User Manual 4.14 Celestial bodies

User Manual 4.14 Celestial bodies

Introduction

Scope

The celestial bodies are described by their main features : position and geometry. The positions are ephemeris that must be loaded from models, the geometries are created as one axis ellipsoids or facet bodies. The package provides a factory able to create any celestial body of the solar system.

Javadoc

The classes for bodies description are available in the package `bodies` of Patrius.

Library	Javadoc
Orekit	Package fr.cnes.sirius.patrius.bodies
Patrius	Package fr.cnes.sirius.patrius.bodies

Links

Orekit bodies : Orekit Bodies architecture description

IAU report : Report of the IAU Working Group on Cartographic Coordinates and Rotational Elements: 2009

Useful Documents

None as of now.

Package Overview

None.

Features Description

Celestial bodies

The Moon, the Sun and planets of the solar system are all represented by the CelestialBody interface. This class associates a name (eg Sun) to :

- a gravitational coefficient GM
- a body centered ICRF frame (which can be retrieved with method `getICRF()`).
- a body centered EME2000 frame (which can be retrieved with method `getEME2000()`).
- a body centered inertial frame taking into account only the constant part (α, δ) of IAUPole (which can be retrieved with method `getInertialFrame(IAUPoleModelType.CONSTANT)`).
- a body centered inertial frame taking into account only the constant and secular parts (α, δ) of

IAUPole (which can be retrieved with method `getInertialFrame(IAUPoleModelType.MEAN)`).
- a body centered inertial frame taking into account the constant secular and harmonics parts (α, δ) of IAUPole (which can be retrieved with method `getInertialFrame(IAUPoleModelType.TRUE)`).
- a body centered rotating frame taking into account only the constant part w of IAUPole (which can be retrieved with method `getRotatingFrame(IAUPoleModelType.CONSTANT)`). It's parent frame is the inertial equator frame.
- a body centered rotating frame taking into account only the constant and secular parts w of IAUPole (which can be retrieved with method `getRotatingFrame(IAUPoleModelType.MEAN)`). It's parent frame is the mean equator frame.
- a body centered rotating frame taking into account the constant secular and harmonics parts w of IAUPole (which can be retrieved with method `getRotatingFrame(IAUPoleModelType.TRUE)`). It's parent frame is the true equator frame.

Inertially-oriented and body-oriented frames are defined in the following way:

- Body-centered inertial frame is centered on the celestial body centered and pole axis (Z axis) is shifted by right ascension α and declination δ.
- Body-centered rotating frame is linked to inertially-oriented frame by a rotation of angle W(t) (reference values for W(t) is provided by IAU).

To build a celestial body, the user can:

- use the static methods of the `CelestialBodyFactory` to create instances of the most common celestial bodies (Moon, Sun, Jupiter, etc.). JPL Ephemeris data are used. Warning: using the factory requires to load JPL Ephemeris data beforehand.
- use the simplified models (Meeus, etc.).
- creates its own celestial body using the class `UserCelestialBody` as well as its own pole motion using the class `UserIAUPole`.

For the two first case (JPL and Meeus), the pole data of the body are automatically retrieved using the IAU data through `IAUPoleFactory` class (data is contained within PATRIUS). By default, IAU pole data for planetary bodies (including Sun and moon) are available in PATRIUS through the use of the `IAUPoleFactory.getIAUPole()`. Data come from the IAU 2009 working group Technical Note.

These methods are detailed in the following sections.

Ephemeris Loader

For any celestial body of the Solar System, the actual computation of its position and velocity relies on the JPL planetary ephemerides files. These files are binary files and loaded thanks to the JPLCelestialBodyLoader object.

Note that celestial bodies and associated ephemeris loaders are distinct: a class loads the ephemeris returning a `CelestialBodyEphemeris` while the other loads the whole `CelestialBody`. However, for sake of clarity, the main visible loaders are celestial body loaders.

For the moment, this object is able to load JPL DE XXX files and IMCCE INPOP files which are the most commonly used data files (see below for an exact list of accepted files). For instance with the most common files, the JPL DE 405 covers the years 1600 to 2200 at maximum precision, the JPL DE 406 covers the years-3000 to +3000 at only slightly reduced precision. The DE 405 file is the basis

for the Astronomical Almanac and leads to sufficiently accurate results and, for most purposes, even the accuracy of DE 406 is sufficient.

