

Data initialization

De Wiki

Aller à : [navigation](#), [rechercher](#)
[Data initialization](#)

The main modifications between V11.3 and previous versions are due to the use of original [PATRIUS](#) classes rather than to use "Custom" classes originally proposed by [GENOPUS](#). Anyway, some "Custom" classes still remain as [PATRIUS](#) classes are not sufficiently consistent.

Sommaire

- [1 Orbit](#)
- [2 Earth features](#)
- [3 Vehicle](#)
- [4 Forces](#)
 - [4.1 Potential](#)
 - [4.2 Aerodynamic](#)
 - [4.3 Solar radiative pressure](#)
 - [4.4 Specific forces](#)
- [5 Maneuvers](#)
 - [5.1 Propulsive properties](#)
 - [5.2 Sequence](#)
 - [5.3 Impulsive maneuver](#)
 - [5.4 Constant thrust maneuver](#)
- [6 Attitude](#)
- [7 Integration](#)
- [8 PSIMU object](#)

Orbit

To initialize an orbit, we simply have to build it using [[PATRIUS](#)] object. For example:

```
final TimeScale TUC = TimeScalesFactory.getUTC();
final AbsoluteDate date = new AbsoluteDate("2010-01-01T12:00:00.000", TUC);
final double sma = Constants.WGS84_EARTH_EQUATORIAL_RADIUS + 250.e3;
final double ecc = 0.;
final double inc = FastMath.toRadians(51.6);
final double raan = FastMath.toRadians(0.);
final double pa = FastMath.toRadians(0.);
final double ano = FastMath.toRadians(0.);
final Frame GCRF = FramesFactory.getGCRF();
final KeplerianOrbit iniOrbit =
    new KeplerianOrbit(sma, ecc, inc, raan, pa, ano, PositionAngle.MEAN, GCRF,
        date, MU);
```

Earth features

Data equivalent to the “[Earth features](#)” tab are distributed via two arguments:

1. A [ExtendedOneAxisEllipsoid](#) object which will define the shape of the planet

```
final ExtendedOneAxisEllipsoid EARTH =  
    new ExtendedOneAxisEllipsoid(REQ, FLAT, ITRF, "EARTH");
```

2. A type of enumerates that will propose some pre-defined configurations as:

- **NOTHING** (all options set to null)
- **ONLY_PREC_NUT** (only precession and nutation)
- **FACTORY** (equivalent to the most complete definition as in the [GUI](#))
- **IGNORE** (nothing is define internally by **PSIMU** which will take into account the user parametrization previously done)

Vehicle

Since V11.3, it is no more necessary to pass an [Assembly](#) object but a [Vehicle one](#). Anyway, it is allways mandatory to pass a [MassProvider](#) object, issued from the [Vehicle](#) one. It could seem curious but it is due to the fact that, if we have maneuvers, it will be mandatory to initialize them with the same [MassProvider](#) (see example [here](#)).

In fact, we need to pass both an [Assembly](#) and a [MassProvider](#). It could seem curious but it is due to the fact that, if we have maneuvers, it will be mandatory to initialize them with the same [MassProvider](#) (see example [here](#)). To get them, we will use the [createAssembly\(\)](#) method given by [Vehicle](#) class.

```
final Assembly assembly = vehicle.createAssembly(GCRF);  
final MassProvider mm = new MassModel(assembly);
```

Note : for CPU time purpose it is recommanded that the frame (GCRF in the previous example), mandatory in the construction of the vehicle, must be exactly the same that the one used for the propagation.

