminate h by subtracting 3.14 from 3.15 to obtain

$$\frac{a^2\sin\theta}{b} - b\sin\theta = \frac{a\rho_0\sin\theta}{b\cos\theta} - z_0 \tag{3.16}$$

If we multiply 3.16 by $b\cos\theta$ and rearrange, we have

$$(a^2 - b^2)\sin\theta\cos\theta - [a\rho_0\sin\theta - bz_0\cos\theta] = 0 \tag{3.17}$$

Multiplying 3.16 by $\frac{2}{\sqrt{a^2\rho_0^2+b^2z_0^2}}$ and using trig identities, we have

$$k\sin 2\theta - 2\sin(\theta - \omega) = 0 \tag{3.18}$$

where $k = \frac{a^2-b^2}{\sqrt{a^2\rho_0^2+b^2z_0^2}}$ and $\omega = \arctan\frac{bz_0}{a\rho_0}$. We solve 3.18 for θ by Newton's method. We observe

$$\theta^{(0)} = \arctan\frac{az_0}{b\rho_0} \tag{3.19}$$

is an exact solution of 3.13 and 3.14 for the case $h = 0$, so we use $\theta^{(0)}$ as a starting value for the Newton iteration. The iteration equation is

$$\theta^{(n+1)} = \theta^{(n)} - \frac{1}{2}\frac{k\sin(2\theta^{(n)}) - 2\sin(\theta^{(n)} - \omega)}{k\cos(2\theta^{(n)}) - cos(\theta^{(n)} - \omega)} \tag{3.20}$$

Starting with the initial value from 3.19, 3.20 normally converges in at most 3 steps. The equations for ϕ and h are

$$\phi = \arctan(\frac{a}{b}\tan\theta) \tag{3.21}$$

$$h = (\rho_0 - a\cos\theta)\cos\theta + (z_0 - b\sin\theta)\sin\theta \tag{3.22}$$

Going the other direction from <alt, lat, long> to Cartesian is straightforward from 3.13 and 3.14. If we let t stand for $\tan\phi$, and write $\sin\theta$ and $\cos\theta$ in terms of t, 3.13 and 3.14 become

$$\rho_0 = \frac{a}{\sqrt{1 + (\frac{b}{a}t)^2}} + h\cos\phi \tag{3.23}$$

$$z_0 = \frac{b\frac{b}{a}t}{\sqrt{1 + (\frac{b}{a}t)^2}} + h\sin\phi \tag{3.24}$$

Simplifying 3.23 by multiplying top and bottom of the first term by $a\cos\phi$, we get

$$\rho_0 = \left[h + \frac{a}{\sqrt{1 - \epsilon^2\sin^2\phi}}\right]\cos\phi \tag{3.25}$$

and

$$z_0 = \left[h + \sqrt{1-\epsilon^2}\frac{a}{\sqrt{1 - \epsilon^2\sin^2\phi}}\right]\sin\phi \tag{3.26}$$

Therefore, for the elliptical case, we can express $\mathbf{R_0}$ in terms of $< h, \phi, \lambda >$ as

$$\mathbf{R_0} = \begin{bmatrix} \left(h + \frac{a}{\sqrt{1-\epsilon^2\sin^2\phi}}\right)\cos\phi\cos\lambda \\ \left(h + \frac{a}{\sqrt{1-\epsilon^2\sin^2\phi}}\right)\cos\phi\sin\lambda \\ \left(h + \sqrt{1-\epsilon^2}\frac{a}{\sqrt{1-\epsilon^2\sin^2\phi}}\right)\sin\phi \end{bmatrix} \tag{3.27}$$

3.2.4 Local Tangent System

In many targeting and tracking applications the local East, North, Down (NED) Cartesian coordinate system can be more intuitive and practical than planet centered planet fixed (PCPF) or planet elliptic coordinates. The local NED coordinates are formed from a plane tangent to the Planet's surface fixed to a specific location and hence it is sometimes known as a "Local Tangent" or "local planeto-detic" plane (or geodetic) when referring to the Earth. By convention the axes are labeled east , the north and up . Given the PCPF cartesians the tangent plane Cartesians are :

$$\begin{bmatrix} x_{north} \\ y_{east} \\ z_{down} \end{bmatrix} = \begin{bmatrix} -\sin\varphi_g \cos\lambda & -\sin\varphi_g \sin\lambda & \cos\varphi_g \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\varphi_g \cos\lambda & -\cos\varphi_g \sin\lambda & -\sin\varphi_g \end{bmatrix} \begin{bmatrix} x_{pcpf} \\ y_{pcpf} \\ z_{pcpf} \end{bmatrix} \tag{3.28}$$