JPLCelestialBodyLoader

In order to generate the ephemerides of one celestial body of the Solar System, one has to use the JPLCelestialBodyLoader as follows :

```
final JPLCelestialBodyLoader loaderEMB = new JPLCelestialBodyLoader(fileName,
                    EphemerisType.EARTH_MOON);
final JPLCelestialBodyLoader loaderSSB = new JPLCelestialBodyLoader(fileName,
                    EphemerisType.SOLAR_SYSTEM_BARYCENTER);
CelestialBodyFactory.addCelestialBodyLoader(CelestialBodyFactory.EARTH_MOON,
loaderEMB);
CelestialBodyFactory.addCelestialBodyLoader(CelestialBodyFactory.
SOLAR_SYSTEM_BARYCENTER, loaderSSB);

// Reference frame
Frame icrf = FramesFactory.getICRF();

// Ephemeris of the Sun
JPLCelestialBodyLoaderloader = new JPLCelestialBodyLoader("unxp2000.405",
                EphemerisType.SUN);

// Creation of the Sun
CelestialBody sun = loader.loadCelestialBody(CelestialBodyFactory.SUN);

// Coordinates of the Sun given a date and a reference frame
PVCoordinates pvSun = sun.getPVCoordinates(date, icrf);
```

When the user wants to create a JPLCelestialBodyLoader, first of all, he must supply the folder where the DE XXX files are stored. Then he has to give the name of the file which contains the data (DE XXX), the type of celestial body and a date (desired central date) as entries of the JPLCelestialBodyLoader.

The first argument (name of the data file) may be `null`. In this case, Patrius takes the first compatible file found. If one wants only files from the DE 405 ephemerides, for instance, he has to give the String `"^unx[mp](\\d\\d\\d\\d)\\.(?:405)$"`. The last argument can also be `null`. If the central date is not mentionned an arbitrary 100 days range will be loaded whereas if it is mentionned all data within a +/-50 days range around this date will be loaded.

Once the data is loaded, thanks to the JPLCelestialBodyLoader, the user can create the celestial body with the method loadCelestialBody() of the JPLCelestialBodyLoaderclass. To do so, the user has to be sure that all required data is loaded. Indeed, apart from the Moon, the Earth and the Earth-Moon barycenter, the creation of a celestial body requires some data on the Earth Moon barycenter and the Solar System barycenter in order to instanciate the frame in which the ephemerides of the celestial body will be defined. Therefore, the user has to create a JPLCelestialBodyLoaderfor the Earth-Moon barycenter and the Solar System barycenter prior to creating the celestial body, otherwise Patrius will arbitrarily load a DE file to generate the corresponding ephemerides that are used afterwards for the generation of the frame. Then the user has to complete the list of the loaders

of the CelestialBodyFactory with these two loaders before calling the method loadCelestialBody().

A static method `hasNoLoader()` in the class `CelestialBodyFactory` allows for a given celestial body to verify if a default loader exists. The user can verify if a specific loader exists for the body before using the default one this way.

NB : The user has to clean the CelestialBodyFactory memory if he does not want to work with the previously defined celestial bodies.

JPL ephemerides

The JPL ephemerides data is available on the JPL FTP server [R3].

Available ephemerides :

Development Ephemerides	Created in...	Description
DE200	September 1981	includes nutations but not librations. Referred to the dynamical equator and equinox of 2000. Covers JED 2305424.13 (1599 DEC 09) to JED 2513360.5 (2169 MAR 31). This ephemeris was used for the Astronomical Almanac from 1984 to 2003. (See Standish, 1982 and Standish, 1990).
DE202	October 1987	includes nutations and librations. Referred to the dynamical equator and equinox of 2000. Covers JED 2414992.5 (1899 DEC 04) to JED 2469808.5 (2050 JAN 02).
DE403	May 1993	includes nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 2305200.5 (1599 APR 29) to JED 2524400.5 (2199 JUN 22). Fit to planetary and lunar laser ranging data.(See Folkner et al. 1994).
DE405, DE406	May 1997	For the DE405: includes both nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 2305424.130 (1599 DEC 09) to JED 2525008.50 (2201 FEB 20) For the DE406 : the same as the DE405 except the time span : Spans JED 0624976.50 (-3001 FEB 04) to 2816912.50 (+3000 MAY 06) This is the same integration as DE405, with the accuracy of the interpolating polynomials has been lessened to reduce file size for the longer time span covered by the file.
DE410	April 2003	includes nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 2415056.5 (1900 FEB 06) to JED 2458832.5 (2019 DEC 15). Ephemeris used for Mars Exploration Rover navigation.

