JSC-61777-models/environment/RNP

JSC Engineering Orbital Dynamics Rotation, Nutation, and Precession Model

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

Package Release JEOD v5.1

Document Revision 1.4
July 2023

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

JSC-61777-models/environment/RNP

JSC Engineering Orbital Dynamics Rotation, Nutation, and Precession Model

Document Revision 1.4
July 2023

Andrew Spencer, 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

The gravitational forces due to the Earth are modeled on the basis of a set of constant coefficients and a force model that computes the accelerations, referred to as a geopotential model. The current geopotential model available to JEOD v5.1 Geopotential models are commonly a function of a form of Earth Centered Earth Fixed, or ECEF, coordinates. In order to transform from an Earth Centered Inertial frame to the ECEF system a matrix rotation must be made.

These rotations can take many forms. The most common for Earth is one composed of four individual matrix operations. These operations account for precession, nutation, polar offset and rotation. This document will refer to this process as the Rotation, Nutation and Precession Model, or the RNP Model.

The JEOD v5.1 RNP model has been built as an extensible software framework. The basis for this framework is an interface to the JEOD Ephemerides model. This gives a standard interface for controlling the orientation between a planet's inertial frame and its planet fixed frame, regardless of the formulation for the orientation. Additionally, there is an implementation of this interface that builds a framework for implementation planet rotation models that follow the previously mentioned RNP paradigm. This allows for reuse of code, where different representations of planet RNP can be created while taking advantage of a common framework. This document describes how this common framework can be extended for use with any rotation model that follows the RNP paradigm.

This document also describes two specific implementations of the JEOD v5.1 Rotation, Nutation and Precession Model. The basic reference system of JEOD v5.1 is the inertial coordinate system known as J2000. The particular representation of RNP that transforms from the J2000 inertial coordinate system to the ECEF frame, has been implemented using the JEOD v5.1 RNP Model, and this document describes the JEOD v5.1 implementation of the J2000 (FK5) representation of Earth RNP. The document also describes an implementation of an RNP representation for the planet Mars, which is new to JEOD as of the version 2.2 release.

Note that while the RNP Model started as a planet orientation model specific to RNP, it has now expanded to support interfaces for any formulation of planet orientation. However, the name has stayed the same for legacy and consistency purposes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose and Objectives of the Rotation, Nutation, and Precession Model

Spacecraft trajectories and dynamics around massive gravitational bodies (planets) are often propagated in planet centered inertial frames, which are located at the center of the planet and have a constant orientation. These trajectories and dynamics, however, can also depend on models (such as atmospheric and gravitational models) requiring information based in a planet centered planet fixed frame. Converting from a planet centered inertial frame to a planet centered planet fixed frame requires a matrix rotation, referred to in this document as a planet rotation model.

Often these planet rotation models are represented as up to four individual matrix operations. These operations account for the concepts of precession, nutation, polar offset and rotation, referred to in this document as the Rotation, Nutation and Precession Model, or the RNP Model.

This document presents a generic implementation for planet orientation models, which creates a standard interface to the orientation portion of the JEOD Ephemerides Model. It also presents a generic implementation of RNP, built on the planet orientation interface. This allows for the reuse of basic code common to planet orientation models formulated in the RNP paradigm.

The generic RNP framework has also been used to create both a specific model for the standard Earth epoch of J2000, and a specific model to represent Mars RNP for the same J2000 epoch. The capability to compute spacecraft trajectories in a coordinate system defined by the mean equator and the equinox of the standard epoch of J2000 is a fixed requirement. The J2000 system is based on the position of far away stars in the J2000 system. The implementation is a collection of functions that compute the rotation, nutation, precession and polar motion from the J2000 (FK5) frame to obtain the Earth Centered Earth Fixed (ECEF) coordinate system. This document will also describe this specific implementation of the Rotation, Nutation and Precession Model which, of course, takes advantage of the JEOD v5.1 reuseable software framework, developed for implementations of the RNP Earth and Mars attitude models.

1.2 Context within JEOD

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [6]

The Rotation, Nutation, and Precession Model forms a component of the environment suite of models within JEOD v5.1. It is located at models/environment/RNP.

1.3 Document History

Author	**Date**	**Revision**	**Description**
Jeff Morris	February 2012	1.4	Added Mars RNP content
Andrew Spencer	January 2012	1.3	Added details for PlanetOrientation class
Andrew Spencer	October 2010	1.2	Reflected changes in enumerations, added metrics
Blair Thompson	February 2010	1.1	Updated default polar motion table info
Andrew Spencer	November 2009	1.0	Initial Version

1.4 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [13].

The document comprises chapters organized as follows:

Chapter 1: Introduction -This introduction describes the objective and purpose of the Rotation, Nutation, and Precession Model.

Chapter 2: Product Requirements -The requirements chapter describes the requirements on the Rotation, Nutation, and Precession Model.

Chapter 3: Product Specification -The specification chapter describes the architecture and design of the Rotation, Nutation, and Precession Model.

Chapter 4: User Guide -The user guide chapter describes how to use the Rotation, Nutation, and Precession Model.

Chapter 5: Inspections, Tests, and Metrics -The inspections, tests, and metrics describes the procedures and results that demonstrate the satisfaction of the requirements for the Rotation, Nutation, and Precession Model.

Chapter 2

Product Requirements

This chapter will describe the requirements for the Rotation, Nutation and Precession Model.

Requirement RNP_1: Top-level

Requirement:
This model shall meet the JEOD project requirements specified in the JEOD v5.1 top-level document.

Rationale:
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.1 release.

Verification:
Inspection

2.1 Data Requirements

This section identifies requirements for the data represented by the Rotation, Nutation and Precession Model. These as-built requirements are based on the Rotation, Nutation and Precession Model data definition header files.

Requirement RNP_2: Earth RNP Data Requirement

Requirement: The Rotation, Nutation and Precession Model shall encapsulate data to facilitate the transformation from base J2000 inertial Earth coordinates into WGS-84 Earth Centered Earth Fixed coordinates. The RNP Model shall also encapsulate data to enable the inclusion of a polar offset if higher fidelity is required. Finally, each component of the RNP Model (Rotation, Nutation, Precession, Polar Motion) shall be available as a separate matrix.

Rationale: The data generated by the Rotation, Nutation and Precession Model are the Earth Centered Earth Fixed (ECEF) coordinates fitted to the coordinate system listed in the requirement above. The state vector of a vehicle is generally given in an inertial coordinate system (J2000), and geopotential gravitational accelerations are computed in the ECEF system, thus the Rotation, Nutation and Precession Model must transform between the two sets of coordinates.

Verification:
Unit tests are described in the Rotation, Nutation and Precession Model validation chapter. The verifiable parameters should agree to as many decimal places as those presented in the validation data from the benchmark documents Satellite Orbits, Montenbruck and Gill [11] and Fundamentals of Astrodynamics and Applications, Second Edition, Vallado [18]. Metric measuring of the modeling contributions to radial, cross track, and in-track errors is given in the JEOD v5.1 Top Level Document [6].

Requirement RNP_3: Mars RNP Data Requirement

Requirement: The Rotation, Nutation and Precession Model shall encapsulate data to facilitate the transformation from base J2000 inertial Mars coordinates into Pathfinder Mars Centered Mars Fixed coordinates. Each component of the RNP Model (Rotation, Nutation, Precession, Polar Motion) shall be available as a separate matrix.

Rationale: The data generated by the Rotation, Nutation and Precession Model are the Mars Centered Mars Fixed (MCMF) coordinates fitted to the coordinate system listed in the requirement above. The state vector of a vehicle is generally given in an inertial coordinate system (J2000), and geopotential gravitational accelerations are computed in the MCMF system, thus the Rotation, Nutation and Precession Model must transform between the two sets of coordinates.

Verification:
Verification of this requirement is by inspection.

2.2 Functional Requirements

This section identifies requirements on the functional capabilities provided by the Rotation, Nutation and Precession Model. These as-built requirements are based on the Rotation, Nutation and Precession Model source files. See math model section in the specification chapter.

Requirement RNP_4: Earth RNP Functional Requirement

Requirement:
The RNP Model shall provide the functionality to convert from the Earth J2000 inertial reference frame to WGS-84 Earth Centered Earth Fixed coordinates.

Rationale:
The purpose of the J2000 implementation of the Rotation, Nutation and Precession Model is to provide the Earth fixed coordinates used in many orbital dynamics routines.

Verification:

Unit tests are described in the Rotation, Nutation and Precession Model Verification and Validation chapter. The verifiable parameters should agree to as many decimal places as the validation data presented in the benchmark documents Satellite Orbits, Montenbruck and Gill [11] and Fundamentals of Astrodynamics and Applications, Second Edition, Vallado [18]. Metric measuring of the modeling contributions to radial, cross track, and in-track errors is given in the JEOD v5.1 Top Level Document [6].

Requirement RNP_5: Mars RNP Functional Requirement

Requirement:

The RNP Model shall provide the functionality to convert from the Mars J2000 inertial reference frame into Pathfinder Mars Centered Mars Fixed coordinates.

Rationale:

The purpose of the RNPMars implementation of the Rotation, Nutation and Precession Model is to provide the Mars fixed coordinates used in gravitational and other orbital dynamics routines.

Verification:

Verification of this requirement is by inspection, since virtually all underlying calculations of this model extension are verified as part of the Earth J2000 implementation verification. [6].

Requirement RNP_6: Planet Orientation Extensibility

Requirement:

The JEOD Rotation, Nutation and Precession Model shall provide a standard interface to the JEOD Ephemerides model, which can be used for implementing planet orientation schemes based on arbitrary formulations.

Verification:

The verification for this requirement will be done by inspection.

Requirement RNP_7: RNP Extensibility

Requirement:

The JEOD Rotation, Nutation and Precession Model shall provide an extensible framework appropriate for implementing orientation schemes based on the rotation, nutation, precession and polar motion framework.

Rationale:

The J2000 representation of RNP is not the only one in existence, and new and better implementations are being developed that JEOD will likely incorporate in the future.

Verification:

The verification for this requirement will be done by inspection.

Chapter 3

Product Specification

3.1 Conceptual Design

This section will present the conceptual design for the general Rotation, Nutation and Precession Model framework, the J2000 (FK5) specific implementation, and the Mars specific implementation.

3.1.1 General Framework

The general framework for the Rotation, Nutation and Precession Model is made up of four generic classes:

- The generic planet orientation class, which provides an interface to the DynManager class intended for the implementation of a planet fixed frame with respect to the planet's inertial frame,
- The generic RNP class, which takes generic nutation, rotation, precession and polar motion matrices and combines them in the correct manner,
- The generic planet rotation class, the base class for any version of a nutation, rotation, precession or polar motion matrix,
- A generic initialization class for planet rotations, a base class for any class used to initialize a planet rotation class.

The generic planet orientation class interfaces with the DynManager class by implementing a planet orientation specific version of the EphemerisInterface class. This class provides an interface for any object intending to interact with a DynManager for the purpose of controlling a planet's orientation. It provides most of the virtual function implementations necessary to instantiating a concrete version of the EphemerisInterface. It also provides common tools necessary for interacting with the DynManager. Further information about the DynManager [5] and the EphemerisInterface [16] can be found in their respective documents.

The generic RNP class contain implementation generic to all planet RNP formulations; the containment of nutation, rotation, precession and polar motion objects (which can be turned off for formulations without this component), the algorithms for combining them, and member and utility functions common to the individual planet rotations that make up the complete RNP rotation.

The remaining classes, the planet rotation class and its initializer, provide tools common to rotation matrices used to implement the major components of an RNP model. Additionally, these rotation classes can more broadly be used to implement many other components of a planet orientation model needing rotation matrices.

This generic framework is designed to work in concert with the JEOD v5.1 Time Representations Model, and the JEOD v5.1 Dynamics Manager. More information on these models can be found in the Time Representations Model documentation [17] and the Dynamics Manager documentation [5].

3.1.2 J2000 / FK5 Specific Implementation

The specific implementation of RNP that transforms from the J2000 ECI frame to the ECEF frame is an implementation of the IAU-76/FK5. Further information about this specification can be found in [1] and [19].

3.1.3 Mars Pathfinder Specific Implementation

The specific implementation of RNP that transforms from the J2000 MCI frame to the MCMF frame is an implementation named for the Mars Pathfinder mission for which it was first used. The Pathfinder Mars RNP model is described in [8] and [9].

3.2 Mathematical Formulations

The ultimate product of the Rotation, Nutation and Precession Model is the transformation from a planet centered inertial frame to a planet centered planet fixed frame. For Earth, this transformation is from the Earth Centered Inertial (ECI) frame defined by the standard epoch J2000, to the Earth Centered Earth Fixed (ECEF) frame, as defined by the World Geodetic System 1984 [3]. The transformation from planet inertial to planet centered planet fixed is often broken into four parts: rotation, nutation, precession and polar motion. This section will give a mathematical basis for the meaning of these separate transformations, using the Earth RNP transformations to illustrate.