Given the PCPF velocity components tangent plane velocity components:

$$\begin{bmatrix} v_{north} \\ v_{east} \\ v_{down} \end{bmatrix} = \begin{bmatrix} -\sin\varphi_g \cos\lambda & -\sin\varphi_g \sin\lambda & \cos\varphi_g \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\varphi_g \cos\lambda & -\cos\varphi_g \sin\lambda & -\sin\varphi_g \end{bmatrix} \begin{bmatrix} v_{pcpf} \\ v_{pcpf} \\ v_{pcpf} \end{bmatrix} \tag{3.29}$$

3.3 Detailed Design

The model planet_fixed utility model is present in figures 3.1 and 3.2.

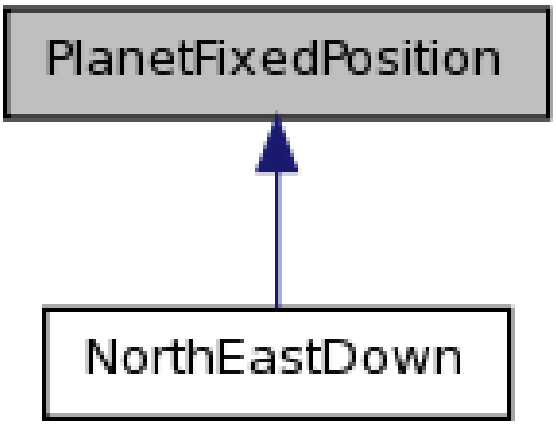

Figure 3.1: Planet fixed top level

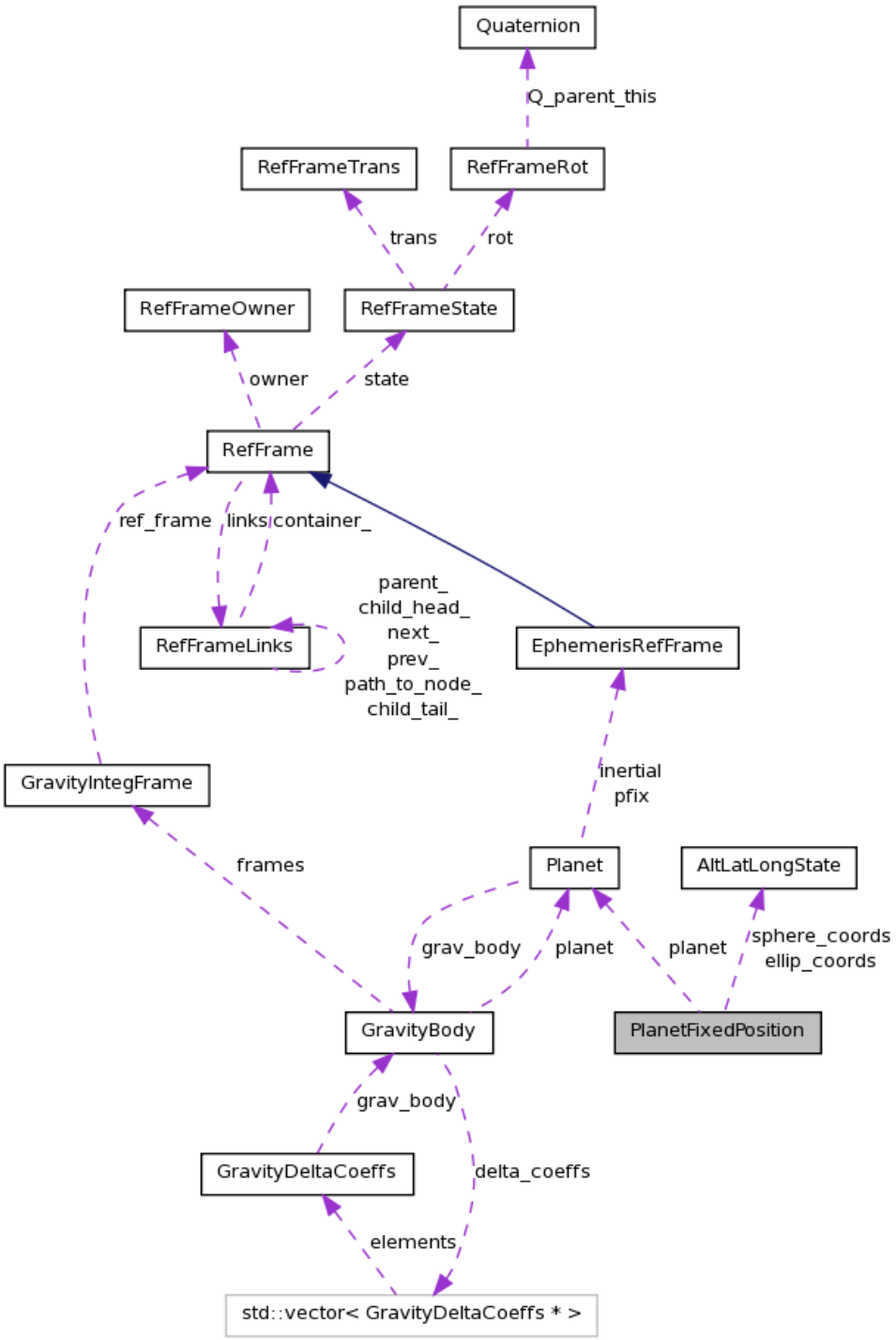

Figure 3.2: Planet fixed detailed

Chapter 4

User Guide

PLANETFIX is a utility in two modules that can be used to convert from Planet Centered Planet Fixed (PCPF) cartestian to PCPF spherical or ellipsoidal(note the nomenclature elliptical is used here for ellipsoidal)and vis versa. The second module PCPF cartesian, spherical and elliptical to a PCPF tangent plane at a given point and inversely.

4.1 The analyst

Specification of a coordinate point needs to be made: Examples:

```
earth.cartesian_pos[0]{km} = 6778.1363e3, 0.0, 0.0;

earth.spherical_pos.altitude{M} = 1000.0;
earth.spherical_pos.latitude {d} = 45.0;
earth.spherical_pos.longitude {d} = 45.0;

earth.elliptical_pos.altitude {km} = 500.0;
earth.elliptical_pos.latitude {d} = 45.0;
earth.elliptical_pos.longitude {d} = 45.0;

or as an example conversion of PCPF to NED
set the flag
earth.ned_frame.altlatlong_type = NorthEastDown::elliptical;
input coordinates:
earth.elliptical_pos.altitude {m} = 500.0;
earth.elliptical_pos.latitude {d} = 45.0;
earth.elliptical_pos.longitude {d} = 45.0;
```

4.2 SIM developers

Builders of S_define files and possibly input files.

```
//===========================================================================