DE413	November 2004	includes nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 2414992.5, (1899 DEC 04) to JED 2469872.5 (2050 MAR 07). Created to update the orbit of Pluto to aid in planning for an occultation of a relatively bright star by Charon on 11 July 2005.
DE414	May 2005	includes nutations and librations. Covers JED 2414992.5, (1899 DEC 04) to JED 2469872.5 (2050 MAR 07). Fit to ranging data from MGS and Odyssey through 2003. (See Konopliv et al., 2006.)
DE418	August 2007	includes nutations and librations. Covers JED 2414864.13 (1899 JUL 29) to JED 2470192.5 (2051 JAN 21)
DE421	February 2008	includes nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 2414864.13 (1899 JUL 29) to JED 2471184.13 (2053 OCT 09) Fit to planetary and lunar laser ranging data. (See Folkner et al., 2009)
DE422	September 2009	includes nutations and librations. Referred to the International Celestial Reference Frame. Covers JED 625648.5, (-3000 DEC 07) to JED 2816816.5, (3000 JAN 30). Intended for the MESSENGER mission to Mercury. Extended integration time to serve as successor to DE406. Fit to ranging data from MGS and Odyssey through 2003. (See Konopliv et al., 2010.)
DE423	February 2010	includes nutations and librations. Referred to the International Celestial Reference Frame version 2.0. Covers JED 2378480.5, (1799 DEC 16) to JED 2524624.13, (2200 FEB 02). Intended for the MESSENGER mission to Mercury.

IMCCE INPOP ephemerides

The IMCCE INPOP ephemeris is available on IMCCE website.

Available INPOP ephemerides :

- 06b
- 06c
- 08a
- 10a
- 10b
- 10e
- 13c
- 17a
- 19a

Simplified analytical models

Meeus Model

The Meeus Model is a simplified model which gives the position of the Sun and the Moon with respect to the time T expressed in centuries (TT time scale). This model is an analytical model less precise than the DE ephemerides given by JPL. It is adapted for onboard applications. The class implementing the Meeus Model allows three different models computing the position of the Sun with appropriate equations : the standard model (provided by Jean Meeus), the Stela model and an onboard model. The main differences between these model is the computation of the obliquity of the ecliptic : indeed, its value is fixed to 0 for the standard model, given by an expression involving [math]T, T^{2}[/math]for Stela model and [math]T, T^{2}, T^{3}[/math] for the onboard model. Moreover, the onboard model computed the position in J2000 frame, whereas standard and Stela models use respectively EOD and MOD frame.

Regarding the precision, one has to expect a maximum difference of 25593km in position for the Sun and a maximum angular difference of 34.13 *(wrt DE405 ephemerides). For the Moon, one has to expect a maximum difference of 26 km in position and a maximum angular difference of 15.2* (wrt DE405 ephemerides). As for the performances (in terms of execution time), the Meeus model for the Sun is faster than the DE405 ephemerides. However, we did not come to the same conclusion for the Moon even by decreasing the degree of precision (number of terms taken into account to compute the latitude, longitude and distance which are needed to compute in fine the position of the Moon).

	Deviation in position wrt DE423	**Angular deviation wrt DE423**
Sun	12000 km	34.13
Moon 62x66x46	12.4 km	15.2
Moon 26x13x13	225 km	122
Moon 9x4x3	1591 km	889
	Deviation in position wrt DE405	**Angular deviation wrt DE405**
Sun	25593km	34.13
Moon 62x66x46	26 km	15.2

	Number of elementary operations	**Number of trigonometric operations**
Sun	89	10
Moon 62x66x46	2671	182
Moon 26x13x13	917	60
Moon 9x4x3	401	24

References for the tables :

- "Modèles d'éphémérides luni-solaires", CNES DCT/SB/MS, 03/14/2011
- "Modèle MEEUS pour éphémérides Lune-Soleil : Compléments sur le nombre d'opérations", CNES DCT/SB/MS

	execution time - DE405	**execution time- MEEUS**
Sun	59 s 548 ms	19 s 938 ms
Moon 62x66x46	11 s 625 ms	3 mn 22 s 323 ms
Moon 26x13x13	11 s 625 ms	1 mn 17 s 74 ms

Moon 9x4x3	11 s 625 ms	40 s 326 ms

In order to build such a Sun or Moon, one has to use the object MeeusSun or MeeusMoon which both extend AbstractCelestialBody. Note that it is not possible to build those celestial bodies from the CelestialBodyFactory, for the moment.