In this discussion the reader may find the following definitions helpful:

Definitions:

Luni-Solar Precession - 50" per year, period of ~26,000 years (due to the torques of the Moon and the sun on the Earth's equatorial bulge).

Planetary Precession - precession of 12"/century and decrease of the obliquity of the ecliptic of 47"/century (due the planetary perturbations on the Earth's orbit, causing changes in the ecliptic).

ζ_A, Z_A, θ_A are Euler angles representing the precession of the mean celestial ephemeris pole with respect to the space-fixed coordinate system defined by the J2000 epoch. See figure 3.1.

Nutation - amplitude 9”, occurs at orbital periods of the sun and the moon (13.7 days, 27.6 days, 6 months, 1 year, 18.6 years, etc.) 18.6 year motion is largest - 20 arc sec amplitude (0.5 km). ε = The mean obliquity of the ecliptic. $\Delta\psi$ = Nutation in longitude $\Delta\varepsilon$ = Nutation in the obliquity. See figure 3.2

Polar Motion Linear - drift of the rotation pole of 3-4 milli arc seconds/year in a direction between Greenland and Hudson Bay (due to post glacial rebound). Long period wobble (30 years) of amplitude 30 milli arc seconds.

Annual Wobble - (amplitude of 0.1 arc seconds - 3 meters on the Earth’s surface), 75% caused by annual variation in the inertia tensor of the atmosphere, rest by mass variations in snow, ice, ground water, etc. Chandler Wobble (430 day period), 6 meters amplitude. Normal mode of the Earth. Caused by atmospheric and oceanic effects.

Equator - the great circle on the surface of a body formed by the intersection of the surface with the plane passing through the center of the body perpendicular to the axis of rotation.

Celestial Equator - the projection onto the celestial sphere of the Earth’s equator.

Ecliptic - the plane of the Earth’s orbit about the sun, affected by planetary precession.

True Equator and Equinox of Date - the celestial coordinate system determined by the instantaneous positions of the celestial equator and ecliptic (motion due to precession and nutation).

Mean Equator and Equinox of Date - the celestial reference system determined by ignoring small variations of short period (nutation) in the motions of the celestial equator (motion due to only precession).

Mean Equator and Equinox of J2000.0 - the celestial reference system at 12 hours, January 1, 2000.

The version of RNP used in JEOD v5.1, found in [19] for Earth and [8] for Mars, transforms a vector from the planet centered planet fixed frame to the J2000 planet centered inertial frame in the following manner:

$$\vec{r}_{ECI} = \mathbf{P} * \mathbf{N} * \mathbf{R} * \mathbf{PM} * \vec{r}_{ECEF} \tag{3.1}$$

where

- $\vec{r}_{ECI}$ is a position vector in the ECI frame,
- $\vec{r}_{ECEF}$ is the same position vector rotated into the ECEF frame,
- $\mathbf{P}$ is the matrix associated with precession,
- $\mathbf{N}$ is the matrix associated with nutation,
- $\mathbf{R}$ is the matrix associated with rotation, and
- $\mathbf{PM}$ is the matrix associated with polar motion.

These are transformation matrices, making the writing of an equivalent transform from J2000 planet centered inertial frame to the planet centered planet fixed frame simple, and shown below:

$$\vec{r}_{ECEF} = \mathbf{PM^T} * \mathbf{R^T} * \mathbf{N^T} * \mathbf{P^T} * \vec{r}_{ECI} \tag{3.2}$$

This collection of matrices is the full transform from planet centered inertial to planet centered planet fixed coordinates, and is referred to in this document as the **RNP** matrix, defined below:

$$\mathbf{RNP} = \mathbf{PM^T} * \mathbf{R^T} * \mathbf{N^T} * \mathbf{P^T} \tag{3.3}$$

The following sections will describe the mathematical formulation for each of these separate matrices. Note that the Mars Pathfinder RNP matrices are constructed similarly, but using different parameters and slightly different equations.

3.2.1 Precession

The J2000 Precession matrix transforms a vector from the Mean Equator and Equinox of Date (MOD) to the mean of equator and equinox of epoch J2000. This transformation is represented by three consecutive single axis rotations, as shown below in 3.1.

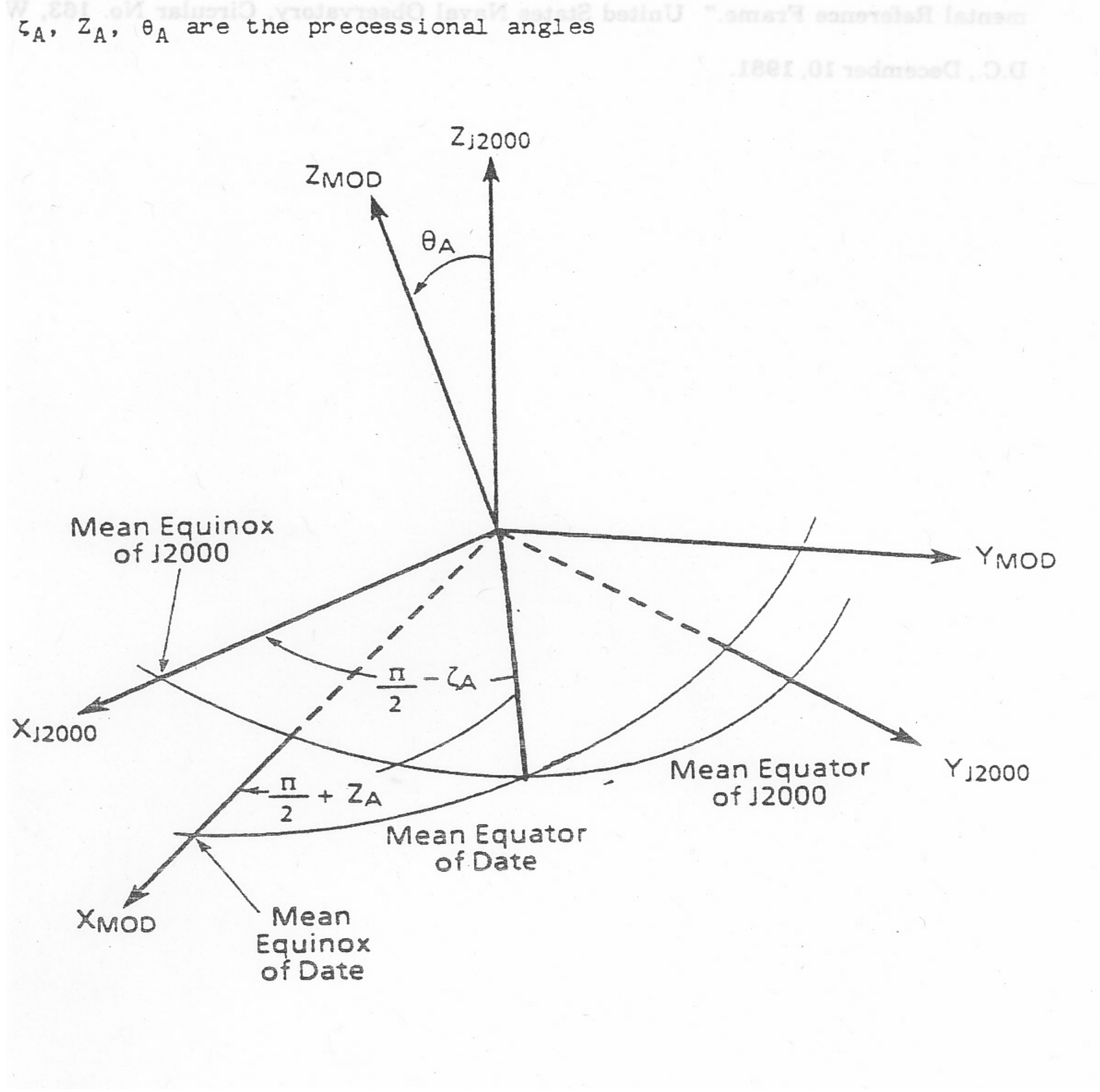

Figure 3.1: Rotation from the mean of equator and equinox of epoch J2000 to Mean Equator and Equinox of Date (MOD) [1]

These three angles are represented as ζ, Θ, and z. These values are calculated using Terrestrial Dynamic Time, which is further explained in the JEOD v5.1 Time Representations Model documentation [17]. Terrestrial time, indicated here as TT, is then transformed into centuries since the J2000 epoch through the following relationship:

$$T_{TT} = \frac{TT - 2451545.0}{36525} \tag{3.4}$$

where

- TT is the Julian Date in Terrestrial Dynamic Time, and

- T_{TT} is the number of Julian Centuries, in Terrestrial Dynamic Time, since the J2000 Epoch.

The three angles are then calculated, according to Vallado [19], as:

$$\zeta = 2306.2181" * T_{TT} + 0.30188 * T_{TT}^2 + 0.017998 * T_{TT}^3 \tag{3.5}$$
$$\Theta = 2004.3109" * T_{TT} + 0.42665 * T_{TT}^2 + 0.041833 * T_{TT}^3 \tag{3.6}$$
$$z = 2306.2181" * T_{TT} + 1.09468 * T_{TT}^2 + 0.018203 * T_{TT}^3 \tag{3.7}$$

The precession matrix is then formed in the following manner, per Vallado [19]:

$$\mathbf{P} = \begin{pmatrix} \cos\Theta * \cos z * \cos\zeta - \sin z * \sin\zeta & \sin z * \cos\Theta * \cos\zeta + \sin\zeta * \cos z & \sin\Theta * \cos\zeta \\ -\sin\zeta * \cos\Theta * \cos z - \sin z * \cos\zeta & -\sin z * \sin\zeta * \cos\Theta + \cos z * \cos\zeta & -\sin\Theta * \sin\zeta \\ -\sin\Theta * \cos z & -\sin\Theta * \sin z & \cos\Theta \end{pmatrix} \tag{3.8}$$

3.2.2 Nutation

The J2000 Nutation matrix transforms a vector from the True Equator and Equinox of date (TOD) to the Mean Equator and Equinox of Date (MOD). This transformation is represented by three consecutive single axis rotations, as shown below in 3.2.

The J2000 Nutation matrix, similar to the Precession matrix, is based on three angles, defined in [1] as:

- ϵ: The mean obliquity of the ecliptic,

- $\Delta\Psi$: The nutation in longitude, and

- $\Delta\epsilon$: The nutation in obliquity

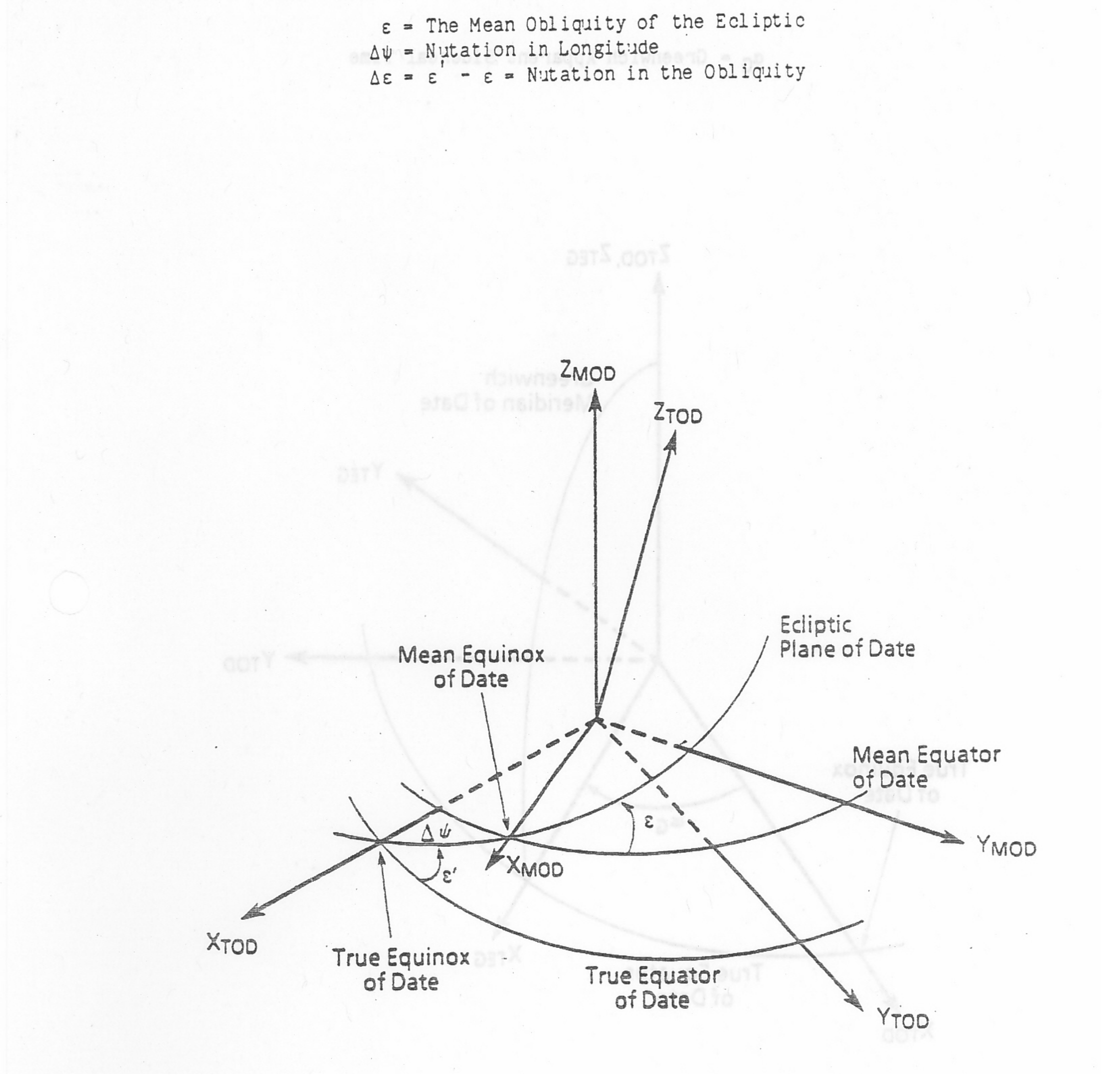

Figure 3.2: Rotation from Mean Equator and Equinox of Date (MOD) to True Equator and Equinox of date (TOD) [1]

Additionally, [1] defines the relationship:

$$\epsilon' = \epsilon + \Delta\epsilon \tag{3.9}$$

Where ϵ' is the true obliquity of the ecliptic.

