# User Manual 4.2 Attitude law


User Manual 4.2 Attitude law

## Introduction

### Scope

The purpose of this chapter is to describe the current Patrius attitude laws.

### Javadoc

The attitude objects are available in the package fr.cnes.sirius.patrius.attitudes.

| Library | Javadoc |
|---|---|
| Patrius | Package fr.cnes.sirius.patrius.attitudes |
| Patrius | Package fr.cnes.sirius.patrius.attitudes |

### Links

Orekit attitudes : Orekit Attitudes architecture description

### Useful Documents

None as of now.

### Package Overview

The general attitude law conception is described hereafter :

## Features Description

### Generalities

The Orekit class `AttitudeProvider` represents a generic provider for an attitude law. Attitude from an attitude provider can be retrieved using the methods `getAttitude(...)`

The new `AttitudeLaw` interface, implementing `AttitudeProvider` has been added to represent an attitude provider without a time interval of validity.

Instead, to meet spacecraft attitude field needs, a wrapper object has been created. The idea is to provide objects more suited for spacecraft attitude field rather than orbit determination field, that means to associate a specific time interval of validity to the attitude laws. The new objects `AttitudeLeg` and `AttitudeLawLeg` meet the requirements. `AttitudeLawLeg` wraps an `AttitudeLaw` object and defines a time interval of validity. This class implements the `AttitudeLeg`

interface which extends the `AttitudeProvider` interface and adds the methods to get the time interval.

In pratical terms, when it comes to create an attitude law for spacecraft attitude field purposes, the user has to create first an `AttitudeLaw` which is then given to an `AttitudeLawLeg` object, in addition to the initial and final date of the time interval of validity:

```
// Attitude law leg provider

// Elliptic earth shape
final OneAxisEllipsoid earthShape = new OneAxisEllipsoid(6378136.460, 1 /
298.257222101, frameITRF);

// Target pointing attitude provider over satellite nadir at date, without
yaw compensation
final NadirPointing nadirLaw = new NadirPointing(earthShape);

// First date
final AbsoluteDate date1 = new AbsoluteDate(new DateComponents(2012, 01, 01),
                           TimeComponents.H00, TimeScalesFactory.getUTC());
// Last date
final AbsoluteDate date2 = date1.shiftedBy(3600);

// Attitude law leg
final AttitudeLawLeg myAttitudeLaw = new AttitudeLawLeg(nadirLaw, date1,
date2);
```

Besides, compared to Orekit, the `Attitude` object has been slightly modified. In addition to the rotation and the rotation rate it can also provide the rotation acceleration, and its computation can be activated (or not) by the user. Therefore, an additional `getAttitude()` method and a `setSpinDerivativesComputation(final boolean setSpinDerivatives)` method have been created to the `AttitudeProvider` interface. This last method allows the user to activate the rotation acceleration computation. By default, only the rotation and the angular velocity is computed.

## Available attitude laws

### Ground pointing attitude laws

The satellite x axis is aligned to the satellite velocity vector, and the z axis points to the ground target:

- Body center ground pointing: the satellite z axis is pointing to the body frame center.
- Nadir pointing: the satellite z axis is pointing to the vertical of the ground point under satellite.
- Target ground pointing: the satellite z axis is pointing to a ground point target;
- LOF offset pointing: the attitude pointing law is defined by an attitude provider and the satellite axis vector chosen for pointing.

These laws require a body shape and a frame.

**Body center attitude law**

The satellite z axis points to a the body center; this law does not require a body shape.

**Target attitude law**

The satellite z axis points to a target; this law does not require a body shape.

**Celestial body pointed attitude law**

The celestial body pointed law is defined by two elements:

- a celestial body towards which some satellite axis is exactly aimed
- a phasing reference defining the rotation around the pointing axis

**Fixed rate attitude law**

The fixed rate attitude law handles a constant rate around a fixed axis. This corresponding attitude provider performs a simple linear extrapolation from an initial orientation, a rotation axis and a rotation rate.

**Constant attitude law**

The satellite frame has a constant orientation in a given reference frame. This orientation is defined by a rotation provided by the user.

**LOF offset attitude law**

It is an attitude law defined by fixed Roll, Pitch and Yaw angles (in any order) with respect to a local orbital frame.

**Spin stabilized attitude law**

Spin stabilized attitude provider. Spin stabilized laws are handled as wrappers for an underlying non-rotating law.