Basic board Sun model

The basic board Sun model is a simplified analytical model which gives the direction of the Sun (the normalized position) with respect to time. This model is similar to the Meeus model (the constants of the model are different) and is adapted for onboard applications. The reference inertial frame is the CIRF.

User-defined celestial bodies

User can defined its own celestial body by using `UserCelestialBody`. This class requires to provide:

- Its name.
- Its position-velocity through time using a `PVCoordinateProvider` implementation.
- Its gravitational constant.
- Its pole motion using `IAUPole` implementation or directly the `UserIAUPole` implementation.

The `UserIAUPole` is a simple way to build standard pole motion from IAU note. It requires to provide for each component (α, δ and W) a list of functions of duration in days and duration in centuries (from a reference epoch) as defined in IAU note. This functions are `UnivariateDifferentiableFunction`. Available functions in PATRIUS include:

- `PolynomialFunction`: polynomial function
- `SineFunction`: function of the form k.sin(f) with f an `UnivariateDifferentiableFunction`
- `CosineFunction`: function of the form k.cos(f) with f an `UnivariateDifferentiableFunction`

Example: building Ceres. According to the IAU report, Ceres has the following parameters:

- α = 291◦ ± 5◦
- δ = 59◦ ± 5◦
- W = 170.90◦ + 952.1532◦d

With *d* being the interval in days from the standard epoch (the standard epoch is JD 2451545.0, i.e. 2000 January 1 12 hours TDB)

Ceres gravitational constant is 6.263E10 m^^3^^kg^^-1^^s^^-2^^.

If one knows Ceres motion given for instance by a `PVCoordinatePropagator` *pv*, then one can build Ceres with the following code:

```
// Gravitational parameter
final double gm = 6.263E10;

// Pole motion - Method 1 - Implementation if IAUPole interface
final IAUPole pole = new IAUPole() {
```

```
    @Override
    public double getPrimeMeridianAngle(final AbsoluteDate date) {
        // W
        final double d = date.durationFrom(AbsoluteDate.J2000_EPOCH) /
Constants.JULIAN_DAY;
        return FastMath.toRadians(170.90) + FastMath.toRadians(952.1532)* d;
    }

    @Override
    public Vector3D getPole(final AbsoluteDate date) {
        // Pole bias: alpha and delta
        return new Vector3D(FastMath.toRadians(291), FastMath.toRadians(59));
    }
    ...
};

// Pole motion - Method 2 - Use of UserIAUPole class
// Alpha 0 coefficients
final List<IAUPoleFunction> alpha0List = new ArrayList<IAUPoleFunction>();
alpha0List.add(new IAUPoleFunction(IAUPoleType.CONSTANT, new
PolynomialFunction(new double[] { 291 }), IAUTimeDependency.DAYS);
final IAUPoleCoefficients1D alpha0Coeffs = new
IAUPoleCoefficients1D(alpha0List);
// Delta 0 coefficients
final List<IAUPoleFunction> delta0List = new ArrayList<IAUPoleFunction>();
delta0List.add(new IAUPoleFunction(IAUPoleType.CONSTANT, new
PolynomialFunction(new double[] { 59 }), IAUTimeDependency.DAYS);
final IAUPoleCoefficients1D delta0Coeffs = new
IAUPoleCoefficients1D(delta0List );
// W coefficients
final List<IAUPoleFunction> w0List = new ArrayList<IAUPoleFunction>();
w0List.add(new IAUPoleFunction(IAUPoleType.CONSTANT, new
PolynomialFunction(new double[] { 170.9 }), IAUTimeDependency.DAYS);
w0List.add(new IAUPoleFunction(IAUPoleType.SECULAR, new
PolynomialFunction(new double[] { 0, 952.1532 }), IAUTimeDependency.DAYS);
final IAUPoleCoefficients1D wCoeffs = new IAUPoleCoefficients1D(w0List);
// Build pole
final IAUPole pole = new UserIAUPole(new IAUPoleCoefficients(alpha0Coeffs,
delta0Coeffs, wCoeffs));

// Build Ceres body
final CelestialBody ceres = new UserCelestialBody("Ceres", pv, gm, pole,
FramesFactory.getICRF(), null);
```

Then `ceres` object possesses all features of:

- A `CelestialBody`: retrieve body-centered inertial frames or body-centered rotating frames
- A `PVCoordinateProvider`: retrieve position and velocity at any time

Body shapes

The one-axis ellipsoid is a good approximate model for most planet-size and larger natural bodies. It is the equilibrium shape reached by a fluid body under its own gravity field when it rotates. The symmetry axis is the rotation or polar axis.