// SIM_OBJECT: earth
// This sim object exercises the PlanetFixedPosition model.
//===========================================================================
sim_object {

   //
   // Data structures
   //

   // Structures needed for testing the full Planetary model functionality
   environment/gravity:       GravityCoeffs              gravity_coefs
      (environment/gravity/data/earth_GGM02C.d);

   environment/gravity:       GravityBody                gravity_source;

   environment/planet:        Planet                     planet
      (environment/planet/data/earth.d);

                              double                     cartesian_pos[3];
   utils/planet_fixed/planet_fixed_posn: AltLatLongState spherical_pos;
   utils/planet_fixed/planet_fixed_posn: AltLatLongState elliptical_pos;

   utils/message:             TrickMessageHandler        msg_handler;


   // Model to be tested
   utils/planet_fixed/planet_fixed_posn: PlanetFixedPosition pfix_pos;


   //
   // Initialization jobs
   //
   P_ENV (initialization) environment/gravity:
   earth.gravity_source.initialize_coefs (
      In   GravityCoeffs &      coefs        = earth.gravity_coefs);

   P_BODY (initialization) environment/planet:
   earth.planet.initialize ( );

   P_BODY (initialization) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.initialize (
```

```
      In   Planet *             planet_in    = &earth.planet);


   //
   // Environment class jobs
   //
   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_cart (
      In    const double              cart[3] = earth.cartesian_pos);

   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_spher (
      In    const AltLatLongState &   spher   = earth.spherical_pos);

   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_ellip (
      In    const AltLatLongState &   ellip   = earth.elliptical_pos);


} earth;
```