In the example below, we will then use the [Vehicle](#) class initializing mass and aerodynamic properties (here no engines and no tanks):

```
// Dry mass  
final double dryMass = 1000.;  
final MassProperty dryMassProperty = new MassProperty(dryMass);  
  
// Shape  
final double sref = 5.0;  
Sphere sphere = new Sphere(sref);  
final RightParallelepiped solarPanels = new RightParallelepiped(10., 0., 0.);
```

```
VehicleSurfaceModel vehicleRefSurface = new VehicleSurfaceModel(sphere,
solarPanels);

// Aerodynamic properties
final double cd = 2.0;
final double cl = 0.;
final AerodynamicProperties aerodynamicProperties = new
AerodynamicProperties(vehicleRefSurface, cd, cl);

final Vehicle vehicle = new Vehicle(vehicleRefSurface, null, dryMassProperty,
aerodynamicProperties, null, null, null);

final Assembly assembly = vehicle.createAssembly(GCRF);
final MassProvider mm = new MassModel(assembly);
```

Note : due to a PATRIUS anomaly, if the user needs to define solar panels characteristics, it is mandatory to use the Vehicle constructor with all parameters and the VehicleSurfaceModel including the panels as above (and not a list of Facet); also, do not use the setters methods.

Forces

Since V11.3, to define, which forces will be applied along the trajectory, we use the [PATRIUS ForceModelsData](#) class (no more necessary to use the specific [CustomForceModels](#) class which does not exist anymore).

Potential

In the following example, we will initialize the potential model (**mandatory**).

```
// Potential
final ForceModel potential =
EarthGravitationalModelFactory.getDroziner(GravityFieldNames.GRGS,
"grim4s4_gr", 8, 8, true);

final ForceModelsData forces = new ForceModelsData(potential, null, null,
null, null, null, null, null, null, null, null, null);
```

Aerodynamic

Then we could add an aerodynamic force model. Since V11.3, it is no more necessary to use "Custom" classes.

```
final double REQ = Constants.WGS84_EARTH_EQUATORIAL_RADIUS;
final double FLAT = Constants.WGS84_EARTH_FLATTENING;
final Frame ITRF = FramesFactory.getITRF();
final ExtendedOneAxisEllipsoid EARTH =
    new ExtendedOneAxisEllipsoid(REQ, FLAT, ITRF, "EARTH");
```

```
final Atmosphere atmosphere = new US76(EARTH);
final DragForce dragForce = new DragForce(1., atmosphere, assembly);

forces.setDragForce(dragForce);
```

Solar radiative pressure

Here are explained how to use both radiative pressure and rediffused radiative pressure force models. Again, since V11.3, no more to use old "Custom" classes.

```
final CelestialBody sunBody = new MeeusSun();
final double dRef = 1.4959787E11;
final double pRef = 4.5605E-6;
final ExtendedOneAxisEllipsoid EARTH =
    new ExtendedOneAxisEllipsoid(REQ, FLAT, ITRF, "EARTH");

SolarRadiationPressureEllipsoid radPres = new
SolarRadiationPressureEllipsoid(dRef, pRef, sunBody, EARTH, assembly, 1.);

forces.setSolarRadiationPressure(radPres);

// By default values used when GUI mode ...
final int inCorona = 1;
final int inMeridian = 10;
final IEmissivityModel inEmissivityModel = new KnockeRiesModel();
// Value accessible via GUI mode
final boolean inAlbedo = false;
final boolean inIr = false;
final double coefAlbedo = 1.;
final double coefIr = 1.;

RediffusedRadiativeModel rdm = new RediffusedRadiativeModel(inAlbedo, inIr,
coefAlbedo, coefIr, assembly);
RediffusedRadiationPressure reDiff = new
RediffusedRadiationPressure(sunBody, GCRF, inCorona, inMeridian,
inEmissivityModel, rdm);

forces.setRediffusedRadiationPressure(reDiff);
```

Specific forces

It is also possible to add specific forces via the [addForceModel\(\)](#) method as below:

```
final Psimu test = new Psimu(...);
test.addForceModel(myForce);
```

Maneuvers

If the user wants to add some maneuvers, it will have to do it using the [\[GENOPUS\] Custom ManeuverSequence](#) class. Indeed, this class is derived from the PATRIUS [ManeuverSequence](#) but allows taking into account Engines and Tanks notion. From [\[PATRIUS\]](#) V3.4, it will be possible to directly use [\[PATRIUS\]](#) class.