OneAxisEllipsoid

This type is used to represent the shape of a planet. One very useful implementation represents an ellipsoid (OneAxisEllipsoid). It is constructed from an equatorial radius, a flattening coefficient, and a reference frame that will be used to localize Geodetic points on the shape.

It could be interesting to obtain the intersection point of a line from the satellite to the surface of the body for a given altitude. To that purpose, one can use the method getIntersectionPoint(Line, Vector3D, Frame, AbsoluteDate, double) of the interface BodyShape.

```
AbsoluteDate date = new AbsoluteDate(new DateComponents(2008, 03, 21),
                                                TimeComponents.H12,
                                                TimeScalesFactory.getUTC());
CelestialBodyFrame frame = FramesFactory.getITRF();
// Body shape model
OneAxisEllipsoid earth = new OneAxisEllipsoid(6378136.460, 1 / 298.257222101,
frame);

// Satellite on any position

circ = new CircularOrbit(7178000.0, 0.5e-4, 0., FastMath.toRadians(50.),
FastMath.toRadians(0.),
                                    FastMath.toRadians(90.),
PositionAngle.MEAN,
                                    FramesFactory.getEME2000(), date, mu);

// Transform satellite position to position/velocity parameters in EME2000
and ITRF200B
PVCoordinates pvSatEME2000 = circ.getPVCoordinates();
PVCoordinates pvSatItrf  = frame.getTransformTo(FramesFactory.getEME2000(),
date).transformPVCoordinates(pvSatEME2000);
Vector3D pSatItrf  = pvSatItrf.getPosition();

// Nadir point of the satellite
Vector3D pointItrf     = new Vector3D.ZERO;
Vector3D direction = Vector3D(1., pSatItrf,-1., pointItrf);
Line line = new Line(pSatItrf, direction);
// intersection point between the ellipsoid and the line that joins the
satellite and the center of the body
double altitude  = 0;
EllipsoidPoint nadir = earth.getIntersectionPoint(line, pSatItrf, frame,
date, altitude);
```

Note: The ThreeAxisEllipsoid class shares many features with the OneAxisEllipsoid and allows to define an ellipsoid fully defined by its three axis radius (along the directions (0, 0, 1) ; (0, 1, 0) ; (0, 0, 1)).

BodyShape and OneAxisEllipsoid

OneAxisEllipsoid extends the AbstractEllipsoidBodyShape class (also extended by ThreeAxisEllipsoid) and AbstractEllipsoidBodyShape implements the EllipsoidBodyShape interface, which extends the StarConvexBodyShape interface, which, at its turn, extends the BodyShape interface. See the [MIS_SENSORS_PatriusBodySpheroid specific page] for more details.

Facet celestial body

`FacetBodyShape` is a class used to define a celestial body as a mesh. This is particularly useful for small irregular bodies such as asteroids. For example, Phobos contains here 100 000 vertices and 200 000 facets:

This class:

- Inherits the `BodyShape` interface and hence can be used together with `EclipseDetector` and `SensorModel`.
- Provides some useful and optimised methods for small body handling. Facets are connected to each other allowing some methods to be writen in a recursive or iterative way and hence being very fast. If *n* is the number of vertices of the body, fast recursive or iterative methods are in O(log(n)).

For use as a `CelestialBody`, this class must be linked to a `UserCelestialBody`.

Mesh provider

A mesh is described by a list of facets `Triangle` and or vertices `Vertex`. A `Triangle` contains three vertices. Mesh is provided by the generic interface `MeshProvider`. This interface currently possesses two implementation:

- `ObjMeshLoader`: in this case, the mesh is provided under official .obj file format. See This page for more information on the format.
- `GeodeticMeshLoader`: in this case, the mesh is provided in an ASCII column file listing each vertex in the format [Latitude Longitude Altitude].