For NED:

```
//============================================================================
// SIM_OBJECT: earth
// This sim object exercises the North-East-Down model.
//============================================================================
sim_object {

   //
   // Data structures
   //

   // Structures needed for testing the North-East-Down model functionality
   environment/gravity:       GravityCoeffs          gravity_coefs
      (environment/gravity/data/earth_GGM02C.d);

   environment/gravity:       GravityBody            gravity_source;

   environment/planet:        Planet                 planet
      (environment/planet/data/earth.d);

                              double                 cartesian_pos[3];
                              double                 cartesian_vel[3];
   utils/planet_fixed/planet_fixed_posn: AltLatLongState spherical_pos;
   utils/planet_fixed/planet_fixed_posn: AltLatLongState elliptical_pos;
```

```
utils/message:             TrickMessageHandler    msg_handler;


// Model to be tested
utils/planet_fixed/north_east_down:   NorthEastDown   ned_frame;


//
// Initialization jobs
//
P_ENV (initialization) environment/gravity:
earth.gravity_source.initialize_coefs (
   In   GravityCoeffs &      coefs        = earth.gravity_coefs);

P_BODY (initialization) environment/planet:
earth.planet.initialize ( );

P_BODY (initialization) utils/planet_fixed/north_east_down:
earth.ned_frame.initialize (
   In    Planet *            planet_in    = &earth.planet);


//
// Environment class jobs
//
(0.0, environment) utils/planet_fixed/north_east_down:
earth.ned_frame.update_from_cart (
   In    const double              cart[3] = earth.cartesian_pos);

(0.0, environment) utils/planet_fixed/north_east_down:
earth.ned_frame.update_from_spher (
   In    const AltLatLongState &   spher   = earth.spherical_pos);

(0.0, environment) utils/planet_fixed/north_east_down:
earth.ned_frame.update_from_ellip (
   In    const AltLatLongState &   ellip   = earth.elliptical_pos);

(0.0, environment) utils/planet_fixed/north_east_down:
earth.ned_frame.build_ned_orientation ( );

(0.0, environment) utils/planet_fixed/north_east_down:
earth.ned_frame.set_ned_trans_states (
   In    double const        pos[3] = earth.cartesian_pos,
   In    double const        vel[3] = earth.cartesian_vel);
```

```
} earth;
```

4.3 Model extenders

- those who are interfacing JEOD with simulations. See the model (and it's component modules), the code and test simulations.

Chapter 5

Verification and Validation

5.1 Verification and Validation

Introduction
The following set of tests form a suite of the verification of specification and the validation for the planet_fixed module set(the two utility modules planet_fixed_posn and north_east_down) . The following set shows that planet_fixed specification are meet and thus verified. The first set of tests is located at:

`models/utils/planet_fixed/planet_fixed_posn/verif/SIM_PFIXPOSN_VERIF/SET_test/RUN_pfixposn_tes`

They test:

1. Test conversion from cartesian to spherical and elliptical coordinates
2. Test conversion from spherical to cartesian and elliptical coordinates
3. Test conversion from elliptical to cartesian and spherical coordinates

the second set is located at:

`models/utils/planet_fixed/north_east_down/verif/SIM_NED_VERIF/SET_test/RUN_nedtest`

The validation of all these tests can be recursively verified by providing a valid fiducial set of rectangular Cartesian or set of spheroidal geodetic coordinates. Conversely reversing the transformation to gain the original initial conditions, the reverse transform should generate the known initial conditions.

1. Test conversion from cartesian to spherical and elliptical coordinates
2. Test setting of NED translational states directly
3. Test conversion from elliptical to cartesian and spherical coordinates

The input file is a unit test configuration such that the individual module subroutines are called from within the planet_fixed model to execute the tests.

5.1.1 Test:RUN_pfixposn_test

- Unit Test 1:
  (a)Procedure: Conversion from cartesian to spherical and elliptical coordinates Input planet fixed coordinates: x = 6778136.3 meters
  y = 0.0 meters
  z = 0.0 meters
  Call the transformation routine at 1 second (b) Result: spherical altitude = 0.0 meters.
  spherical latitude = 0.0 deg.
  spherical longitude = 0.0 deg.
  elliptical altitude = 0.0 deg.
  elliptical latitude = 0.0 deg.
  elliptical longitude = 0.0 deg.
  (c)Pass/Fail Criterion:
  The test position is located on the equator at (equatorial radius, latitude 0 deg, long 0 deg) the height is the same as the equatorial radius thus is zero, the latitude and longitude , both spherical and elliptical are by default zero. The test is passed.

- Unit Test 2:
  (a)Procedure: Test conversion from spherical to cartesian and elliptical coordinates Input planet fixed coordinates at 2 seconds : spherical altitude = 1000.0 meters
  spherical latitude = 3.1416 radians (or 180 degrees)
  spherical longitude = 1.0 (radian), (or 57.2957795 degrees)
  Call the transformation routine at 1 second.
  (b) Result:
  x = 6.778136e+06 meters.
  y = 0.0 meters
  z = 0.0 meters
  elliptical altitude = 1.159497e-06 meters.
  elliptical latitude = -4.237551e-04 deg.
  elliptical longitude = -1.227042e+02 deg.
  (c)Pass/Fail Criterion:
  The initial values given in this test were computed independently using FORTRAN routines supplied by Vallado [5] and Borkowski [1] and found to differ by only about .5 meters and .3 degrees, since these are more sophisticated methods the results are considered in good agreement. The test is passed.