Propulsive properties

First, the user will have to define propulsive properties with engine(s) and tank(s).

```
// Propulsive properties
final PropulsiveProperty engine = new PropulsiveProperty(400, 320.);
engine.setPartName("ENGINE");
final ArrayList<PropulsiveProperty> listOfEngines = new
ArrayList<PropulsiveProperty>();
listOfEngines.add(engine);
final TankProperty tank = new TankProperty(200.);
tank.setPartName("TANK");
final ArrayList<TankProperty> listOfTanks = new ArrayList<TankProperty>();
listOfTanks.add(tank);
listOfTanks.add(tank);

final Vehicle vehicle = new Vehicle(vehicleRefSurface, null, dryMassProperty,
null, null, listOfEngines, listOfTanks);
```

Sequence

Then, a maneuver sequence will be created thanks to the [CustomManeuverSequence](#) (the [\[PATRIUS\] ManeuverSequence](#) is not directly used as it does not still manage events).

Impulsive maneuver

Here we have an example of a 10m/s impulse maneuver in [TNW](#) occurring for an [AOL](#) of 180 degrees:

```
final double aol = FastMath.toRadians(180.);
final CustomAOLDetector aolDetector = new CustomAOLDetector(aol,
PositionAngle.TRUE, GCRF, MAXCHECK, THRESHOLD, Action.STOP);
final IntervalOccurrenceDetectorWrapper impEvent = new
IntervalOccurrenceDetectorWrapper(aolDetector, 1, 1, 1, Action.STOP);
final Vector3D deltaV = new Vector3D(10., 0., 0.);
final CustomImpulseManeuver imp = new CustomImpulseManeuver("Impulse
maneuver", listOfEngines.get(0), listOfTanks.get(0), LOFType.TNW, impEvent,
deltaV, mm);
```

Note: it is mandatory to configure the event with a STOP action (needed by [\[PATRIUS\]](#)).

At last, the user will create the maneuver sequence to pass to **PSIMU**:

```
final CustomManeuverSequence manSeq = new CustomManeuverSequence(0., 0.);
manSeq.addManeuver(imp);
```

Constant thrust maneuver

The next source code shows how to define a constant thrust maneuver ...

```
final double start = iniOrbit.getKeplerianPeriod();
final AbsoluteDate startDate = date.shiftedBy(start);
final CustomDateDetector startEvent = new CustomDateDetector(startDate,
MAXCHECK, THRESHOLD, Action.STOP);
final double dur = 1000.;
final AbsoluteDate endDate = startDate.shiftedBy(dur);
final CustomDateDetector stopEvent = new CustomDateDetector(endDate,
MAXCHECK, THRESHOLD, Action.STOP);
final Vector3D direction = new Vector3D(-1., 0., 0.);
CustomConstantManeuver cont =
    new CustomConstantManeuver("Continuous maneuver", listOfEngines.get(0),
listOfTanks.get(0), LOFTType.TNW, startEvent, stopEvent, direction, mm);
```

Attitude

It is mandatory to get at least one attitude law. With the V11.0, it is not possible to have only one law and it is mandatory to use the “switch” possibility by using the [\[PATRIUS\]](#) [AttitudesSequence](#) class.

This example shows how to define a single law (available since V11.1):

```
final AttitudeLaw attitudeLaw = new LofOffset(LOFTType.TNW);
```

This other example uses a switching possibility:

```
final AttitudeLaw law = new LofOffset(LOFTType.TNW, RotationOrder.ZYX, 0., 0.,
0.);
final double maxCheck = AbstractDetector.DEFAULT_MAXCHECK;
final double threshold = AbstractDetector.DEFAULT_THRESHOLD;
final EventDetector event =
    new DateDetector(date, maxCheck, threshold, Action.RESET_STATE);
final AttitudesSequence seqAtt = new AttitudesSequence();
seqAtt.addSwitchingCondition(law, event, true, false, law);
```

Note : be very careful not to use by default constructors for event detectors as they introduce a STOP event, so your propagation will stop anyway !