Available methods

Note that most methods return **exact** results. For example, `distanceTo` methods returns exact distance to mesh body. Only `getEllipsoid(EllipsoidType)` and `getApparentRadius` methods return a simplified result.

`FacetBodyShape` provides the following methods:

- `getIntersection(Line, Vector3D, Frame, AbsoluteDate)` and similar which return an `Intersection` object.This object contains the intersection point as well as the `Triangle` containing the intersection point. This method is recursive and is in O(log(n)).
- `buildPoint(LLHCoordinatesSystem, double, double, double, String)` and similar

which return the facet point associated to the specified corrinates in the given LLH coordinates system.
- `getApparentRadius()` methods which provides the local radius from an object and an occulted body using an approximate iterative algorithm. This method is to be used only by `EclipseDetector`.
- `resize(MarginType, double)` method which returns a resized body shape.
- `closestPointTo` methods which return the two closest points between the line given in input and the current facet body shape. One of the two points belongs to the line, the other to the facet body shape.
- `distanceTo()` method which returns the distance to the body. This method is recursive and is hence in O(log(n)).
- `getNeighbors()` methods which returns the neighbors triangles to:
  - A point or a triangle given a "neighborhood distance"
  - A triangle given a "neighborhood order". In this case, order 1 returns the immediate neighbors of the triangles, order 2 returns also the neighbors' neighbors and so on.

This method is recursive and is hence in O(log(n)).

- `getFieldData()` which returns a container `FieldData` containing data related to the field of view:
  - Visible triangles from the satellite field of view `IFieldOfView`. A visible triangle must be entirely in the field of view and not masked by any other triangle.
  - Contour of the seen triangles. Contour is strictly contained in the field of view

If main direction of the field of view is provided, this method is recursive and is in O(log(n)). Otherwise, it is in O(n) and is much slower.

- `isInEclipse()` method which returns true if the satellite is currently in eclipse, false otherwise. Penumbra is not taken into account. This method is in O(log(n)).
- `getNeverVisibleTriangles()` which returns a list of facets which are never visible from the provided ephemeris. Visibility criteria is the same as for `getFieldData()` method. This method is in O((n)).
- `getNeverEnlightenedTriangles()` which returns a list of facets which are never enlightened by the Sun. Visibility criteria is the same as for `getFieldData()` method. This method is in O((n)).
- `getVisibleAndEnlightenedTriangles()` which returns a list of facets which are visible and enlightened at the same time, at least once on the provided ephemeris. Visibility criteria is the same as for `getFieldData()` method. This method is in O((n)).
- `getEllipsoidType()` which returns the type of the ellipsoid.

The class `BodyShapeFitter` allows to build shapes fitted on the main Shape provided as input. Different criteria are available and the type of ellipsoid returned can be obtained by calling the method `getEllipsoid(EllipsoidType)`.

EllipsoidPoint

The ellipsoid point is defined by a latitude, a longitude and an altitude in the frame associated to the ellipsoid (EllipsoidBodyShape). It could be interesting to know the position of a satellite in terms of geodetic coordinates rather than Cartesian ones and vice versa (the corresponding methods in OneAxisEllipsoid).

```
// equatorial radius of the celestial body
```

```
double ae = 6378137.0;
// flatness of the celestial body
double f = 1.0 / 298.257222101;
// date
AbsoluteDate date = AbsoluteDate.J2000_EPOCH;
// reference frame attached to the body
CelestialBodyFrame frame = FramesFactory.getITRF();
// body shape model (ellipsoid)
OneAxisEllipsoid model = new OneAxisEllipsoid(ae, f, frame);

// transformation with jacobian matrix : cartesian to geodetic

// initial cartesian point that will be transformed
Vector3D cp = new Vector3D(4637885.347, 121344.1308, 4362452.869);
// coresponding ellipsoid point
EllipsoidPoint point = model.buildPoint(cp, frame, date, "point");

// transformation with jacobian matrix : geodetic to cartesian

// coresponding Cartesian point
Vector3D cp2 =
model.computePositionFromEllipsodeticCoordinates(0.852479154923577,
0.0423149994747243, 111.6);
```

It could be also interesting to obtain the jacobian matrix of the transformation.

```
// transformation : cartesian to geodetic

final double[][] computedJacobian =
LLHCoordinatesSystem.ELLIPSODETIC.jacobianFromCartesian(point);

// transformation : geodetic to cartesian

final double[][] computedJacobian2 =
LLHCoordinatesSystem.ELLIPSODETIC.jacobianToCartesian(point);
```

Getting Started

TBD

Contents

Interfaces

Interface	Summary	Javadoc
BodyShape	Interface representing the rigid surface shape of a natural body.	...
CelestialBody	Interface for celestial bodies like Sun, Moon or solar system planets.	...