Similar to precession, the nutation matrix is dependent on the centuries since the J2000 epoch calculated using Terrestrial Dynamic Time. This calculation is explained in the previous section, and shown in Equation 3.4.

ϵ is then calculated as a polynominal in T_{TT}:

$$\epsilon = 84381.448" - 46.8150 * T_{TT} - 0.00059 * T_{TT}^2 + 0.001813 * T_{TT}^3 \tag{3.10}$$

The nutation in longitude and nutation in obliquity are then calculated using trigonometric series. These formulations are long and complex, and the full details will not be presented here. These formulations in full can be found in [1].

The nutation matrix is then computed in the following manner, using the calculated angles, as follows [1]:

$$\mathbf{N} = \begin{pmatrix} \cos\Delta\Psi & \cos\epsilon' * \sin\Delta\Psi & \sin\epsilon' * \sin\Delta\Psi \\ -\sin\Delta\Psi * \cos\epsilon & \cos\epsilon' * \cos\Delta\Psi * \cos\epsilon + \sin\epsilon' + \sin\epsilon & \sin\epsilon' * \cos\Delta\Psi * \cos\epsilon - \cos\epsilon' * \sin\epsilon \\ -\sin\Delta\Psi * \sin\epsilon & \cos\epsilon' * \cos\Delta\Psi * \sin\epsilon - \sin\epsilon' * \cos\epsilon & \sin\epsilon' * \cos\Delta\Psi * \sin\epsilon + \cos\epsilon' * \cos\epsilon \end{pmatrix} \tag{3.11}$$

Note that, while the notation of [1] is used, the matrix represented in (3.11) is the transpose of what is shown in [1], which for the ortho-normal transformation matrix is of course the inverse of the original matrix. This is a result of [19] using a different formulation for the relationship between the J2000 ECI frame and the ECEF frame, as shown in 3.1, where the rotation, nutation, rotation and polar motion matrices have been transposed and moved to the opposite side. The end formulations and results, however, are completely equivalent.

A graphical representation of the effects of nutation and precession can be seen below in 3.2.2.

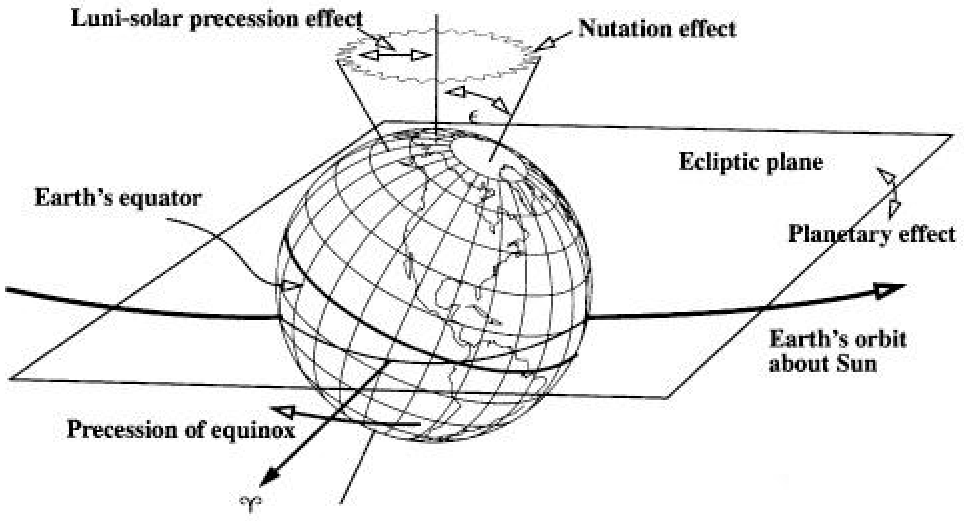

Figure 3.3: The effects of nutation and precession on the Earth rotation axis.[19]

3.2.3 Rotation

The rotation matrix contained in the RNP formulation represents the sidereal rotation of the Earth. This is the major, axial rotation of the Earth, and is the largest contributor to the Earth orientation.

The rotation matrix is calculated from Greenwich Mean Sidereal Time, which is further explained in the JEOD v5.1 Time Representations Model documentation [17]. This time is then converted to an angle, as described in [19], using the following relationship:

$$\theta_{GMST} = \frac{T_{GMST}}{240} \tag{3.12}$$

This angle is known as the Greenwich Apparent Sidereal Time, and is calculated using the following relationship [19]:

$$\theta_{GAST} = \theta_{GMST} + \Delta\Psi * \cos\epsilon' \tag{3.13}$$

where $\cos\epsilon'$ is known as the equation of the equinoxes.

This angle is then converted to a rotation matrix through the following relationship:

$$\mathbf{R} = \begin{pmatrix} \cos\theta_{GAST} & -\sin\theta_{GAST} & 0.0 \\ \sin\theta_{GAST} & \cos\theta_{GAST} & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \tag{3.14}$$

3.2.4 Polar Motion

The J2000 polar motion is based on a table of values for angular displacements of the axis of rotation of the planet. These displacements represent angular displacements of the pole in the x_p direction (positive along the direction of the Prime Meridian), and the y_p direction (positive along the 90°W meridian. Thus, the rotation matrix based on these angular displacements, as represented in [19] is:

$$\mathbf{PM} = \begin{pmatrix} \cos x_p & 0 & -\sin x_p \\ \sin x_p * \sin y_p & \cos y_p & \cos x_p * \sin y_p \\ \sin x_p * \cos y_p & -\sin y_p & \cos x_p * \cos y_p \end{pmatrix} \tag{3.15}$$

Because x_p and y_p are extremelly small, this matrix is often simplified to:

$$\mathbf{PM} = \begin{pmatrix} 1 & 0 & -x_p \\ 0 & 1 & y_p \\ x_p & -y_p & 1 \end{pmatrix} \tag{3.16}$$

3.3 Detailed Design

3.3.1 Planet Orientation Class

The PlanetOrientation class implements an inheriting class of the EphemerisInterface model. This implementation intends to provide an EphemerisInterface for controlling a planet's orientation.

The exact details of the EphemerisInterface class and how it interacts with the EphemerisManager portion of the DynManager can be found here [16].

The PlanetOrientation provides implementations of the following EphemerisInterface functions:

- ephem_initialize,
- ephem_activate, and
- ephem_build_tree.

"name", "timestamp", "ephem_update" and "get_name" are all left to the inheriting class.

The PlanetOrientation class also provides tools for interacting with the DynManager [5]. This includes providing tools for turning a PlanetOrientation model on and off and instantiating an EphemerisOrientation object [16], intended to control the planet orientation in question.

Additionally, the PlanetOrientation class is also a child class of the RefFrameOwner class [15], providing tools for interacting with the RefFrame representing the planet fixed frame associated with the PlanetOrientation model.

3.3.2 Generic RNP Framework

The Generic RNP Framework is split into 4 classes, giving basic functionality necessary to interacting with the rest of the JEOD package. These classes are:

- PlanetRotation,
- PlanetRotationInit,
- PlanetRNP, and
- RNPMessages

The following sections will give a description of the purpose of each of these classes.

PlanetRotation

The PlanetRotation class is a generic base class used to implement any one of the single transformations encapsulated by the complete RNP transformation (i.e. precession, nutation, rotation or polar motion). It contains functionality common to transformation matrices, including a data field for a 3x3 matrix, ability to 'get' the current transformation matrix as well as its transform, and the ability to set the time that the next update of the rotation should use as the independent variable.

The PlanetRotation class also contains a polymorphic interface for updating the current transformation. This allows for the PlanetRotation class to be used for any individual RNP transformation, giving a great deal of flexibility to the end user when extending the generic framework for a particular implementation of RNP.

PlanetRotationInit

The PlanetRotationInit class is an extremely simple base class. It is intended to be used in the case of large amounts of data being required in a specific PlanetRotation derived class, where the data is not meant to be changed after initialization.

There is no data contained in the base class PlanetRotationInit by default; all data is intended to be added by the user upon inheritance.

The PlanetRotationInit derived class is then for initialization through the PlanetRotation method having the following form:

```
virtual void initialize(PlanetRotationInit* init);
```

This function takes a pointer to a PlanetRotationInit derived class, through a PlanetRotationInit base class pointer, utilizing polymorphism. Data from the PlanetRotationInit derived class is then appropriately copied into the PlanetRotation derived object.

PlanetRNP

The PlanetRNP class is the main, encapsulating class for the generic RNP framework. It gives functionality for interfacing with the JEOD Dynamics Manager Model [5], as the Dynamics Manager is where the final transformation from ECI to ECEF coordinates will be contained.

The PlanetRNP also encapsulates the individual transformations that make up the complete RNP transformation. These are in the form of four PlanetRotation pointers representing precession, nutation, rotation and polar motion. This represents a polymorphic interface to PlanetRotation derived classes representing specific implementations of these individual transformations. Additionally, the PlanetRNP base class gives the implementation of (3.3), which can be used by any class that inherits from PlanetRNP. This is enabled by the polymorphic interface created by inheriting individual RNP transformations from the PlanetRotation base class.

RNPMessages

Ths Rotation, Nutation and Precession Model includes a set of messages intended to be used during execution of an RNP implementation in case of either a fatal error in the RNP or a warning associated with data in the model. These messages are intended to be used in conjunction with the JEOD Message Handling system [14].

The error messages included in the RNPMessages class are:

- initialization_error, an error in the first initialization of a PlanetRNP derived class or a PlanetRotation derived class,
- fidelity_error, a mismatch between the user requested fidelity of the RNP model and what is currently available to the framework, and

- setup_error, an error in a specific implementation of a PlanetRNP derived class that causes the generic framework to fail.

The warning messages included in the RNPMessages class are:

- polar_motion_table_warning, a warning that the user specified time for a polar motion interpolated data table which falls outside of the intended bounds of the interpolation table.

3.3.3 J2000 Specific Implementation

- RNPJ2000,
- PrecessionJ2000,
- NutationJ2000,
- NutationJ2000Init,
- RotationJ2000,
- PolarMotionJ2000, and
- PolarMotionJ2000Init

The following sections will describe each of these classes.

RNPJ2000

The RNPJ2000 class is a PlanetRNP derived class. It encapsulates the four classes that make up the individual transformations of the full J2000 RNP:

- PrecessionJ2000
- NutationJ2000
- RotationJ2000
- PolarMotionJ2000

These individual transformations are used to create the full J2000 RNP.

PrecessionJ2000

The PrecessionJ2000 class is a PlanetRotation derived class that implements the J2000 precession formulation as presented in Equation (3.8). This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the PrecessionJ2000 class should be in Julian Centuries from the standard epoch J2000 to the current date, in the T_{TT} time standard, as presented in the Mathematical Formulation.

NutationJ2000

The NutationJ2000 class is a PlanetRotation derived class that implements the J2000 nutation formulation as presented in Equation (3.11). This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the NutationJ2000 class should be in Julian Centuries from the standard epoch J2000 to the current date, in the T_{TT} time standard, as presented in the Mathematical Formulation.

NutationJ2000Init

The NutationJ2000Init class is a PlanetRotationInit derived class intended to initializae a NutationJ2000 object. It is only a container class for data, and has no direct functions. It contains data elements associated with the calculation of the J2000 implementation of nutation, as presented in Equation (3.11). This data is then meant to be copied into the NutationJ2000 object, and the NutationJ2000Init object is no longer used.