- Unit Test 3:
  Test conversion from elliptical to cartesian and spherical coordinates (a)Procedure:
  Test conversion from elliptical to cartesian and spherical coordinates Input planet fixed coordinates: elliptical altitude = 500.0 meters.
  elliptical latitude = 5.729578e+01 deg.
  elliptical longitude = 1.800004e+02 deg
  Call the transformation routine at 1 second
  (b) Result:
  x = -3454588.934 meters.
  y = -25.378827 meters

z = 5344189.181 meters
spherical altitude = -1.460475e+04 meters.
spherical latitude = 5.712058e+01 deg.
spherical longitude = -1.799996e+02 deg.
(c)Pass/Fail Criterion:
The initial values given in this test were computed independently using FORTRAN routines supplied by Vallado [5] and Borkowski [1] and found to differ by only about .5 meters and .3 degrees. These more sophisticated methods the produce results that are considered in good agreement with the JEOD unit test above. The test is passed.

5.1.2 Test:RUN_nedtest

- Unit Test 1:
  a)Procedure:
  Test setting of NED translational states directly :
  Input:
  x= 6778136.3 meters
  y= 0.0 meters
  z= 0.0 meters
  (b)Result both spherical and elliptical:
  Geodetic height = 399999.3 meters
  Latitude = 0.0 degrees
  Longitude = 0.0 degrees
  (c)Pass/Fail Criterion:
  The height is the difference between the radius given and the Earth's mean equatorial radius. The location is the prime meridian thus the longitude and latitude are (0,0) The test passes the success criterion.

- Unit Test 2:
  a)Procedure:
  Test conversion from cartesian to spherical and elliptical coordinates, as well as setting of translational (pos/vel) states based on conversion:
  Input
  x= 6778136.3 meters
  y= 0.0 meters
  z= 0.0 meters
  vx = 0.0 m/s
  vy = 100 m/s
  vz = 700 m/s
  (b)Result both spherical and elliptical
  Geodetic height = 399999.3 meters
  Latitude = 0.0 degrees
  Longitude = 0.0 degrees
  vxned = 0.0 m/s
  vyned = 100.0 m/s
  vzned = -700.0 m/s

(c)Pass/Fail Criterion:
The geodetic height is same as unit test 1.
The transform matrix is:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{5.1}$$

by hand calculation the resulting tangential coordinates are correct and the test is passed.

- Unit Test 3:
  a)Procedure:
  Test conversion from spherical to cartesian and elliptical coordinates, as well as setting of frame's orientation states based on conversion Input spherical
  altitude = 1000.0 m;
  latitude = 3.1416 r;
  longitude = 1.0 r;
  (b)Result (NED-Cartesians)
  x= -3446122 meters
  y= -5367017 meters
  z= -46.86 meters
  (c)Pass/Fail Criterion:
  Since one does not need to iterate for latitude the results can calculated by hand using equations (3.20), the results are same and the test validated.

- Unit Test 4:
  a)Procedure:
  Test conversion from elliptical to cartesian and spherical coordinates, as well as setting of frame's orientation states based on conversion Input elliptical altitude = 500.0 m
  latitude = 1.0 r
  longitude = 3.1416 r
  (b)Result
  xned = -3454588.934 m
  yned = -25.3788 m
  zned = 5344189.181 m

  (c)Pass/Fail Criterion:

This test is hard to verify by hand. An independent check with an example in the book Global Positioning System and Inertial Navigation [2] was input into the JEOD utility routine. From an example in chapter 2 the following is input and the planet_fixed module for this conversion was check by an independent simulation:

$$\begin{bmatrix} \lambda(longitude \\ \phi(latitude) \\ h(height) \end{bmatrix} = \begin{bmatrix} .59341195(radians) \\ -2.0478571(radians) \\ 251.702(meteres) \end{bmatrix} \tag{5.2}$$

With the resulting transformation:

$$\begin{bmatrix} x_{ned} \\ y_{ned} \\ z_{ned} \end{bmatrix} = \begin{bmatrix} -2,430,601.8 \\ -4,702,442.7 \\ 3,546,587.4 \end{bmatrix} (meters) \tag{5.3}$$

Therefore the algorithm is verified.