CelestialBodyEphemeris	Interface for celestial body ephemeris like Sun, Moon or solar system planets.	...
CelestialBodyLoader	Interface for celestial body loaders.	...
CelestialBodyEphemerisLoader	Interface for celestial body ephemeris loaders.	...
MeshLoader	Interface for FacetCelestialBody mesh provider.	...

Classes

Class	Summary	Javadoc
CelestialBodyFactory	Factory class for bodies of the solar system.	...
BodyPoint	Point location relative to a 2D body surface.	...
JPLCelestialBodyLoader	Loader for JPL ephemerides binary files (DE 405, DE 406, ...).	...
OneAxisEllipsoid	Modeling of a one-axis ellipsoid.	...
MeeusSun	Position of the Sun according to Meeus model. Three models with there appropriate equations are avaible : the standard model (former MeeusSun), Stela model (former MeeuSunStela) and an on-board model	...
MeeusMoon	Position of the Moon according to Meeus model.	...
BasicBoardSun	Direction of the Sun according to a basic board Sun model.	...
UserCelestialBody	User-defined celestial body	...
UserIAUPole	User-defined IAU pole motion	...
IAUPoleFactory	Factory for retrieval of solar system bodies IAU pole data	...
IAUPoleFunction	Atomic IAU pole function. IAU pole data is the sum of atomic IAUPoleFunction.	...
FacetBodyShape	Celestial body defined by a mesh.	...
BodyShapeFitter	Fitter for a given body shape provided as input.	...
Triangle	Unitary facet (triangle) for a FacetCelestialBody.	...
ObjMeshLoader	.obj 3D file mesh loader.	...
GeodeticMeshLoader	Mesh loader for ASCII files describing the body with latitude/longitude/altitude components (1 per line).	...

Espaces de noms

- Page
- Discussion

Variantes

Affichages

- Lire
- Voir le texte source
- Historique
- Exporter en PDF

Plus

Rechercher

Rechercher Lire

PATRIUS

- Welcome

Evolutions

- Main differences between V4.15 and V4.14
- Main differences between V4.14 and V4.13
- Main differences between V4.13 and V4.12
- Main differences between V4.12 and V4.11
- Main differences between V4.11 and V4.10
- Main differences between V4.10 and V4.9
- Main differences between V4.9 and V4.8
- Main differences between V4.8 and V4.7
- Main differences between V4.7 and V4.6.1
- Main differences between V4.6.1 and V4.5.1
- Main differences between V4.5.1 and V4.4
- Main differences between V4.4 and V4.3
- Main differences between V4.3 and V4.2
- Main differences between V4.2 and V4.1.1
- Main differences between V4.1.1 and V4.1
- Main differences between V4.1 and V4.0
- Main differences between V4.0 and V3.4.1

User Manual

- User Manual 4.15
- User Manual 4.14
- User Manual 4.13
- User Manual 4.12
- User Manual 4.11
- User Manual 4.10
- User Manual 4.9
- User Manual 4.8
- User Manual 4.7
- User Manual 4.6.1
- User Manual 4.5.1
- User Manual 4.4
- User Manual 4.3
- User Manual 4.2
- User Manual 4.1
- User Manual 4.0
- User Manual 3.4.1
- User Manual 3.3

Tutorials

- Tutorials 4.15
- Tutorials 4.14
- Tutorials 4.13.5
- Tutorials 4.12.1
- Tutorials 4.8.1
- Tutorials 4.5.1
- Tutorials 4.4
- Tutorials 4.1
- Tutorials 4.0

Links

- CNES freeware server

Navigation

- Accueil
- Modifications récentes
- Page au hasard
- Aide

Outils

- Pages liées

- Suivi des pages liées
- Pages spéciales
- Adresse de cette version
- Information sur la page
- Citer cette page

- Dernière modification de cette page le 5 septembre 2024 à 14:01.

- Politique de confidentialité
- À propos de Wiki
- Avertissements

-