RotationJ2000

The RotationJ2000 class is a PlanetRotation derived class that implements the J2000 rotation formulation as presented in the Mathematical Formulation section. This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the RotationJ2000 class should be in seconds since the standard epoch J2000, in the T_{GMST} time standard.

PolarMotionJ2000

The PolarMotionJ2000 class is a PlanetRotation derived class that implements the J2000 polar motion formulation as presented in the Mathematical Formulation section. This formulation is an interpolation table, executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the PolarMotionJ2000 class should be the modified julian date, in the T_{UT1} time standard.

PolarMotionJ2000Init

The PolarMotionJ2000Init class is a PlanetRotationInit derived class intended to initialize a PolarMotionJ2000 object. It is only a container class for data, and has no direct functions. It contains data elements associated with the calculation of the J2000 implementation of polar motion, as presented in the Mathematical Formulation section. This data is then meant to be copied into the PolarMotionJ2000 object, and the PolarMotionJ2000Init object is no longer used.

3.3.4 Mars Pathfinder Specific Implementation

- RNPMars,

- PrecessionMars,
- NutationMars, and
- RotationMars

The following sections will describe each of these classes.

RNPMars

The RNPMars class is a PlanetRNP derived class. It encapsulates the four classes that make up the individual transformations of the full Mars RNP:

- PrecessionMars
- NutationMars
- RotationMars

These individual transformations are used to create the full Mars RNP.

PrecessionMars

The PrecessionMars class is a PlanetRotation derived class that implements the Mars precession formulation as presented in [8]. This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the PrecessionMars class should be in seconds from the standard epoch J2000 to the current date, in the T_{TT} time standard.

NutationMars

The NutationMars class is a PlanetRotation derived class that implements the Mars nutation formulation as presented in Equation [8]. This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the NutationMars class should be in seconds from the standard epoch J2000 to the current date, in the T_{TT} time standard.

RotationMars

The RotationMars class is a PlanetRotation derived class that implements the Mars rotation formulation as presented in the Mathematical Formulation section. This formulation is executed when the 'update' function is called, based on the last time set in the inherited 'set_time' function. The input variable to 'set_time' for the RotationMars class should be in seconds since the standard epoch J2000, in the T_{GMST} time standard.

3.3.5 Reference Manual

The complete API for the Rotation, Nutation and Precession Model can be found in the *Reference Manual* [2].

3.4 Inventory

All Rotation, Nutation, and Precession Model files are located in `${JEOD_HOME}/models/environment/RNP`. Relative to this directory,

- Model header and source files are located in model `include` and `src` subdirectories. See table **??** for a list of these configuration-managed files.
- Model documentation files are located in the model `docs` subdirectory. See table **??** for a list of the configuration-managed files in this directory.

Chapter 4

User Guide

The Instructions for Simulation Users section of the user guide is intended primarily for users of pre-existing simulations. It contains:

- A description of how to modify Rotation, Nutation and Precession Model variables after the simulation has compiled, including an in-depth discussion of the input file,
- An overview of how to interpret (but not edit) the S_define file,
- A sample of some of the typical variables that may be logged.

The Instructions for Simulation Developers section of the user guide is intended for simulation developers. It describes the necessary configuration of the Rotation, Nutation and Precession Model within an S_define file, and the creation of standard run directories. The latter component assumes a thorough understanding of the preceding Analysis section of the user guide. Where applicable, the user may be directed to selected portions of Product Specification (Chapter 3).

The Instructions for Model Developers section of the user guide is intended primarily for developers needing to extend the capability of the Rotation, Nutation and Precession Model. Such users should have a thorough understanding of how the model is used in the preceding Integration section, and of the model specification (described in Chapter 3).

Note that, for the simulation users and simulation developers sections, the Rotation, Nutation and Precession Model generic framework will not be discussed, as only a specific implementation of the framework is intended to be used for these endeavors. Similarly, only the generic frameworks will be discussed in the extension section, as it is the part of the Rotation, Nutation and Precession Model intended to be used by a third party for implementation of new RNP models.

4.1 Instructions for Simulation Users

The Simulation Users and Simulation Developers sections will assume, for the purpose of illustration, S_define objects of the following form; note that both the J2000 and Mars implementations are included in this example code:

```
sim_object {

     environment/time: TimeUT1 time_ut1;
     environment/time: TimeTT time_tt;
     environment/time: TimeGMST time_gmst;

} time;

sim_object {

   dynamics/dyn_manager:     DynManager             dyn_manager;

} mngr;


sim_object {

   environment/RNP/RNPJ2000:           RNPJ2000 rnp
             (environment/RNP/RNPJ2000/data/rnp_j2000.d);
   environment/RNP/RNPJ2000:           NutationJ2000Init nut_init
             (environment/RNP/RNPJ2000/data/nutation_j2000.d);
   environment/RNP/RNPJ2000:           PolarMotionJ2000Init pm_init
             (environment/RNP/RNPJ2000/data/polar_motion/xpyp_daily.d);


   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.initialize(
      Inout DynManager& dyn_manager = mngr.dyn_manager);

   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.NJ2000.initialize(
      In PlanetRotationInit* init = &earth.nut_init);

   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.PMJ2000.initialize(
      In PlanetRotationInit* init = &earth.pm_init);

   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.update_rnp(
      In TimeTT & time_tt = time.time_tt,
      In TimeGMST & time_gmst = time.time_gmst,
      In TimeUT1 & time_ut1 = time.time_ut1 );

  (100.0, environment) environment/RNP/RNPJ2000:
  earth.rnp.update_rnp(
     In TimeTT & time_tt = time.time_tt,
     In TimeGMST & time_gmst = time.time_gmst,
     In TimeUT1 & time_ut1 = time.time_ut1 );

  (0.01, environment) environment/RNP/RNPJ2000:
  earth.rnp.update_axial_rotation(
      In TimeGMST & time_gmst = time.time_gmst );

} earth;


sim_object {

   environment/RNP/RNPMars:        RNPMars             rnp
      (environment/RNP/RNPMars/data/rnp_mars.d);


   (initialization) environment/RNP/RNPMars:
   mars.rnp.initialize(
      Inout DynManager& dyn_manager = mngr.dyn_manager);

   (initialization) environment/RNP/RNPMars:
   mars.rnp.update_rnp (
      In TimeTT & time_tt = time.tt );

   (100.0, environment) environment/RNP/RNPMars:
   mars.rnp.update_rnp (
      In TimeTT & time_tt = time.tt );

   Idynamics (derivative) environment/RNP/RNPMars:
   mars.rnp.update_axial_rotation(
      In TimeTT & time_tt = time.tt );

} mars;
```

Note that this code is only representative of objects necessary for this discussion, and does not hold a complete implementation. For full implementation details on the Time model, please see the JEOD v5.1 Time Representations Model documentation [17]. For full implementation details on the DynManager model, please see the JEOD v5.1 DynManager documentation [5].

One variable that must be set by some means, whether via simulation input file or data file, is the name of the planet for which the RNP model will be updating the orientation. This variable must be set for both the J2000 and Mars implementations before the following functions, from the above code example, are called:

```
(initialization) environment/RNP/RNPJ2000:
earth.rnp.initialize(
   Inout DynManager& dyn_manager = mngr.dyn_manager);

(initialization) environment/RNP/RNPMars:
mars.rnp.initialize(
   Inout DynManager& dyn_manager = mngr.dyn_manager);
```

During this function call, the "earth.rnp" and "mars.rnp" objects will attain planet information from the
"mngr.dyn_manager" object, based on the variables "earth.rnp.name" and "mars.rnp.name". These variables are standard C strings in the form of 'char' pointers, and they default to NULL. Thus, the names MUST be set for successful simulation execution. These strings must each exactly match the names of their respective planets as set in the "mngr.dyn_manager" object, or a failure error issued through the MessageHandler [14] will result, and the RNP initialization will not be correctly completed.

Often the names of the planets will be set with the first letter capitalized, such as "Earth" or "Mars". Setting these names for both the J2000 and Mars RNP models would be accomplished with the following input or data file snippet:

```
earth.rnp.name = "Earth";
mars.rnp.name = "Mars";
```

The second initialization option for the JEOD v5.1 J2000 and Mars RNP models is the desired fidelity of the RNP representation. There are three options for this fidelity:

- FullRNP, which gives a full fidelity RNP with all components fully computed,
- RotationOnly, which holds polar motion, nutation and precession as identity and allows rotation to spin, from an initial identity matrix, at a constant rate set by the user, and
- ConstantNP, similar to RotationOnly but will take the first nutation and precession matrices set by the user and use those throughout the complete simulation.

The RotationOnly and ConstantNP options are, obviously, very simplified versions of the RNP model and should only be used in very special cases where low fidelity (and thus accuracy) of the RNP matrix is acceptable. For most users, the FullRNP option should be used.

In the above example, this option can be set in the following manner:

```
earth.rnp.rnp_type = PlanetRNP::FullRNP;
mars.rnp.rnp_type = PlanetRNP::FullRNP;
```

Note that PlanetRNP::FullRNP is default value of this option if no value is provided by the user.

Another option given to the user for the J2000 RNP model is associated with the polar motion module. It is a boolean that either enables or disables the polar motion module. If this boolean is set to true, the polar motion is calculated and included in the complete RNP transformation as described in the mathematical formulation section of this document. If this boolean is set to false, the polar motion is not calculated, and the polar motion portion of the RNP transformation will remain identity and not be included in the full calculation. In the example above, this option can be set (using a boolean true/false) using the following command:

```
earth.rnp.enable_polar = true;
```

Note that the Mars RNP model does not include polar motion modeling as of the version 2.2 release of JEOD, because a good model of Martian polar motion has not yet been developed. Should one be developed in the future, it could easily be added.

For the J2000 RNP model, the final option given to the user gives control over the variables used in the polar motion calculation. While the variable “earth.rnp.enable_polar” seen above turns the polar motion transformation either on or off, finer control is given to the user directly through the polar motion object. Recall that, as stated in the Mathematical Formulation section of this document, the polar motion as implemented in JEOD v5.1 is based on an interpolated table lookup. This table lookup can be overridden and the values set directly by the user. Overriding the table lookup would, in the above example, be done through the following calls:

```
earth.rnp.PMJ2000.override_table = true;
earth.rnp.PMJ2000.xp = USER_DEFINED_XP;
earth.rnp.PMJ2000.yp = USER_DEFINED_YP;
```

Note that the default units of xp and yp are radians. While Trick will automatically convert from other Trick supported units for angular displacement, users of this software module outside of Trick should be wary of the units expected by the JEOD Rotation, Nutation and Precession Model.

From an analytic user’s perspective, the main outputs from the Rotation, Nutation and Precession Model are the individual transformation matrices of rotation, nutation, precession and polar motion, as well as the total transformation from planet fixed inertial coordinates to planet centered planet fixed coordinates. The rotation, nutation, precession and polar motion matrices can be obtained, in the example above, through the following calls:

```
earth.rnp.RJ2000.rotation; /* Rotation Matrix */
earth.rnp.NJ2000.rotation; /* Nutation Matrix */
earth.rnp.PJ2000.rotation; /* Precession Matrix */
earth.rnp.PMJ2000.rotation; /* Polar Motion Matrix */

mars.rnp.RMars.rotation; /* Rotation Matrix */
mars.rnp.NMars.rotation; /* Nutation Matrix */
mars.rnp.PMars.rotation; /* Precession Matrix */
```

The generic RNP object also stores off the ’NP’ quantity, defined as:

$$\mathbf{NP} = \mathbf{N}^{\mathbf{T}} * \mathbf{P}^{\mathbf{T}} \tag{4.1}$$

This matrix is re-stored every time precession and nutation are calculated. This can be accessed, in the above example, through the calls:

```
earth.rnp.NP_matrix;
mars.rnp.NP_matrix;
```

As noted in the Detailed Design section of this document, the complete transformation is not stored directly in the Rotation, Nutation and Precession Model architecture. It is, instead, stored in the reference frame system kept in the DynManager object passed to the RNPJ2000 and RNPMars objects at initialization. The specific transformation will be stored in the Planet object, inside of the DynManager, with the name that matches the name set in the RNPJ2000 and RNPMars objects before initialization. Explicit information on accessing this transformation matrix can be found in the DynManager documentation [5], as well as the documentation for the Planet object [12].

4.2 Instructions for Simulation Developers

This section will use the same example S_define found in the Analysis section. Please again note that this code is only representative of objects necessary for this discussion, and does not hold a complete implementation. For full implementation details on the Time model, please see the JEOD v5.1 Time Representations Model documentation [17]. For full implementation details on the DynManager model, please see the JEOD v5.1 DynManager documentation [5].

To integrate the J2000 specific implementation of the generic RNP framework into a simulation, the following objects will be necessary:

- A TimeGMST object, to update the J2000 rotation module,
- A TimeTT object, to update the J2000 nutation and precession modules,
- A TimeUT1 object, to update the J2000 polar motion module,
- A DynManager object, which will contain the Planet object for which the RNPJ2000 object will update the ECI to ECEF transformation,
- An RNPJ2000 object,
- A NutationJ200Init object, and a
- A PolarMotionJ2000Init object.