Integration

Then integration data have to be defined. Here an example with [RK4](#) integrator ...

```
final double duration = 86400.;
final double hStop = 120.e+3;
final PsimuPropagationData propagationData = new
PsimuPropagationData(duration, GCRF, hStop);
final double step = 5.;
final RK4data integrationData = new RK4data(propagationData, step);
```

... or with the **DOP** integrator ...

```
final double minStep = 0.1;
final double maxStep = 30.;
final double[] vecAbsoluteTolerance =
    { 1.0E-5, 1.0E-5, 1.0E-5, 1.0E-8, 1.0E-8, 1.0E-8 };
final double[] vecRelativeTolerance =
    { 1.0E-8, 1.0E-8, 1.0E-8, 1.0E-8, 1.0E-8, 1.0E-8 };
final double absMassTolerance = 1.0E-3;
final double relMassTolerance = 1.0E-2;
final double positionError = 0.;
final DopData intData = new DopData(propagationData, minStep, maxStep,
vecAbsoluteTolerance, vecRelativeTolerance, absMassTolerance,
relMassTolerance);
```

PSIMU object

As we have all mandatory data to initialize **PSIMU**, we can build such an object (note that in this example there is a “null” argument for the sequence of maneuvers.

```
final Psimu test = new Psimu(iniOrbit, EARTH, UserFramesConfsEnum.IGNORE,
integrationData, vehicle, mm, forces, manSeq, attitudeLaw);
```

Récupérée de « http://psimu.cnes.fr/index.php?title=Data_initialization&oldid=912 »

Menu de navigation

Outils personnels

- [3.135.198.49](#)
- [Discussion avec cette adresse IP](#)
- [Créer un compte](#)
- [Se connecter](#)

Espaces de noms

- [Page](#)
- [Discussion](#)

Variantes

Affichages

- [Lire](#)
- [Voir le texte source](#)
- [Historique](#)
- [Exporter en PDF](#)

Plus

Rechercher

PSIMU

- [Welcome](#)
- [Quick start](#)
- [News](#)

GUI Mode

- [Overall presentation](#)
- [Initial Orbit](#)
- [Earth features](#)
- [Vehicle](#)
- [Forces](#)
- [Maneuvers](#)
- [Attitude](#)
- [Integrator](#)
- [Events](#)
- [Output](#)
- [Console](#)

Batch mode

- [How to call it](#)

Java interface

- [Basic principle](#)
- [Data initialization](#)
- [Propagation](#)
- [Printing results](#)
- [Customize output variables](#)

Evolutions

- [Main differences between V11.7.3 and V11.7.4](#)
- [Main differences between V11.7.2 and V11.7.3](#)
- [Main differences between V11.7.1 and V11.7.2](#)
- [Main differences between V11.6.2 and V11.7.1](#)
- [Main differences between V11.5 and V11.6.2](#)
- [Main differences between V11.4.1 and V11.5](#)
- [Main differences between V11.4 and V11.4.1](#)
- [Main differences between V11.3 and V11.4](#)
- [Main differences between V11.2 and V11.3](#)
- [Main differences between V11.1 and V11.2](#)
- [Main differences between V11.0 and V11.1](#)

Training

- [Tutorials package for V11.7.x](#)
- [Tutorials package for V11.6](#)
- [Tutorials package for V11.5](#)
- [Tutorials package for V11.4](#)
- [Tutorials package for V11.3](#)
- [Tutorials package for V11.2](#)
- [Tutorials package for V11.0](#)

Links

- [CNES freeware server](#)

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 14 mai 2020 à 17:34.

- [Politique de confidentialité](#)
- [À propos de Wiki](#)

- [Avertissements](#)