Chapter 6

Requirements Traceability

Table 6.1: planet_fixed Traceability - Tests

Requirments and Tests	
Planet fixed Cartesian 1.1	Test 5.1.1 and Test 5.1.1
Planet fixed spherical 1.2	Test 5.1.1 and Test 5.1.1
Planet fixed Elliptical 1.3	Test 5.1.1 and Test 5.1.2
Planet fixed tangent plane 1.4	Test 5.1.2 and Test 5.1.2 and Test 5.1.2

Appendices

Appendix A

S_define Unit Test One

```
//===========================================================================
// This simulation is a demonstration Trick simulation for exercising the base
// planet-fixed position model in JEOD. The following simulation objects are
// defined in this sim:
//
//          sys - Trick runtime executive and data recording routines
//         time - Universal time
//        earth - Object used to exercise planet-fixed position model
//
//===========================================================================


// Define job calling intervals
#define ENVIRONMENT     1.00   // Ephemeris update interval


// Define the phase initialization priorities.
// NOTE: Initialization jobs lacking an assigned phase initialization priority
// run after all initialization jobs that have assigned phase init priorities.
#define P_TIME  P10   // Highest priority; these jobs only depend on time
#define P_ENV   P20   // Environment initializations
#define P_BODY  P30   // Orbital body initializations



//===========================================================================
// SIM_OBJECT: sys
// This is the Trick executive model; this model should be basically
// the same for all Trick applications.
//===========================================================================
sim_object {
```

```
   //
   // Data structures
   //

   sim_services/include: EXECUTIVE exec (sim_services/include/executive.d);


   //
   // Automatic jobs
   //

   sim_services/input_processor: input_processor (
      Inout INPUT_PROCESSOR *IP = &sys.exec.ip);

} sys;



//============================================================================
// SIM_OBJECT: time
// This sim object relates simulation time to time on the Earth.
//============================================================================
sim_object {

   //
   // Data structures
   //
   environment/time: TimeManager time_manager;
   environment/time: TimeManagerInit time_manager_init;

   //
   // Initialization jobs
   //
   P_TIME (initialization)  environment/time: time.time_manager.initialize(
      Inout TimeManagerInit * time_manager_init = &time.time_manager_init);

   //
   // Scheduled jobs
   //
   (ENVIRONMENT, environment) environment/time: time.time_manager.update(
      In    double            simtime           = sys.exec.out.time);

} time;
```

```
//===========================================================================
// SIM_OBJECT: earth
// This sim object exercises the PlanetFixedPosition model.
//===========================================================================
sim_object {

   //
   // Data structures
   //

   // Structures needed for testing the full Planetary model functionality
   environment/gravity:       GravityCoeffs              gravity_coefs
      (environment/gravity/data/earth_GGM02C.d);

   environment/gravity:       GravityBody                gravity_source;

   environment/planet:        Planet                     planet
      (environment/planet/data/earth.d);

                              double                     cartesian_pos[3];
   utils/planet_fixed/planet_fixed_posn: AltLatLongState spherical_pos;
   utils/planet_fixed/planet_fixed_posn: AltLatLongState elliptical_pos;

   utils/message:             TrickMessageHandler        msg_handler;


   // Model to be tested
   utils/planet_fixed/planet_fixed_posn: PlanetFixedPosition pfix_pos;


   //
   // Initialization jobs
   //
   P_ENV (initialization) environment/gravity:
   earth.gravity_source.initialize_coefs (
      In   GravityCoeffs &      coefs        = earth.gravity_coefs);

   P_BODY (initialization) environment/planet:
   earth.planet.initialize ( );

   P_BODY (initialization) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.initialize (
      In   Planet *             planet_in    = &earth.planet);


   //
```

```
   // Environment class jobs
   //
   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_cart (
      In    const double              cart[3] = earth.cartesian_pos);

   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_spher (
      In    const AltLatLongState &   spher   = earth.spherical_pos);

   (0.0, environment) utils/planet_fixed/planet_fixed_posn:
   earth.pfix_pos.update_from_ellip (
      In    const AltLatLongState &   ellip   = earth.elliptical_pos);


} earth;
```