To integrate the Mars RNP specific implementation of the generic RNP framework into a simulation, the following objects will be necessary:

- A TimeTT object, to update the rotation, nutation, and precession modules,
- A DynManager object, which will contain the Planet object for which the Mars RNP will update the MCI to MCMF transformation, and
- An RNPMars object.

As noted above, all of the Time classes must be correctly initialized and set up using a TimeManager object. This setup is outside of the scope of this document, and further details can be found in the Time Representations Model documentation [17].

Similarly, the DynManager object must be correctly set up, and must contain Planet objects with names matching the ones to be set in the RNPJ2000 and RNPMars objects as discussed in the previous section. DynManager setup is outside of the scope of this document, and further details can be found in the Dynamics Manager documentation [5]. Additionally,

The main objects for the Rotation, Nutation and Precession Model are of type RNPJ2000, for representing Earth J2000 RNP, and RNPMars, for representing Mars Pathfinder RNP. These objects are instantiated in the above example S_define with the following:

```
environment/RNP/RNPJ2000:       RNPJ2000    rnp
             (environment/RNP/RNPJ2000/data/rnp_j2000.d);
environment/RNP/RNPMars:        RNPMars     rnp
             (environment/RNP/RNPMars/data/rnp_mars.d);
```

These declarations each include default data files. The RNPJ2000 data file contains the following:

```
RNPJ2000.name = "Earth";
RNPJ2000.rnp_type = PlanetRNP::FullRNP;
RNPJ2000.enable_polar = true;
RNPJ2000.planet_omega = 7.292115146706388e-5; /* FROM GEM-T1 Gravity model */
```

Note that this default data file sets the RNPJ2000 object to use full fidelity. The RNPMars data file contains similar settings:

```
RNPMars.name = "Mars";
RNPMars.rnp_type = PlanetRNP::FullRNP;
RNPMars.enable_polar = false;
RNPMars.planet_omega {d/day} = 350.891985303;
```

Since the rotation, nutation, and precession behaviors of Mars are all described with equations in the Pathfinder Mars RNP model, the number of parameters required to represent them is relatively small. Thus, the additional initialization data required for the RotationMars, PrecessionMars, and NutationMars objects are all contained in the single default data file already mentioned, eliminating the need for further initialization steps once that file is loaded. This is a clear difference between the Mars and J2000 RNP implementations, the latter of which requires several more steps in order for initialization to be complete.

The J2000 nutation and polar motion objects, by nature, have large amounts of data associated with them (nutation has the coefficients associated with the series, polar motion has the data for the interpolation table). This data is meant to be input and to never change, so its loading has been broken into separate classes, one for nutation and one for polar motion. The instantiation of these initialization objects in the example S_define is shown below:

```
environment/RNP/RNPJ2000:           NutationJ2000Init nut_init
          (environment/RNP/RNPJ2000/data/nutation_j2000.d);
environment/RNP/RNPJ2000:           PolarMotionJ2000Init pm_init
          (environment/RNP/RNPJ2000/data/polar_motion/xpyp_daily.d);
```

The default data file for the "nut_init" object should always be used for the initialization of the nutation portion of an RNPJ2000 object. This data is static and never expected to change for this implementation of RNP.

The default data file for the interpolation table contained in the polar motion object, however, is highly variable based on data range and frequency of data. In the above example, the default data file used is named:

```
xpyp_daily.d
```

This default file gives daily values for polar motion parameters based on the latest data available at the time of the most recent JEOD version release. If desired, a user can parse his/her own data file(s) using the Perl scripts:

```
$JEOD_HOME/models/environment/RNP/RNPJ2000/data/polar_motion/make_daily_file.pl
$JEOD_HOME/models/environment/RNP/RNPJ2000/data/polar_motion/make_monthly_file.pl
```

Instructions for data parsing can be found in the Perl scripts and in the header sections of the default data files.

These populated objects must now be used to initialize the nutation and polar motion objects contained within the RNPJ2000 object. This initialization is demonstrated in the following code snippet:

```
   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.NJ2000.initialize(
      In PlanetRotationInit* init = &earth.nut_init);

   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.PMJ2000.initialize(
      In PlanetRotationInit* init = &earth.pm_init);
```

Note that the "initialize" function is a polymorphic interface, and that the function input is of type PlanetRotationInit, whereas the inputs themselves are specifically of type NutationJ2000Init and PolarMotionJ2000Init, respectively. The input must be of the correct type, and this will be checked when either function is invoked. If this type check fails, a failure message will be issued through the MessageHandler [14].

Once all default data has been loaded by whichever method was required, the next step for both the RNPMars and RNPJ2000 objects is to initialize them using the DynManager object, as illustrated by the following pair of function calls:

```
   (initialization) environment/RNP/RNPJ2000:
   earth.rnp.initialize(
      Inout DynManager& dyn_manager = mngr.dyn_manager);

   (initialization) environment/RNP/RNPMars:
   mars.rnp.initialize(
      Inout DynManager& dyn_manager = mngr.dyn_manager);
```

This initialization will cause both the RNPJ2000 object and the RNPMars object to each search the DynManager for a Planet with a name that matches its own previously set internal name. Upon finding a match, each will cache a pointer to their corresponding Planet object and update their own orientation state. Additionally, the RNPJ2000 and RNPMars objects will be registered with the DynManager as EphemerisInterface instances, and the internal ephemeris objects for each (inherited from PlanetOrientation) will be properly set up.

As has been stressed throughout this document, it is imperative that the DynManager object has both been correctly set up, and contains Planet objects with names matching those specified in the "earth.rnp.name" "and mars.rnp.name" variables; otherwise, a failure error will be issued through the MessageHandler [14].

For both RNPMars and RNPJ2000, the complete RNP must now be initialized from the set of Time Representations Model objects applicable to each. These calls, in initialization form, are represented in the example code snippet as:

```
(initialization) environment/RNP/RNPJ2000:
earth.rnp.update_rnp(
   In TimeTT & time_tt = time.time_tt,
   In TimeGMST & time_gmst = time.time_gmst,
   In TimeUT1 & time_ut1 = time.time_ut1 );

(initialization) environment/RNP/RNPMars:
mars.rnp.update_rnp(
   In TimeTT & time_tt = time.tt );
```

For each, the function "update_rnp" calculates all components of the RNP model, as dictated by the fidelity set by the user (and as described in the Analysis section). The function also updates the 'NP_matrix' variable found within the RNPJ2000 and RNPMars classes, and it stores the full RNP matrix in the correct Planet object for each, contained in the DynManager supplied at initialization. Note that these "update_rnp" initialization calls must be made after all Time Representations Model objects have been initialized, or incorrect RNP results can occur.

Also note that it is necessary to call 'update_rnp' at initialization before there is any expectation of 'update_ephem' being called on either the RNPJ2000 or RNPMars objects through the EphemeridesManager with which they were registered (i.e. the DynManager passed into their initialization functions.) This is a necessity because the 'update_rnp' function caches off necessary time information from its given arguments that are required for 'update_ephem' to run. If this requirement is not met, a message will be sent and the RNP will not be updated. For more information on the EphemeridesManager and the calling of 'update_ephem', see the appropriate documentation [16].

The last set of functions to call have to do with updating the RNP throughout the run-time of the sim. Examples of this for both RNPJ2000 and RNPMars are:

```
(100.0, environment) environment/RNP/RNPJ2000:
earth.rnp.update_rnp(
   In TimeTT & time_tt = time.time_tt,
   In TimeGMST & time_gmst = time.time_gmst,
   In TimeUT1 & time_ut1 = time.time_ut1 );

(0.01, environment) environment/RNP/RNPJ2000:
earth.rnp.update_axial_rotation(
   In TimeGMST & time_gmst = time.time_gmst );


(100.0, environment) environment/RNP/RNPMars:
mars.rnp.update_rnp (
   In TimeTT & time_tt = time.tt );

(0.01, environment) environment/RNP/RNPMars:
mars.rnp.update_axial_rotation(
   In TimeTT & time_tt = time.tt );
```

The "update_rnp" and "update_axial_rotation" functions are provided directly to the user for updating components of the RNP matrices. In this example, the two functions being called as scheduled jobs for both RNPJ2000 and RNPMars—one at a slow rate and one at a much higher rate for each. The 'update_rnp' function, as stated above, updates the full RNP module including rotation, precession, nutation and polar motion (if applicable). Because updating all components of the model is time intensive, and because of the extremely slow-moving nature of the non-rotation parts of the RNP, a second function is supplied to the user that will only update the axial rotation portion (the 'rotation' transformation) and will use the previously calculated nutation, precession and polar motion transformations to complete the RNP transformation. This is acceptable as the rate of change of the axial rotation portion of RNP is much greater than that of the nutation, precession and polar motion transformations. Thus, updating axial rotation at a much higher rate than the full RNP results in a good compromise between accuracy and efficiency when using either RNPJ2000 or RNPMars.

Also note that often in simulations, the "update_axial_rotation" function is called at the derivative rate instead of as a scheduled job. This is done so its effects are considered at every state calculation during integration. This can increase simulation accuracy, giving better orbital trajectories, and is a highly recommended practice.

In this example, “update_axial_rotation” is explictly being called by the user. However, it is also possible to allow the EphemeridesManager portion of the instantiated DynManager to take responsibility for updating the axial rotation of the Earth, by virtue of the fact that both RNPJ2000’s and RNPMars’s implementations of “ephem_update” wrap function “update_axial_rotation.” The axial portion of the RNP will then be updated appropriately through the DynManager, as described in the Ephemerides Model documentation [16]. A few things should be noted by users of this model:

- The EphemeridesManager will only be updating the axial rotation portion of the RNP, as that is the function that is wrapped by “ephem_update.” This leaves the calling of the full “rnp_update” function to the user at their desired rate.
- The rate at which “ephem_update” is automatically called is determined by the setup of the EphemeridesManager in question, and care should be taken that it is being done at the appropriate rate for the user’s needs.

Note that this makes the “update_axial_rotation” function unnecessary in many instances. However, it has been left in both places for flexibility and backward compatability reasons. Also note that all RNP update methods prevent duplicate, consecutive updates from being done, meaning that if an RNP update method receives the same input time twice in a row, it will automatically use the previous calculations. This is, of course, done for efficiency reasons, and allows both the “update_axial_rotation” function and the “ephem_update” function to be called at the same time step with little to no loss in efficiency.

4.3 Instructions for Model Developers

4.3.1 PlanetOrientation Extension

This section will discuss the manner in which the PlanetOrientation generic framework is intended to be extended. As PlanetOrientation inherits from both RefFrameOwner [15] and EphemerisInterface [16], the respective documentation for these models is invaluable.

Note that PlanetRNP, RNPJ2000, and RNPMars are each specific implementations of the PlanetOrientation base class, and thus serve as a partial, abstract implementation example, and two concrete examples, respectively.

Two distinct tasks must be completed to extend the PlanetRNP class:

- implementing the specific planet orientation model in question, and
- fulfilling the obligations of inheriting from EphemerisInterface.

How to implement the specific planet orientation model (meaning the mathematical formulation) is left to the user. The user can either implement the orientation model completely in the inherited virtual function ‘ephem_update’ (described in full below) or the user can create a separate function with a stand alone interface, then wrap this in 'ephem_update.' Note that the RNPJ2000 class does the latter, implementing both 'update_rnp' and 'update_axial_rotation' with separate interfaces and wrapping the axial version in 'ephem_update.'

The following functions are pure virtual in EphemerisInterface and are left unimplemented in PlanetOrientation. This requires concrete implementations for any class inheriting from PlanetOrientation that is intended to be instantiable. The functions are:

- get_name, and
- ephem_update,
- timestamp.

These functions will be described in the following sections.

get_name

'get_name', as it implies, returns the user determined name of the ephemeris interface in the form of a constant pointer to a character string, and should be implemented as such.

ephem_update

'ephem_update' is the meat of the ephemeris interface. It represents the main component of the ephemeris interface, the one that will be called whenever the EphemeridesManager's 'update_ephemerides' is called [16].

'ephem_update' can represent any update of the planetary orientation the user wishes. The two most popular options are:

- the full and complete update of the planet orientation, and
- the partial, high rate component of the planet orientation model, such as the 'update_axial_rotation' function present in PlanetRNP and its child classes.

The first option is particularly applicable when there will only be one update function. The second option is popular when multiple update functions are used, especially when one is intended to be called at a high or derivative rate, and the other is called at a much lower rate, behavior exhibited in the PlanetRNP and its child classes. RNPJ2000, for example, directly calls its already implemented 'update_axial_rotation' function as part of its 'ephem_update' function.