**Yaw compensation attitude law**

Yaw compensation is mainly used for Earth observation satellites: as a satellite moves along its track, the image of ground points moves on the focal point of the optical sensor. This motion is a combination of the satellite motion, the Earth rotation and the current attitude (in particular if the pointing includes Roll or Pitch offset).
In order to reduce geometrical distortion, the yaw angle is changed a little from the simple ground pointing attitude such that the apparent motion of ground points is along a prescribed axis (orthogonal to the optical sensor rows), taking into account all effects.

**Yaw steering attitude law**

Yaw steering is mainly used for low Earth orbiting satellites with no missions-related constraints on yaw angle. It sets the yaw angle in such a way the solar arrays have maximal lightning without changing the roll and pitch.

**Two directions attitude law**

The two directions provided by this attitude law are the following:

- the first direction is aligned with a given satellite axis;
- the second direction is aligned at best with another given satellite axis.

This attitude law can be used to represent the **GAP** (geocentric), and the **SUP** (heliocentric) pointing laws.

**RelativeTabulatedAttitudeLaw**

`RelativeTabulatedAttitudeLaw` is an implementation of `AttitudeLaw`. It is composed of a `AttitudeLegLaw:attitudeLegLaw` attribute defined by :
- An instance of `RelativeTabulatedAttitudeLeg:leg` built with an `List<Pair<Double, Rotation>>` (or an `List<Pair<Double, AngularCoordinates>>`) input.
- Two instances of `AttitudeLaw` to apply if the input date is inferior or superior to the interval of validity of `leg`. These laws can be of two types : `ConstantAttitudeLaw` or `ExtrapolatedAttitudeLaw` (local private class of `RelativeTabulatedAttitudeLaw`).

The `ConstantAttitudeLaw` is created with the input frame of `RelativeTabulatedAttitudeLaw`, and the value of the rotation to the bound of the interval (minimum or maximum).

The `ExtrapolatedAttitudeLaw` is a local private class. It is an implementation of `AttitudeLaw`. It is created with an `AngularCoordinates` and the expression frame of this attitude. Its method `getAttitude()` returns `AngularCoordinates.shiftedBy(double)`.

The `ExtrapolatedAttitudeLaw` is created with :
- the input frame of `RelativeTabulatedAttitudeLaw`
- the `AngularCoordinates` to the bound of the interval (minimum or maximum).
Here is an example of a creation of an instance of `RelativeTabulatedAttitudeLaw` :

```
// build law before as a ConstantAttitudeLaw
final RelativeTabulatedAttitudeLaw.AroundAttitudetype lawBefore =
     RelativeTabulatedAttitudeLaw.AroundAttitudetype.CONSTANT_ATT;

// build law after as a ExtrapolatedAttitudeLaw
final RelativeTabulatedAttitudeLaw.AroundAttitudetype lawAfter =
     RelativeTabulatedAttitudeLaw.AroundAttitudetype.EXTRAPOLATED_ATT;

// build RelativeTabulatedAttitudeLaw
final RelativeTabulatedAttitudeLaw relativeTabulatedAttitudeLaw =
      new RelativeTabulatedAttitudeLaw(refDate, listAr, frame, lawBefore,
lawAfter);
```

**AeroAttitudeLaw**

`AeroAttitudeLaw` is a law that aligns platform frame (Ox, Oy, Oz) with a frame defined by 3 angles: angle of attack, sideslip and velocity roll. With the following conventions:
- Platform frame is (Ox, Oy, Oz) with Ox along main frame, Oz in the symmetry plane of the aircraft

toward up direction.
- Velocity frame is (Oxv, Oyv, Ozv) with Oxv being along velocity and Ozv in the symmetry plane of the aircraft toward up direction.
The three angles are defined in the following way:
- Angle of attack is angle between (Oxv, Oyv) plane and Ox vector.
- Sideslip is angle between Oxv vector and (Ox, Oz) symmetry plane
- Velocity roll is angle around velocity vector.

**AttitudeLegLaw**

`AttitudeLegLaw` is an implementation of `AttitudeLaw`. It is composed of : an instance of `AttitudeLeg:leg`, an instance of `AttitudeLaw:lawbefore` and an other instance of `AttitudeLaw:lawAfter`. Its method `getAttitude(pvProv, date, frame)` returns :
- `lawbefore.getAttitude(pvProv, date, frame)` is the date is inferior to the interval of validity of leg
- `leg.getAttitude(pvProv, date, frame)` is the date is contained in the interval of validity of leg
- `lawAfter.getAttitude(pvProv, date, frame)` is the date is superior to the the interval of validity of leg

## Attitudes sequence

The attitudes sequence represents a sequence of several attitude laws (`AttitudeLaw` instances).