Appendix B

Input file for Unit Test 1

```
/*****************************************************************************
   Purpose:   Input file used for JEOD NorthEastDown model unit test

   Test:      Verify all methods of NorthEastDown work correctly

*****************************************************************************/

#define LOG_CYCLE 0.10

// Default data file generated in S_define
#include "S_default.dat"

// Set up simulation executive parameters
sys.exec.in.trap_sigfpe = Yes;

// Set up parameters to record
#include "Log_data/log_ned_verif.d"


// Set pointers between test planet and test grav_source
earth.planet.grav_source = &earth.gravity_source;
earth.gravity_source.planet = &earth.planet;


/* Test conversion from cartesian to spherical and elliptical coordinates, as
   well as setting of translational (pos/vel) states based on conversion */
read = 1.0;
earth.cartesian_pos[0] = 6778.1363e3, 0.0, 0.0;
call earth.earth.ned_frame.update_from_cart (earth.cartesian_pos);
```

```
// Test setting of NED translational states directly
read = 2.0;
earth.cartesian_pos[0] = 6778.1363e3, 0.0, 0.0;
earth.cartesian_vel[0] = 0.0, 100.0, -750.0;
call earth.earth.ned_frame.set_ned_trans_states (earth.cartesian_pos);


/* Test conversion from spherical to cartesian and elliptical coordinates, as
   well as setting of frame's orientation states based on conversion */
read = 3.0;
earth.spherical_pos.altitude = 1000.0;
earth.spherical_pos.latitude = 3.1416;
earth.spherical_pos.longitude = 1.0;
call earth.earth.ned_frame.update_from_spher (earth.spherical_pos);
earth.ned_frame.altlatlong_type = NorthEastDown::spherical;
call earth.earth.ned_frame.build_ned_orientation ();


// Reset enum defining the spherical/elliptical basis for N-E-D definition
read = 3.5;
earth.ned_frame.altlatlong_type = NorthEastDown::undefined;


/* Test conversion from elliptical to cartesian and spherical coordinates, as
   well as setting of frame's orientation states based on conversion */
read = 4.0;
earth.elliptical_pos.altitude = 500.0;
earth.elliptical_pos.latitude = 1.0;
earth.elliptical_pos.longitude = 3.1416;
call earth.earth.ned_frame.update_from_ellip (earth.elliptical_pos);
earth.ned_frame.altlatlong_type = NorthEastDown::elliptical;
call earth.earth.ned_frame.build_ned_orientation ();


// End simulation
stop = 5.0;
```

Appendix C

S_define Unit Test Two

```
//===========================================================================
// This simulation is a demonstration Trick simulation for exercising the JEOD
// North-East-Down model. The following simulation objects are defined in
// this sim:
//
//          sys - Trick runtime executive and data recording routines
//         time - Universal time
//        earth - Object used to exercise the North-East-Down model
//
//===========================================================================


// Define job calling intervals
#define ENVIRONMENT     1.00   // Environment update interval


// Define the phase initialization priorities
/* NOTE: Initialization jobs lacking an assigned initialization priority run
   after all initialization jobs that have assigned phase init priorities. */
#define P_TIME  P10   // Highest priority; these jobs only depend on time
#define P_ENV   P20   // Environment initializations
#define P_BODY  P30   // Orbital body initializations



//===========================================================================
// SIM_OBJECT: sys
// This is the Trick executive model; this model should be basically
// the same for all Trick applications.
//===========================================================================
sim_object {
```

```
   //
   // Data structures
   //
   sim_services/include: EXECUTIVE exec (sim_services/include/executive.d);

   //
   // Automatic jobs
   //
   sim_services/input_processor: input_processor (
      Inout INPUT_PROCESSOR *IP = &sys.exec.ip);

} sys;



//============================================================================
// SIM_OBJECT: time
// This sim object relates simulation time to time on the Earth.
//============================================================================
sim_object {

   //
   // Data structures
   //
   environment/time: TimeManager time_manager;
   environment/time: TimeManagerInit time_manager_init;

   //
   // Initialization jobs
   //
   P_TIME (initialization)  environment/time: time.time_manager.initialize(
      Inout TimeManagerInit * time_manager_init = &time.time_manager_init);

   //
   // Scheduled jobs
   //
   (ENVIRONMENT, environment) environment/time: time.time_manager.update(
      In    double            simtime           = sys.exec.out.time);

} time;



//============================================================================
// SIM_OBJECT: earth
// This sim object exercises the North-East-Down model.
```

```
//===========================================================================
sim_object {

   //
   // Data structures
   //

   // Structures needed for testing the North-East-Down model functionality
   environment/gravity:       GravityCoeffs          gravity_coefs
      (environment/gravity/data/earth_GGM02C.d);

   environment/gravity:       GravityBody            gravity_source;

   environment/planet:        Planet                 planet
      (environment/planet/data/earth.d);

                              double                 cartesian_pos[3];
                              double                 cartesian_vel[3];
   utils/planet_fixed/planet_fixed_posn: AltLatLongState spherical_pos;
   utils/planet_fixed/planet_fixed_posn: AltLatLongState elliptical_pos;

   utils/message:             TrickMessageHandler    msg_handler;


   // Model to be tested
   utils/planet_fixed/north_east_down:   NorthEastDown   ned_frame;


   //
   // Initialization jobs
   //
   P_ENV (initialization) environment/gravity:
   earth.gravity_source.initialize_coefs (
      In   GravityCoeffs &      coefs        = earth.gravity_coefs);

   P_BODY (initialization) environment/planet:
   earth.planet.initialize ( );

   P_BODY (initialization) utils/planet_fixed/north_east_down:
   earth.ned_frame.initialize (
      In    Planet *            planet_in    = &earth.planet);


   //
   // Environment class jobs
   //
```

```
   (0.0, environment) utils/planet_fixed/north_east_down:
   earth.ned_frame.update_from_cart (
      In    const double              cart[3] = earth.cartesian_pos);

   (0.0, environment) utils/planet_fixed/north_east_down:
   earth.ned_frame.update_from_spher (
      In    const AltLatLongState &   spher   = earth.spherical_pos);

   (0.0, environment) utils/planet_fixed/north_east_down:
   earth.ned_frame.update_from_ellip (
      In    const AltLatLongState &   ellip   = earth.elliptical_pos);

   (0.0, environment) utils/planet_fixed/north_east_down:
   earth.ned_frame.build_ned_orientation ( );

   (0.0, environment) utils/planet_fixed/north_east_down:
   earth.ned_frame.set_ned_trans_states (
      In    double const        pos[3] = earth.cartesian_pos,
      In    double const        vel[3] = earth.cartesian_vel);


} earth;
```