Note that the 'ephem_update' function does not take any direct arguments. Since many planet orientation models depend on knowledge of time, the implementer of a PlanetOrientation child class is often required to create and call a function that either caches off a pointer to an applicable JEOD time object [17] or requests the necessary time objects from a given TimeManager object. This allows the argument-less 'ephem_update' to have knowledge of the necessary input at all times, as long as the TimeManager is being correctly updated. This is demonstrated in the function ‘RNPJ2000::update_rnp’, which caches off a pointer to a TimeGMST object necessary for RNPJ2000’s implementation of ‘ephem_update.’ Note that this technique can be used for any necessary input to a planet orientation class.

Users should note that, whenever they update the reference frame state referred to by ‘planet_rot_state’, they should time tag it with the current dynamic time of the simulation, accessible through the TimeDyn object present in every TimeManager driven simulation. Details on this time stamping and the JEOD time model can be found in the Reference Frames documentation [15] and the Time Model documentation [17].

Additionally, the PlanetOrientation base class contains an ‘active’ flag, as well as an EphemerisOrientation object named ‘orient_interface.’ The ‘active’ flag should be checked before updating the model. The EphemerisOrientation object also indicates if this particular ephemeris object should be active through the ‘is_active’ function, as described in the Ephemerides documentation [16]. This flag should also be checked when implementing the ’ephem_update’ function.

timestamp

‘timestamp’ returns the time of the last update through the ‘ephem_update’ function. This timestamp is ephemeris specific, and requires the user to have access to whatever appropriate time was used during the last update.

4.3.2 PlanetRNP extension

This section will discuss the manner in which the PlanetRNP generic framework is intended to be extended for other specifications of RNP, including examples of implementation from the J2000 specific implementation.

Note that, because the base class of PlanetRNP is PlanetOrientation, all work described in the previous section must also be done when implementing a PlanetRNP child class.

The RNP Generic Framework supplies three base classes for use in implementing a new formulation for RNP:

- PlanetRotation, a class intended to be extended to represent specific implementations of precession, nutation, rotation and polar motion,
- PlanetRotationInit, a class that may be extended to initialize a specific implementation of the PlanetRotation class, and
- PlanetRNP, the main class that encapsulates the entire RNP model.

4.3.3 PlanetRotation Extension

The PlanetRotation class is meant to provide a single matrix based on time as an independent variable, usually associated with a single part of the RNP model (precession, nutation rotation and polar motion). This class is meant to be inherited from and extended through overriding certain virtual functions. There are multiple examples of extending this class in the J2000 specific implementation, including:

- PrecessionJ2000,
- NutationJ2000,
- RotationJ2000,
- PolarMotionJ2000

Implementation of a specific version of PlanetRotation is simple. The inheriting class must merely override the following virtual function with its own implementation for calculating its specific transformation matrix:

```
virtual void update_rotation();
```

Thus, all that must be added is a transformation specific implementation of this "update_rotation()" function, as well as any required data fields associated with this calculation.

It should also be noted that, in order to facilitate the calculation of a generic transformation matrix, the PlanetRotation class has a built in function of the following form:

```
virtual void update_time(double time);
```

where the input parameter "double time" is saved to the PlanetRotation member data field "double current_time". This gives an avenue for the current time that this particular transformation matrix should be calculated from. "update_time" is intended to be called from within a class inheriting from PlanetRNP, directly before the "update_rotation" function of the new class inheriting from PlanetRotation.

This method of updating time, of course, assumes that the new PlanetRotation inherited class has an output transformation matrix dependent on a single time variable, which can be represented in a "double" format. If this is not the case, the extender can also add their own function for updating the independent variables of the output transformation matrix. Anything that the new PlanetRotation inherited class needs in order to calculate its output transformation matrix can be set in this way.

4.3.4 PlanetRotationInit Extension

A PlanetRotationInit derived class is not necessary for all PlanetRotation derived classes. It is only intended to be used if there is a large amount of non-changing data contained within the PlanetRotation derived class, with the most obvious examples being nutation and polar motion objects. Two examples of such a class can be seen in the J2000-specific implementation of the generic RNP framework. These classes are:

- NutationJ2000Init, and

- PolarMotionJ2000Init.

If the extender does wish to use the PlanetRotationInit functionality, two steps must be taken. The first is the extender must create a new class, inheriting from PlanetRotationInit and adding any data that is necessary for the initialization of the PlanetRotation. It is intended that this data will be set in the PlanetRotationInit derived class, through a default data file as seen in the Integration section of this document.

Secondly, the extender must override the following function, inherited from the PlanetRotation base class:

```
virtual void initialize(PlanetRotationInit* init);
```

This function executes the user defined initialization routine on the invoking PlanetRotation derived object, using the information found in the PlanetrotationInit derived object 'init'. Examples of such a function can be found in both the NutationJ2000 class and the PolarMotionJ2000 class.

It should be noted that this function is intended to take in a pointer of a PlanetRotationInit derived class through a PlanetRotationInit pointer, utilizing polymorphism. As a result, the first step that must be taken is to check that the correct type of PlanetrotationInit derived class has been supplied to the function. Both the NutationJ2000 class and the PolarMotionJ2000 class do this using dynamic casting. The following code snippet demonstrates this, implicitely utilizing the JEOD Message Handling functionality [14]:

```
NutationJ2000Init* nut_init = dynamic_cast<NutationJ2000Init*>(init);

if(nut_init == NULL){
   // SEND A FAILUE MESSAGE THROUGH THE JEOD MESSAGE HANDLER
}
```

Since the 'dynamic_cast' call will, of course, return a NULL pointer if the 'init' object is not the correct type, this will result in only objects of the correct type being used for initialization.

4.3.5 PlanetRNP Extension

The final class to extend is the PlanetRNP class. This is the class intended to contain the four individual rotations that make up the complete RNP, and to serve as the main interface to both the user and the Dynamics Manager [5]. An example of this particular extension is the RNPJ2000 class.

The PlanetRNP class gives basic functionality for the creation of the complete RNP transformation from generic rotation, nutation, precession and polar motion transformations. It also gives functionality for interfacing with the previously mentioned Dynamics Manager class, as described in the Analysis and Integration sections. A person who wishes to create a new model based on different RNP theory can leverage this functionality by inheriting from the PlanetRNP class.

The extender must first create PlanetRotation derived classes for their particular RNP model's representation of precession, nutation, rotation and polar motion. Information on this can be found in the previous section on extending the PlanetRotation class.

Next, the extender must create a PlanetRNP derived class. This class should contain instances of the four extender PlanetRotation derived classes representing the individual transformations of the new RNP. For example, the RNPJ2000 class contains members of the following form (note that this is only a representative code snippet):

```
class RNPJ2000 : public PlanetRNP {

public:

   RotationJ2000 RJ2000;
   NutationJ2000 NJ2000;
   PrecessionJ2000 PJ2000;
   PolarMotionJ2000 PMJ2000;

};
```

These four classes are the only additional data requirements on the extender, although any other required data can be added if necessary.

The PlanetRNP class contains four pointers of PlanetRotation type, intended to be set to the appropriate PlanetRotation derived object in a PlanetRNP class. For example, RNPJ2000 does this during the constructor, where the nutation, precession, polar_motion and rotation pointers are set to the appropriate, specific J2000 version of those transformations. This can be seen in the following code snippet:

```
RNPJ2000::RNPJ2000() // Return: -- void
{
   // Assign pointer for polymorphic functionality.
   nutation = &this->NJ2000;
   precession = &this->PJ2000;
   polar_motion = &this->PMJ2000;
   rotation = &this->RJ2000;
}
```

This will enable the base functionality contained in PlanetRNP to use the extender-specified PlanetRotation objects through their polymorphic interface.

If the extender finds that additional initalization is necessary for their particular implemenation of the RNP framework, they can override the following function found in PlanetRNP:

```
virtual void initialize(DynManager& manager);
```

For example, the rotation transformation associated with the J2000 RNP needs knowledge of the J2000 nutation object; this association is done during the RNPJ2000 specific implementation of initialization.

If the extender wishes to implement their own version of this initialize function, it is absolutely necessary that they, at some point during their own initialize function, make a call to its base classe's version of initialize, if it exists. This will most commonly be a call to the initialize function found in PlanetRNP, which can be accomplished through the following command:

```
PlanetRNP::initialize(manager);
```

If this is not done then the relationship between the PlanetRNP derived class and the Dynamics Manager will not be created, and the PlanetRNP derived class will not work correctly.

The last step is to create functions in the PlanetRNP derived class for calls to update the complete RNP, and to update the rotation portion of the RNP only. To maximize flexibility the form of these calls has been left to the end user, as different formulations of RNP need different independent variables (often time standards contained within the JEOD v5.1 Time Representations Model [17]) in order to calculate the RNP transformation. Examples of both of these functions are found in the RNPJ2000 class, which contains:

```
void update_rnp(TimeTT& time_tt, TimeGMST& time_gmst, TimeUT1& time_ut1);

void update_axial_rotation(TimeGMST& time_gmst);
```

Sending in the TimeTT, TimeGMST and TimeUT1 object allows the RNPJ2000 object to calculate the complete RNP; conversely, only the TimeGMST object is necessary to update the axial rotation.

The 'update_rnp' function must contain the correct logic to update the independent variables for all four individual transformations of the RNP. This method for updating the independent variable should have been established in some form during the extension of the PlanetRotation classes as described above. Once all independent variables have been set, the user must only make a call to the PlanetRNP 'update_rnp' function, using the following command:

```
PlanetRNP::update_rnp();
```

This will automatically update all pieces of the RNP from the most recent independent variables, create the complete RNP transformation, and set the correct orientation in the Dynamics Manager associated with the RNP object.

Similarly, the 'update_axial_rotation' function must contain the correct logic to update the independent variables of the particular axial rotation associated with that particular PlanetRNP derived class. Once this is done, the extender can call the base class function 'update_axial_rotation' using the following command:

```
PlanetRNP::update_axial_rotation();
```

This will then update the complete RNP transformation with the new axial rotation component, and update it in the Dynamics Manager associated with the RNP object.

Chapter 5

Verification and Validation

5.1 Verification

Inspection RNP_1: Top-level inspection

This document structure, the code, and associated files have been inspected, and together satisfy requirement RNP_1.

Inspection RNP_2: Data Requirements Inspection

The Rotation, Nutation and Precession Model has been inspected, and contains all attributes necessary to satisfy requirements RNP_2 and RNP_3.

Inspection RNP_3: Mars Functional Requirement Inspection

The Rotation, Nutation and Precession Model has been inspected, and contains all attributes necessary to satisfy requirement RNP_5.

Inspection RNP_4: Planet Orientation Extensibility Inspection

The Rotation, Nutation and Precession Model has been inspected, and contains the functionality necessary to satisfy the requirement RNP_6. Additionally, PlanetRNP and the J2000 and Mars specific implementations of PlanetOrientation are examples of this extension.

Inspection RNP_5: RNP Extensibility Inspection

The Rotation, Nutation and Precession Model has been inspected, and contains the functionality necessary to satisfy the requirement RNP_7. Additionally, the J2000 and Mars specific implementations of the generic framework are examples of this extension.

5.2 Validation

The validation tests for the Rotation, Nutation and Precession Model use the following for benchmark data:

1. *Fundamentals of Astrodynamics and Applications, Second Edition* [18]
2. *Satellite Orbits* [11]
3. *Naval Observatory Vector Astrometry Subroutines* [7]

In the following reference is made to the number of significant digits in a test that matches the benchmark. This refers to the published number of significant digits in the primary source. It is probable that the quantities under consideration are correct to additional decimal places.

Note, as mentioned in the Mathematical Formulation section of this document, that the benchmark data uses a different, older formulation of RNP as seen in [1], which results in transposed matrices for precession, rotation, nutation and polar motion from those seen in [19]. As a result, the benchmark data for precession, rotation, nutation and polar motion will be transposed from that supplied by the JEOD v5.1 J2000 RNP model. However, as already noted, these formulations are otherwise identical, and the final transformation from the ECI to the ECEF frame will be identical.

Test RNP_1: RNP Matrices Initialization

a. Objective:
The objective of this test is to validate the initialization of the Rotation, Nutation and Precession Model matrices generated by the Rotation, Nutation and Precession Model math model.

This is done by comparison of a benchmark matrix from a known validated source. The rotation, nutation, precession and polar motion transformations are generated in this test.

This test demonstrates the partial satisfaction of the requirement RNP_4.

b. Initialization:

```
Initial conditions:
 1999/03/04  00:00:00.000 UTC
deltaAT = 32.0 seconds
DUT1    = 0.649232
Polar Offset in arcseconds:
  xp = 0.06740
  yp = 0.24173
```

c. Procedure:
Input initial time in UTC time and generate initial UT1, GMST and TT times, and compute the initial rotation, nutation, precession and polar motion matrices.