Unlike the `AttitudeLeg` objects, the `AttitudeLaw` instances do not have an interval of validity, therefore the switching from one law to another can not be based on time intervals. For a given date, only one attitude law in the sequence is in an "active" state, so that the attitude of the spacecraft is computed from that attitude law. The laws are activated in turn according to some switching events, which are added to the sequence before starting the propagation.

**Building an attitudes sequence**

Here is how an attitudes sequence is built:

- An empty attitudes sequence is created: `new AttitudesSequence();`
- The attitudes laws that will compose the sequence are instantiated;
- The switching detectors are instantiated and set up using the `addSwitchingCondition(AttitudeLaw, EventDetector, boolean, boolean, AttitudeLaw)` command; in addition to the event detector, the previous and the next attitude laws associated to the switching condition are specified; Warning: switching detectors should have `Action.RESET_STATE` as action.
- The `registerSwitchEvents(Propagator)` method is called once before propagation, after the switching conditions have been set, in order to wrap switch events to the propagator.

**Code sample**

```
final Frame gcrf = FramesFactory.getGCRF();

// Build the attitudes sequence:
final AttitudesSequence aseq = new AttitudesSequence();
```

```
final EventDetector switcherToTwo = new DateDetector(date.shiftedBy(100)) {
    @Override
    public Action eventOccurred(final SpacecraftState s, final boolean
increasing, final boolean forward) throws PatriusException {
            // Action.RESET_STATE is compulsory for the switch to be taken
into account
            return Action.RESET_STATE;
        }
    };

final EventDetector switcherToOne = new DateDetector(date.shiftedBy(200)) {
    @Override
    public Action eventOccurred(final SpacecraftState s, final boolean
increasing, final boolean forward) throws PatriusException {
            // Action.RESET_STATE is compulsory for the switch to be taken
into account
            return Action.RESET_STATE;
        }
    };

// Instantiate the attitude laws:
final AttitudeLaw lawOne = new LofOffset(gcrf, LOFType.QSW);
final AttitudeLaw lawTwo = new LofOffset(gcrf, LOFType.TNW);

// Add the switching detectors to the sequence:
aseq.addSwitchingCondition(lawTwo, switcherToOne, true, true, lawOne);
aseq.addSwitchingCondition(lawOne, switcherToTwo, true, true, lawTwo);

aseq.resetActiveProvider(lawOne);

// Properly register switching events to the propagator:
final Propagator propagator = new KeplerianPropagator(orbit, aseq);
aseq.registerSwitchEvents(propagator);

// Propagation:
propagator.propagate(date.shiftedBy(150));
```

**Limitations**

There is a limitation to be considered when using the AttitudesSequence class :

- it lacks robustness in the switching design. When a switching event occurs, the AttitudesSequence instance changes the attitude law, but does not retain the event date at which the law was set up, and also forgets which was the former law : if the AttitudesSequence computes a new Attitude after the switch happened programatically, it will always be with the new attitude law even if the requested date is before the switching event.

In practice, this happens when a step handler is added to a propagation to retrieve additional data samples. During a propagation step, the step handlers are processed after the event handlers, so the step handlers will always see the new law, even if the law changed only late during the step. This

leads to inaccuracies in the data retrieved by step handlers. These problems occur with analytical and numerical propagators as well.

### Attitude composition

The `ComposedAttitudeLaw` class allows to compose a main attitude law (implementing the `AttitudeLaw` interface) and one or several orientation laws (`IOrientationLaw` interface) whose purpose is to modify this law in order to obtain the "composed" attitude law.

It shall be created this way :

```
// attitude law creation
AttitudeLaw attitudeLaw = new MyAttitudeLaw(...);

// orientation laws creation (modifiers)
IOrientationLaw orientationLaw1 = new AnyOrientationLaw(...);
IOrientationLaw orientationLaw2 = new AnyOtherOrientationLaw(...);

// modifiers list creation
final LinkedList<IOrientationLaw> modifiers = new
LinkedList<IOrientationLaw>();

modifiers.add(orientationLaw1);
modifiers.add(orientationLaw2);

// composed attitude creation
final ComposedAttitudeLaw composedAtt = new ComposedAttitudeLaw(attitudeLaw,
modifiers);
```

The `ComposedAttitudeLaw` class implements the `AttitudeLawModifier` interface : it shall be used as any `AttitudeLaw`, and can also provide its "underlying attitude law" (the main attitude law used to build it).

Dedicated frames exist to describe those laws and are used to compute the composition :

- The `AttitudeFrame` is a frame defined by any attitude law (`AttitudeProvider`) and position velocity coordinates (`PVCoordinatesProvider`). At any date, its center is the position of the `PVCoordinatesProvider`, and its rotation from a reference frame is given by the attitude law.
- The `OrientationFrame` is defined from another `OrientationFrame` or from an `AttitudeFrame` by the corrections (`IOrientationLaw` objects) applied to them.