Appendix D

Input file for Unit Test 2

```
/*******************************************************************************
   Purpose:   Input file used for JEOD NorthEastDown model unit test

   Test:      Verify all methods of NorthEastDown work correctly

*******************************************************************************/

#define LOG_CYCLE 0.10

// Default data file generated in S_define
#include "S_default.dat"

// Set up simulation executive parameters
sys.exec.in.trap_sigfpe = Yes;

// Set up parameters to record
#include "Log_data/log_ned_verif.d"


// Set pointers between test planet and test grav_source
earth.planet.grav_source = &earth.gravity_source;
earth.gravity_source.planet = &earth.planet;


/* Test conversion from cartesian to spherical and elliptical coordinates, as
   well as setting of translational (pos/vel) states based on conversion */
read = 1.0;
earth.cartesian_pos[0] = 6778.1363e3, 0.0, 0.0;
call earth.earth.ned_frame.update_from_cart (earth.cartesian_pos);


// Test setting of NED translational states directly
```

```
read = 2.0;
earth.cartesian_pos[0] = 6778.1363e3, 0.0, 0.0;
earth.cartesian_vel[0] = 0.0, 100.0, -750.0;
call earth.earth.ned_frame.set_ned_trans_states (earth.cartesian_pos);


/* Test conversion from spherical to cartesian and elliptical coordinates, as
   well as setting of frame's orientation states based on conversion */
read = 3.0;
earth.spherical_pos.altitude = 1000.0;
earth.spherical_pos.latitude = 3.1416;
earth.spherical_pos.longitude = 1.0;
call earth.earth.ned_frame.update_from_spher (earth.spherical_pos);
earth.ned_frame.altlatlong_type = NorthEastDown::spherical;
call earth.earth.ned_frame.build_ned_orientation ();


// Reset enum defining the spherical/elliptical basis for N-E-D definition
read = 3.5;
earth.ned_frame.altlatlong_type = NorthEastDown::undefined;


/* Test conversion from elliptical to cartesian and spherical coordinates, as
   well as setting of frame's orientation states based on conversion */
read = 4.0;
earth.elliptical_pos.altitude = 500.0;
earth.elliptical_pos.latitude = 1.0;
earth.elliptical_pos.longitude = 3.1416;
call earth.earth.ned_frame.update_from_ellip (earth.elliptical_pos);
earth.ned_frame.altlatlong_type = NorthEastDown::elliptical;
call earth.earth.ned_frame.build_ned_orientation ();


// End simulation
stop = 5.0;
```

Bibliography

[1] K. M. Borkowski. Transformation of Geocentric to Geodetic Coordinates Without Approximations. *Bulletin Geodesique 63*, 139:1–4, Dec 1987.

[2] Matthew Barth Jay A. Farrell, Matthew Barth. *The Global Positioning System and Inertial Navigation*. McGraw-Hill, 1998.

[3] NASA. NASA Software Engineering Requirements. Technical Report NPR-7150.2, NASA, NASA Headquarters, Washington, D.C., September 2004.

[4] Seidelmann, P.K. *Explanatory Supplement to the Astronomical Almanac*. University Science Books, Sausalito, California, 2006.

[5] Vallado, D.A., with tech contributions by McClain, W.D. *Fundamentals of Astrodynamics and Applications, Second Edition*. Microcosm Press, El Segundo,Calif., 2001.