The example below is from *Montenbruck and Gill* [11] and conforms to the International Earth Rotation Service models. EOP Data used: IERS Bulletins B (No. 135) and C(No.16)

d. Pass/Fail Criterion:

Reference NOVAS [7] IERS routines give:

IAU 1976 Precession Matrix

```
 +0.99999998 +0.00018581 +0.00008074
 -0.00018581 +0.99999998 -0.00000001
 -0.00008074 -0.00000001 +1.00000000
```

IAU 1980 Nutation Matrix

```
 +1.00000000 +0.00004484 +0.00001944
 -0.00004484 +1.00000000 +0.00003207
 -0.00001944 -0.00003207 +1.00000000
```

Earth Rotation Matrix

```
 -0.94730417 +0.32033547 +0.00000000
 -0.32033547 -0.94730417 +0.00000000
 +0.00000000 +0.00000000 +1.00000000
```

Polar Motion Matrix

```
 +1.00000000 +0.00000000 +0.00000033
 +0.00000000 +1.00000000 -0.00000117
 -0.00000033 +0.00000117 +1.00000000
```

RNP Transformation Matrix (Polar Motion included)

```
 -0.94737803 +0.32011696 -0.00008431
 -0.32011696 -0.94737803 -0.00006363
 -0.00010024 -0.00003330 +0.99999999
```

Test must be good to published benchmark significant digits.

e. Results: From the SIM RUN

JEOD IAU 1976 Precession Matrix (Transpose)

```
  0.99999998  0.00018581  0.00008074
 -0.00018581 0.99999998  -0.00000001
 -0.00008074 -0.00000001  0.99999999
```

JEOD IAU 1980 Nutation Matrix (Transpose)

```
  0.99999999  0.00004484  0.00001944
 -0.00004484  1.00000000  0.00003207
 -0.00001944 -0.00003207 0.99999999
```

JEOD Earth Rotation Matrix (Transpose)

```
 -0.94730417  0.32033547  0.00000000
 -0.32033547 -0.94730417  0.00000000
  0.00000000 +0.00000000  1.00000000
```

JEOD Polar Motion Matrix (Transpose)

```
  1.00000000  0.00000000  0.00000033
  0.00000000  1.00000000 -0.00000117
 -0.00000033  0.00000117  1.00000000
```

JEOD RNP Transformation Matrix (Polar Motion included)

```
 -0.94737803  0.32011696 -0.00008431
 -0.32011696 -0.94737803 -0.00006363
 -0.00010024 -0.00003330  0.99999999
```

The R, N, P, and polar matrices are a perfect match.

f. Problems:
None

Test RNP_2: RNP Matrices Propagation

a. Objective:
The objective of this test is to validate the propagation RNP matrices generated by the RNP math model.

This is done by comparison of a benchmark matrix from a known validated source. The rotation, nutation, precession and polar motion transformations are generated in this test.

This test demonstrates the partial satisfaction of the requirement RNP_4.

b. Initialization:

```
Initial conditions:
 1999/03/03  00:00:00.000 UTC
deltaAT = 32.0 seconds
DUT1    = 0.649232
Polar Offset in arcseconds:
  xp = 0.06740
  yp = 0.24173
```

c. Procedure:
The date is set one day before the initialization case, propagate one day and compare to benchmark.

Example from *Montenbruck and Gill* [11] and conform to the International Earth Rotation Service models. EOP Data used: IERS Bulletins B (No. 135) and C(No.16)

d. Pass/Fail Criterion:
The benchmark data to comply to are:

Reference NOVAS [7] IERS routines give:

IAU 1976 Precession Matrix

```
+0.99999998 +0.00018581 +0.00008074
-0.00018581 +0.99999998 -0.00000001
-0.00008074 -0.00000001 +1.00000000
```

IAU 1980 Nutation Matrix

```
+1.00000000 +0.00004484 +0.00001944
-0.00004484 +1.00000000 +0.00003207
-0.00001944 -0.00003207 +1.00000000
```

Earth Rotation Matrix

```
-0.94730417 +0.32033547 +0.00000000
-0.32033547 -0.94730417 +0.00000000
+0.00000000 +0.00000000 +1.00000000
```

Polar Motion Matrix

```
+1.00000000 +0.00000000 +0.00000033
+0.00000000 +1.00000000 -0.00000117
-0.00000033 +0.00000117 +1.00000000
```

RNP Transformation Matrix (Polar Motion included)

```
-0.94737803 +0.32011696 -0.00008431
-0.32011696 -0.94737803 -0.00006363
-0.00010024 -0.00003330 +0.99999999
```

Test must be good to published benchmark significant digits.

e. Results:
From the SIM RUN propagated for one day.

JEOD IAU 1976 Precession Matrix (Transpose)

```
  0.99999998  0.00018581  0.00008074
 -0.00018581 0.99999998  -0.00000001
 -0.00008074 -0.00000001  0.99999999
```

JEOD IAU 1980 Nutation Matrix (Transpose)

```
  0.99999999  0.00004484  0.00001944
 -0.00004484  1.00000000  0.00003207
 -0.00001944 -0.00003207 0.99999999
```

JEOD Earth Rotation Matrix (Transpose)

```
 -0.94730417  0.32033547  0.00000000
 -0.32033547 -0.94730417  0.00000000
  0.00000000 +0.00000000  1.00000000
```

JEOD Polar Motion Matrix (Transpose)

```
  1.00000000  0.00000000  0.00000033
  0.00000000  1.00000000 -0.00000117
 -0.00000033  0.00000117  1.00000000
```

JEOD RNP Transformation Matrix (Polar Motion included)

```
 -0.94737803  0.32011696 -0.00008431
 -0.32011696 -0.94737803 -0.00006363
 -0.00010024 -0.00003330  0.99999999
```

The propagated times agree with published benchmarks and the R, N, P, and polar matrices are a perfect match.

f. Problems:
None

Test RNP_3: RNP Matrices Generation

a. Objective:
The objective of this test is to validate the initialization of the RNP matrices generated by the RNP math model.

This is done by comparison of a benchmark matrix from a known validated source. The rotation, nutation, precession and polar motion transformations are generated in this test. In this case, the polar offsets are selected from the interpolated table.

This test demonstrates the partial satisfaction of the requirement RNP_4.

b. Initialization:

```
Initial conditions:
 1999 03 04  00:00:00.000 UTC
earth.utime.in.override_DUT1    = No
earth.utime.in.override_deltaAT = No
earth.rnp.in.enable_polar = Yes
```

c. Procedure:
Input initial time in UTC time and generate initial UT1, TT and GMST time, compute the initial rotation, nutation, precession and polar offset matrices.

The below example is from *Montenbruck and Gill* [11] and conforms to the International Earth Rotation Service models. EOP Data used: IERS Bulletins B (No. 135) and C(No.16)

d. Pass/Fail Criterion:
The benchmark data to comply to are:

Reference NOVAS [7] IERS routines give:

IAU 1976 Precession Matrix

```
 +0.99999998 +0.00018581 +0.00008074
 -0.00018581 +0.99999998 -0.00000001
 -0.00008074 -0.00000001 +1.00000000
```

IAU 1980 Nutation Matrix

```
 +1.00000000 +0.00004484 +0.00001944
 -0.00004484 +1.00000000 +0.00003207
 -0.00001944 -0.00003207 +1.00000000
```

Earth Rotation Matrix

```
 -0.94730417 +0.32033547 +0.00000000
 -0.32033547 -0.94730417 +0.00000000
 +0.00000000 +0.00000000 +1.00000000
```

Polar Motion Matrix

```
 +1.00000000 +0.00000000 +0.00000033
 +0.00000000 +1.00000000 -0.00000117
 -0.00000033 +0.00000117 +1.00000000
```

RNP Transformation Matrix (Polar Motion included)

```
 -0.94737803 +0.32011696 -0.00008431
 -0.32011696 -0.94737803 -0.00006363
 -0.00010024 -0.00003330 +0.99999999
```

Benchmark must be good to published benchmark significant digits.

e. <u>Results</u>:
From the SIM RUN

```
earth.utime.in.deltaAT = 32
earth.utime.in.DUT1 = 0.649232
earth.rnp.out.gast = 2.815509062392805
```

JEOD IAU 1976 Precession Matrix (Transpose)

```
  0.99999998  0.00018581  0.00008074
 -0.00018581 0.99999998  -0.00000001
 -0.00008074 -0.00000001  0.99999999
```

JEOD IAU 1980 Nutation Matrix (Transpose)

```
  0.99999999  0.00004484  0.00001944
 -0.00004484  1.00000000  0.00003207
 -0.00001944 -0.00003207 0.99999999
```

JEOD Earth Rotation Matrix (Transpose)

```
 -0.94730417  0.32033547  0.00000000
 -0.32033547 -0.94730417  0.00000000
  0.00000000 +0.00000000  1.00000000
```

JEOD Polar Motion Matrix (Transpose)

```
  1.00000000  0.00000000  0.00000033
  0.00000000  1.00000000 -0.00000117
 -0.00000033  0.00000117  1.00000000
```

JEOD RNP Transformation Matrix (Polar Motion included)

```
 -0.94737803  0.32011696 -0.00008431
 -0.32011696 -0.94737803 -0.00006363
 -0.00010024 -0.00003330  0.99999999
```

-0.9417314416293043

The R, N, P, and polar motion matrices are a perfect match.

f. <u>Problems</u>:
None

Test RNP_4: Polar Offset Matrix Generation

a. Objective:
The objective of this test is to validate the initialization of the Polar offset matrix generated by the RNP math model with the benchmark.

In this case the time and polar offset flags were set to Off so that the deltaAT, DUT1 were selected from a table the polar off set model was turned off.

This test demonstrates the partial satisfaction of the requirement RNP_4.

b. Initialization:

```
Initial conditions:
 1999 03 04  00:00:00.000 UTC
earth.utime.in.override_DUT1    = No
earth.utime.in.override_deltaAT = No
earth.rnp.in.enable_polar = No
```

c. Procedure:
Input initial time in UTC and generate initial UT1, GMST and TT times, and compute the initial rotation, nutation, precession and polar offset matrices.

The below example is from *Montenbruck and Gill* [11] and conforms to the International Earth Rotation Service models. EOP Data used: IERS Bulletins B (No. 135) and C(No.16)

d. Pass/Fail Criterion:
The benchmark data to comply to are: Polar Motion Matrix

```
 +0.00000000 +0.00000000 +0.00000000
 +0.00000000 +0.00000000 -0.00000000
 -0.00000000 +0.00000000 +0.00000000
```

e. Results:
From the SIM RUN

```
earth.rnp.out.polar[0][0] =
 0,                     0,                     0
earth.rnp.out.polar[1][0] =
 0,                     0,                     0
earth.rnp.out.polar[2][0] =
 0,                     0,                     0
```

The polar motion matrix is null as expected.

f. Problems:
None

Test RNP_5: J2000 to ECEF Transformation Initialization

a. Objective:
The objective of this test is to validate the initialization of the transformation from inertial J2000 to Earth Centered Earth Fixed Coordinates (ECEF).

This is done by comparison of a benchmark inertial state vector and transformed ECEF vector to a validated source. The rotation, nutation, precession and polar motion transformations are generated in this test.

This test demonstrates the partial satisfaction of the requirement RNP_4.

b. Initialization:

```
Initial conditions:
 1991 April 6 7h 51 min 28.386009 sec UTC
deltaAT = 26 seconds
DUT1    = 0.402521 seconds

Polar Offsets selected from table.

J2000 inertial position vector {km}
Xj2000 = 5102.5096
Yj200  = 6123.01152
Zj2000 = 6378.1363
```

c. Procedure:
Input initial time in UTC and generate initial times in UT1, GMST and TT, compute the initial Rotation, Nutation, Precession and Polar Offset Matrices and rotate the inertial coordinates to the ECEF.

The below example is from *Vallado* [18] Example 3-6 Vallado page 225

d. Pass/Fail Criterion:
The benchmark data to comply with are:

```
gast(degrees)  311.977914290
```

Earth Centered Earth Fixed State Vector km

```
Xecef = -1120.598506
Yecef =  7894.483204
Zecef =   6374.07961
```

Benchmark must match the test data published in Vallado page 225 [18].

e. Results:
From the SIM RUN

```
earth.rnp.out.gast(degrees) = 311.977914352
Difference of about 0.000000062  degrees.
```

JEOD ECEF Vector km

```
Xecef = -1120.598531
Yecef =  7894.483203
Zecef =  6374.079608
diffs = .000025 , .000001 , .000008 meters
```

The gast angle differs by 0.00000006 degrees from the benchmark and the ECEF vectors differ by .00003 , .000001 and .000008 meters from the benchmark-these are very small differences.

f. Problems:
None

Special Note:
Subsequent communication with David Vallado clarified the differences shown above. The example 3-6 in [18] was computed using the The International Earth Rotation and Reference Systems Service (IERS), IERS 1996 model [10]. The IERS 96 models for precession and nutation are a refinement of the ITERS 1990 [4] models; however the differences in the final results are quite small as can be seen above.

Conclusion:
The mathematical modeling of the Rotation, Nutation, Precession and Polar Off-set rotation matrices match the benchmarks to the desired published accuracy. The code for the modeling is fully referenced and conforms to the mathematical modeling in the literature and references where it was published.