The `ComposedAttitudeLaw` class can be used to represent a biased geocentric pointing law (**biased GAP**): a geocentric attitude law is associated to a "modifier" that represents a constant rotation. This specific pointing law is used for the thrust: the bias added to the main pointing direction is the direction of the thrust.

## Getting Started

TBD

# Contents

## Interfaces

| Interface | Summary | Javadoc |
|---|---|---|
| **AttitudeProvider** | This interface is the main interface of the attitudes package, it is related to the orbit determination field. | ... |
| **AttitudeLaw** | This interface represents a generic attitude law provider, for which no interval of validity is specified. | ... |
| **AttitudeLawModifier** | This interface represents an attitude law provider that modifies/wraps another underlying provider. | ... |
| **IOrientationLaw** | Orientation law. | ... |

## Classes

| Class | Summary | Javadoc |
|---|---|---|
| **AbstractAttitudeLaw** | Abstract class representing the basic implementation of an attitude law provider. | ... |
| **Attitude** | Object representing the attitude of the spacecraft for a specific date and in a specific frame. | ... |
| **AttitudesSequence** | This class represents a sequence of attitude laws that are activated in turn according to switching events.. | ... |
| **TwoDirectionsAttitude** | This class implements a generic two directions attitude law. The first direction is aligned with a given satellite axis, the second direction is aligned at best with another given satellite axis. | ... |
| **AttitudeFrame** | Spacecraft frame (dynamic frame whose orientation is defined by the attitude law). | ... |
| **OrientationFrame** | Spacecraft orientation frame. | ... |
| **BodyCenterPointing** | This class implements a body center pointing attitude law: the satellite z axis is pointing to the body frame center. | ... |
| **BodyCenterGroundPointing** | This class implements a body center ground pointing attitude law: the satellite z axis is pointing to the ground pointing corresponding to the body frame center. | ... |
| **CelestialBodyPointed** | This class handles a celestial body pointed attitude provider. | ... |
| **FixedRate** | This class handles a simple attitude provider at constant rate around a fixed axis. | ... |
| **GroundPointing** | This class is a basic model for different kind of ground pointing attitude providers. | ... |
| **GroundPointingWrapper** | This class leverages common parts for compensation modes around ground pointing attitudes. | ... |
| **ConstantAttitudeLaw** | This class handles an constant attitude provider. | ... |
| **LofOffset** | Attitude law defined by fixed Roll, Pitch and Yaw angles (in any order) with respect to a local orbital frame. | ... |
| **LofOffsetPointing** | This attitude pointing law is defined by an attitude provider and the satellite axis vector chosen for pointing. | ... |

| | | |
|---|---|---|
| **NadirPointing** | This class handles nadir pointing attitude provider. | ... |
| **SpinStabilized** | This class handles a spin stabilized attitude provider. | ... |
| **TargetPointing** | This class handles target pointing attitude provider. | ... |
| **TargetGroundPointing** | This class handles target ground pointing attitude provider. | ... |
| **YawCompensation** | This class handles yaw compensation attitude provider. | ... |
| **YawSteering** | This class handles yaw steering attitude law. | ... |
| **ComposedAttitudeLaw** | Composed attitude law provider. | ... |
| **DirectionTrackingOrientation** | One direction orientation law. This law has to be used within a composed attitude law as a law modifier. | ... |
| **SunPointing** | This class implements a Sun pointing attitude law. The first direction is the satellite-sun direction, the second direction is either the sun poles axis or the normal to the satellite orbit plane. | ... |
| **IsisSunPointing** | This class implements a ISIS Sun pointing attitude law. The first direction is the satellite-sun direction (being the - Z_sun axis computed in GCRF frame), the second direction is the Y_sun axis computed in GCRF. | ... |
| **AttitudeLegLaw** | This class implements an attitude leg, one attitude law before the leg and one attitude law after the leg. | ... |
| **RelativeTabulatedAttitudeLaw** | This class implements a tabulated attitude leg with relative dates, with one attitude law before the leg and one attitude law after the leg. | ... |
| **AeroAttitudeLaw** | This class implements an aerodynamic attitude law defined by angle of attack, side slip and velocity roll. | ... |


Catégorie :

- User Manual 4.2 Attitude

# Menu de navigation

## Outils personnels

- 3.144.90.108
- 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

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 7 janvier 2019 à 13:39.

- Politique de confidentialité
- À propos de Wiki
- Avertissements
-