User Manual 3.4.1 Attitude leg

De Wiki
Aller à : navigation, rechercher
User Manual 3.4.1 Attitude leg

Sommaire

Introduction

Scope

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

An attitude leg is a time-bounded attitude law. Generalities on attitude laws can be found here.

Javadoc

Library	Javadoc
Orekit	Package org.orekit.attitudes
Orekit addons	Package org.orekit.attitudes

Links

Orekit attitudes : Orekit Attitudes architecture description

Useful Documents

Modèle:SpecialInclusion prefix=$theme sub section="UsefulDocs"/

Package Overview

The attitude leg `AttitudeLeg` interface inherits the `AttitudeProvider` interface. Its place in the global Attitude design can be found here.

Features Description

Generalities

Attitude legs inherit the interface `AttitudeLeg`. They are time-bounded attitude providers. In addition to `AttitudeProvider` services, they provide the method `getTimeInterval()` returning the leg timespan.

Available attitude leg

Attitude legs sequence

An attitude legs sequence is a container for several attitude legs, contiguous for their time intervals, in such a way that the attitude legs sequence can be processed like a single attitude leg by the propagator.

The attitude legs sequence is the equivalent of the Attitudes sequence, using attitude legs (`AttitudeLeg` instances) rather than attitude laws (`AttitudeLaw` instances). The switching from one attitude leg to another is based on the time interval of validity of the two legs.

An attitude legs sequence is associated to a `PVCoordinatesProvider` instance, which will **override any PVCoordinatesProvider** passed as parameter to the methods like getAttitude(). The reason for such a behaviour, which violates the contract of the `AttitudeProvider` interface, is that :

- an attitude legs sequence needs to **enforce coherence** between its inner attitude legs and its homing maneuvers.
- homing maneuvers are created and computed once by using a specific `PVCoordinatesProvider`, in order to preserve good performances.

Therefore, the attitude legs sequence can only compute attitudes with the `PVCoordinatesProvider` instance it was built with, and the inner attitude legs should be coherent with this provider (the attitude sequence does not check if it's the case!)

TabulatedAttitude

`TabulatedAttitude` is an implementation of `AttitudeLeg`. It represents a tabulated attitude leg.

In order to interpolate the attitude at a date, the user must specify a list of **ordered** attitudes, and can specify a number of points used by Hermite interpolation. If not specified, the number of points used by Hermite interpolation is set to a default number (currently 2).

```
final List<Attitude> attList = new ArrayList<Attitude>();
attList.add(attitude0);
attList.add(attitude1);
final int nbrInterpPoints = 2;
final TabulatedAttitude attLeg = new TabulatedAttitude(attList,
nbrInterpPoints);
```

It is possible to get the non-interpolated ordered attitudes :

```
final List<Attitude> attitudes = attLeg.getAttitudes();
```

Once the tabulated is defined, the computation can be performed on any orbital state using `getAttitude()` method:

```
Attitude attitude = attLeg.getAttitude(orbit, date,
FramesFactory.getEME2000());
```

RelativeTabulatedAttitudeLeg

`RelativeTabulatedAttitudeLeg` is an implementation of `AttitudeLeg`. An instance of `RelativeTabulatedAttitudeLeg` can be created with a `List<Pair<Double, Rotation>>` or with a `List<Pair<Double, AngularCoordinates>>`. Each `Rotation` (or `AngularCoordinates`) is associated with a `double` representing its time ellapsed in seconds since the reference date. Here is an example of a creation of an instance of `RelativeTabulatedAttitudeLeg` :

```
// date and frame
AbsoluteDate refDate = new AbsoluteDate(2008, 1, 1,
TimeScalesFactory.getTAI());
Frame frame = FramesFactory.getGCRF();
double timeEllapsedSinceRefDate = 1.0;

// List of AR
List<Pair<Double, AngularCoordinates>> listAr = new ArrayList<Pair<Double,
AngularCoordinates>>();
final AngularCoordinates ar = new AngularCoordinates(
                new Rotation(false, 0.48, 0.64, 0.36, 0.48), Vector3D.PLUS_I,
Vector3D.PLUS_J);
listAr.add(new Pair<Double, AngularCoordinates>(timeEllapsedSinceRefDate,
ar));

// create RelativeTabulatedAttitudeLeg
final RelativeTabulatedAttitudeLeg relativeTabulatedAttitudeLeg =
                new RelativeTabulatedAttitudeLeg(refDate, frame, listAr);
```

Getting Started

Building an attitude legs sequence

The attitude legs sequence was designed as a simple container, it performs only a few coherence checks on its inner attitude laws. Here's how an attitude sequence is built.

- An attitude legs sequence is created empty, associated to a single `PVCoordinatesProvider` instance.
- The sequence is mutable, attitude laws can be added to it one by one.
- Each attitude law is identified by a code.
- The sequence has a validity time interval, which is the grouping of the validity time intervals of all contained laws.
- The time interval of a newly added law must be contiguous to the grouped time interval of the already added laws. Otherwise an OrekitException is thrown.
- As soon as the sequence contains at least one law, methods of the `AttitudeLeg` interface can be called on the attitude sequence. The attitude sequence forwards the request to the appropriate attitude leg (according to the asking date), but replaces the `PVCoordinatesProvider` attribute of the call with the inner `PVCoordinatesProvider` instance.

AttitudeLawLeg and AttitudeLegsSequence : Code sample

```
final BodyCenterPointing earthCenterAttitudeLaw = new
BodyCenterPointing(itrf);
final AttitudeLeg law1 = new AttitudeLawLeg(earthCenterAttitudeLaw, date1,
date2);
final AttitudeLeg law2 = ... ;
final AttitudeLeg law3 = ... ;

PVCoordinatesProvider pvProvider = new CartesianOrbit(pvCoordinates, gcrf,
date1, mu);
final AttitudeLegsSequence sequence = new AttitudeLegsSequence(pvProvider);
// After each add the sequence has to be contiguous, so the order may be
important
sequence.add("L1", law1);
sequence.add("L2", law2);
sequence.add("L3", law3);

// Call to getAttitude on the sequence ignores otherPvProvider and uses
pvProvider internally instead
final Attitude sAttitude = sequence.getAttitude(otherPvProvider, date, itrf);
```

Contents

Interfaces

Interface	Summary	Javadoc
AttitudeLeg	This interface extends the AttitudeProvider interface and adds the time interval of validity notion to the attitude laws.	...

Classes

Class	Summary	Javadoc
AttitudeLawLeg	Object representing an attitude law for spacecraft attitude field purposes.	...
TabulatedAttitude	Object representing a tabulated attitude leg : the attitude at a date is interpolated from a list of known ones.	...
AttitudeLegsSequence	Object representing a sequence of contiguous attitude legs.	...
RelativeTabulatedAttitudeLeg	This class implements a tabulated attitude leg with relative dates.	...

Plus

Rechercher

Rechercher Lire

PATRIUS

- 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