5.3 Requirements Traceability

Table 5.1 summarizes the inspections and tests that demonstrate the satisfaction of the requirements levied on the model.

Table 5.1: Requirements Traceability

Requirement	**Traces to**
RNP_1 Top-level	Insp. RNP_1 Top-level inspection
RNP_2 Earth RNP Data Requirement	Insp. RNP_2 Data Requirements Inspection
RNP_3 Mars RNP Data Requirement	Insp. RNP_2 Data Requirements Inspection
RNP_4 Earth RNP Functional Requirement	Test RNP_1 RNP Matrices Initialization
	Test RNP_2 RNP Matrices Propagation
	Test RNP_3 RNP Matrices Generation
	Test RNP_4 Polar Offset Matrix Generation
	Test RNP_5 J2000 to ECEF Transformation Initialization
RNP_5 Mars RNP Functional Requirement	Insp. RNP_3 Mars Functional Requirement Inspection
RNP_6 Planet Orientation Extensibility	Insp. RNP_4 Planet Orientation Extensibility Inspection
RNP_7 RNP Extensibility	Insp. RNP_5 RNP Extensibility Inspection

5.4 Metrics

Table 5.2 presents coarse metrics on the source files that comprise the model.

Table 5.2: Coarse Metrics

	Number of Lines			
File Name	**Blank**	**Comment**	**Code**	**Total**
Total	0	0	0	0

Table 5.3 presents the extended cyclomatic complexity (ECC) of the methods defined in the model.

Table 5.3: Cyclomatic Complexity

Method	**File**	**Line**	**ECC**
jeod::PlanetOrientation::(std:: set_name (std::string name_ in)	GenericRNP/include/planet_ orientation.hh	168	1
jeod::PlanetRotation::update_ rotation (void)	GenericRNP/include/planet_ rotation.hh	155	1
jeod::PlanetRotation:: initialize (PlanetRotation Init* init JEOD_UNUSED)	GenericRNP/include/planet_ rotation.hh	170	1
jeod::PlanetOrientation:: PlanetOrientation (void)	GenericRNP/src/planet_ orientation.cc	56	1
jeod::PlanetOrientation::~ PlanetOrientation (void)	GenericRNP/src/planet_ orientation.cc	71	1
jeod::PlanetOrientation:: initialize (DynManager& dyn_manager)	GenericRNP/src/planet_ orientation.cc	80	3
jeod::PlanetOrientation:: activate ()	GenericRNP/src/planet_ orientation.cc	138	1
jeod::PlanetOrientation:: deactivate ()	GenericRNP/src/planet_ orientation.cc	147	1
jeod::PlanetOrientation:: ephem_initialize (EphemeridesManager& manager JEOD_UNUSED)	GenericRNP/src/planet_ orientation.cc	156	1
jeod::PlanetOrientation:: ephem_activate (EphemeridesManager& manager JEOD_UNUSED)	GenericRNP/src/planet_ orientation.cc	171	1
jeod::PlanetOrientation:: ephem_build_tree (EphemeridesManager& manager JEOD_UNUSED)	GenericRNP/src/planet_ orientation.cc	189	1
jeod::PlanetRNP::PlanetRNP (void)	GenericRNP/src/planet_ rnp.cc	69	1
jeod::PlanetRNP::~PlanetRN P (void)	GenericRNP/src/planet_ rnp.cc	85	1
jeod::PlanetRNP::update_rnp (void)	GenericRNP/src/planet_ rnp.cc	95	7

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

Method	**File**	**Line**	**ECC**
jeod::PlanetRNP::update_axial_rotation (void)	GenericRNP/src/planet_rnp.cc	169	2
jeod::PlanetRNP::propagate_rnp (void)	GenericRNP/src/planet_rnp.cc	194	7
jeod::PlanetRotation::Planet Rotation (void)	GenericRNP/src/planet_rotation.cc	50	1
jeod::PlanetRotation::~Planet Rotation (void)	GenericRNP/src/planet_rotation.cc	73	1
jeod::PlanetRotation::update_time (double time)	GenericRNP/src/planet_rotation.cc	81	1
jeod::PlanetRotation::get_rotation (double rot[3][3])	GenericRNP/src/planet_rotation.cc	93	1
jeod::PlanetRotation::get_rotation_transpose (double rot[3][3])	GenericRNP/src/planet_rotation.cc	105	1
jeod::PlanetRotationInit::PlanetRotationInit (void)	GenericRNP/src/planet_rotation_init.cc	43	1
jeod::PlanetRotationInit::~PlanetRotationInit (void)	GenericRNP/src/planet_rotation_init.cc	52	1
jeod::NutationJ2000::NutationJ2000 (void)	RNPJ2000/src/nutation_j2000.cc	63	1
jeod::NutationJ2000::~NutationJ2000 (void)	RNPJ2000/src/nutation_j2000.cc	93	19
jeod::NutationJ2000::update_rotation (void)	RNPJ2000/src/nutation_j2000.cc	139	2
jeod::NutationJ2000::initialize (PlanetRotationInit* init)	RNPJ2000/src/nutation_j2000.cc	260	3
jeod::NutationJ2000Init::NutationJ2000Init (void)	RNPJ2000/src/nutation_j2000_init.cc	53	1
jeod::NutationJ2000Init::~NutationJ2000Init (void)	RNPJ2000/src/nutation_j2000_init.cc	73	10
jeod::PolarMotionJ2000::PolarMotionJ2000 (void)	RNPJ2000/src/polar_motion_j2000.cc	60	1
jeod::PolarMotionJ2000::~PolarMotionJ2000 (void)	RNPJ2000/src/polar_motion_j2000.cc	78	7
jeod::PolarMotionJ2000::update_rotation (void)	RNPJ2000/src/polar_motion_j2000.cc	99	8

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::PolarMotionJ2000::initialize (PlanetRotationInit* init)	RNPJ2000/src/polar_motion_j2000.cc	203	3
jeod::PolarMotionJ2000Init::PolarMotionJ2000Init (void)	RNPJ2000/src/polar_motion_j2000_init.cc	51	1
jeod::PolarMotionJ2000Init::~PolarMotionJ2000Init (void)	RNPJ2000/src/polar_motion_j2000_init.cc	68	4
jeod::PrecessionJ2000::PrecessionJ2000 (void)	RNPJ2000/src/precession_j2000.cc	44	1
jeod::PrecessionJ2000::~PrecessionJ2000 (void)	RNPJ2000/src/precession_j2000.cc	52	1
jeod::PrecessionJ2000::update_rotation (void)	RNPJ2000/src/precession_j2000.cc	60	1
jeod::RNPJ2000::RNPJ2000 (void)	RNPJ2000/src/rnp_j2000.cc	70	1
jeod::RNPJ2000::~RNPJ2000 (void)	RNPJ2000/src/rnp_j2000.cc	92	1
jeod::RNPJ2000::initialize (DynManager& dyn_manager)	RNPJ2000/src/rnp_j2000.cc	100	4
jeod::RNPJ2000::update_rnp (const TimeTT& time_tt, TimeGMST& time_gmst, const TimeUT1& time_ut1)	RNPJ2000/src/rnp_j2000.cc	135	10
jeod::RNPJ2000::update_axial_rotation (TimeGMST& time_gmst)	RNPJ2000/src/rnp_j2000.cc	236	7
jeod::RNPJ2000::timestamp ()	RNPJ2000/src/rnp_j2000.cc	308	1
jeod::RNPJ2000::get_name ()	RNPJ2000/src/rnp_j2000.cc	312	1
jeod::RNPJ2000::ephem_update ()	RNPJ2000/src/rnp_j2000.cc	317	4
jeod::RNPJ2000::get_dyn_time_ptr (TimeGMST& gmst)	RNPJ2000/src/rnp_j2000.cc	341	3
jeod::RotationJ2000::RotationJ2000 (void)	RNPJ2000/src/rotation_j2000.cc	57	1

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RotationJ2000::~RotationJ2000 (void)	RNPJ2000/src/rotation_j2000.cc	72	1
jeod::RotationJ2000::update_rotation (void)	RNPJ2000/src/rotation_j2000.cc	81	4
jeod::NutationMars::NutationMars (void)	RNPMars/src/nutation_mars.cc	61	1
jeod::NutationMars::~NutationMars (void)	RNPMars/src/nutation_mars.cc	82	4
jeod::NutationMars::update_rotation (void)	RNPMars/src/nutation_mars.cc	105	3
jeod::PrecessionMars::PrecessionMars (void)	RNPMars/src/precession_mars.cc	58	1
jeod::PrecessionMars::~PrecessionMars (void)	RNPMars/src/precession_mars.cc	75	1
jeod::PrecessionMars::update_rotation (void)	RNPMars/src/precession_mars.cc	85	2
jeod::PrecessionMars::compute_fixed_matrices (void)	RNPMars/src/precession_mars.cc	136	1
jeod::RNPMars::RNPMars (void)	RNPMars/src/rnp_mars.cc	65	1
jeod::RNPMars::~RNPMars ()	RNPMars/src/rnp_mars.cc	95	1
jeod::RNPMars::initialize (DynManager& dyn_manager)	RNPMars/src/rnp_mars.cc	102	4
jeod::RNPMars::update_rnp (TimeTT& time_tt)	RNPMars/src/rnp_mars.cc	137	8
jeod::RNPMars::update_axial_rotation (TimeTT& time_tt)	RNPMars/src/rnp_mars.cc	200	7
jeod::RNPMars::timestamp ()	RNPMars/src/rnp_mars.cc	255	1
jeod::RNPMars::get_name ()	RNPMars/src/rnp_mars.cc	264	1
jeod::RNPMars::ephem_update ()	RNPMars/src/rnp_mars.cc	273	4
jeod::RNPMars::get_dyn_time_ptr (TimeTT& time_tt)	RNPMars/src/rnp_mars.cc	297	3

Continued on next page

Table 5.3: Cyclomatic Complexity (continued)

Method	**File**	**Line**	**ECC**
jeod::RotationMars::RotationMars (void)	RNPMars/src/rotation_mars.cc	58	1
jeod::RotationMars::~RotationMars (void)	RNPMars/src/rotation_mars.cc	73	1
jeod::RotationMars::update_rotation (void)	RNPMars/src/rotation_mars.cc	82	4

Bibliography

[1] Victor R. Bond. The RNP Routine for the Standard Epoch J2000. Technical Report NASA:JSC-24574, National Aeronautics and Space Administration, 2101 NASA Parkway, Houston, Texas, 77058, September 1990.

[2] Generated by doxygen. *RNP Reference Manual*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2023.

[3] J. Cunningham. WGS84 Coordinate Validation and Improvement for the NIMA, NSWCDD TR-96 201. Technical report, Naval Surface Warfare Center, Dahlgreen Virginia, 1996.

[4] D.M. Dennis (ed). IERS Standards, IERS Technical Note 13. Technical report, Observatoire de Paris, Paris, France, 1992.

[5] Hammen, D. Dynamics Manager Model. Technical Report JSC-61777-dynamics/dyn_manager, NASA, Johnson Space Center, Houston, Texas, July 2023.

[6] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, July 2023.

[7] George H. Kaplan. US Naval Observatory Vector Astrometry Subroutines. Technical report, USNO, Washington,D.C.,http://aa.usno.navy.mil/software/novas/novas_info.html, 2001.

[8] Alex S. Konopliv. A global solution for the Mars static and seasonal gravity, Mars orientation, Phobos and Deimos masses, and Mars ephemeris. Technical report, Icarus, Volume 182, Issue 1, Pages 23-50, 2006.

[9] Alex S. Konopliv. Mars high resolution gravity fields from MRO, Mars season gravity, and other dynamical parameters. Technical report, Icarus, Volume 211, Issue 1, Pages 401-428, 2011.

[10] D.D. McCarthy and G. Petit. IERS Conventions (2003), IERS Technical Note 32. Technical report, Bundesamt, Franfkurt am Main, 2004.

[11] Montenbruck, O. and Gill, E. *Satellite Orbits*. Springer, Netherlands, 2005.

[12] Morris, J. Planet Model. Technical Report JSC-61777-environment/planet, NASA, Johnson Space Center, Houston, Texas, July 2023.

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

[14] Shelton, R. Message Handler Model. Technical Report JSC-61777-utils/message, NASA, Johnson Space Center, Houston, Texas, July 2023.

[15] Spencer, A. Reference Frame Model. Technical Report JSC-61777-utils/ref_frames, NASA, Johnson Space Center, Houston, Texas, July 2023.

[16] Thompson, B. Ephemerides Model. Technical Report JSC-61777-environment/ephemerides, NASA, Johnson Space Center, Houston, Texas, July 2023.

[17] Turner, G. Time Model. Technical Report JSC-61777-environment/time, NASA, Johnson Space Center, Houston, Texas, July 2023.

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

[19] D.A. Vallado with tech contributions by Wayne D. McClain. *Fundamentals of Astrodynamics and Applications, Third Edition.* Microcosm Press, El Segundo,Calif., 2007.