User Manual 3.3 Time

De Wiki
Aller à : navigation, rechercher
User Manual 3.3 Time

Introduction

Scope

Despite its current use, time requires careful attention in the description of physical phenomena. Indeed the rate at which time passes has to be accurate in order to guarantee the best physical description of a phenomenon. To that purpose several time scales have been set up and with the introduction of atomic clocks, more accurate time scales have been defined.

Javadoc

The object related with dates and time scales are available in the package `org.orekit.time` of OREKIT.

Library	Javadoc
Orekit	Package org.orekit.time
Orekit addons	Package org.orekit.utils

Links

Orekit Time architecture description, Orekit site.

Useful Documents

None as of now.

Package Overview

This package overview is from the OREKIT website, under apache license.

Features Description

OREKIT Dates

In Orekit, the class `AbsoluteDate` represents a specific instant in time. There are different ways to create a date :

- for instance, one way is to give the date with a calendar form (year, month and day as 3 integers or as a `DateComponents` object) and a time scale : `AbsoluteDate(int, int, int, TimeScale)` or `AbsoluteDate(DateComponents, TimeScale)`
- to be more precise the informations of the hour, the minute and the second can be added :

`AbsoluteDate(int, int, int, int, int, double, TimeScale)` or `AbsoluteDate(DateComponents, TimeComponents, TimeScale)`

NB : a DateComponents object represents a date broken up as year, month and day ; a TimeComponents object represents a day broken up as hour, minute and second.

- another way is to give a predefined date and an elapsed duration from this date : `AbsoluteDate(AbsoluteDate, double)`

For advanced users only : a constructor `AbsoluteDate(long, double)` is available. It uses as parameters an "epoch" and an "offset", which are in fact the attributes of AbsoluteDate. These attributes have no meaning outside of AbsoluteDate, and are not meant to be exposed or manipulated directly by regular users. This constructor was added for the purpose of creating AbsoluteDate instances very quickly from a static storage (apart from serialization) and without loss of precision. Getters for "epoch" and "offset" are also available for this purpose.**Use at your own risk!**

As for the time scale, OREKIT counts 10 different time scales among which the TAI scale (International Atomic Time), the TT scale (Terrestrial Time), the UT1 scale (Universal Time), the UTC scale (Coordinated Universal Time)... The creation of a time scale is done through the `TimeScalesFactory` as follows : `TimeScale scale = TimeScalesFactory.getTT();`

Some reference epochs are directly available, for instance :

- J2000 epoch (2000-01-01T12:00:00) in TT scale : `AbsoluteDate date = AbsoluteDate.J2000_EPOCH;`
- julian epoch (-4712-01-01T12:00:00) in TT scale : `AbsoluteDate date = AbsoluteDate.JULIAN_EPOCH;`
- fifties epoch (CNES julian dates origin, 1950-01-01T00:00:00) in TT scale : `AbsoluteDate date = AbsoluteDate.FIFTIES_EPOCH_TT;`
- fifties epoch (CNES julian dates origin) in UTC scale : `AbsoluteDate date = AbsoluteDate.FIFTIES_EPOCH_UTC;`
- Java Reference epoch (1970-01-01T00:00:00) in UTC scale : `AbsoluteDate date = AbsoluteDate.JAVA_EPOCH;`
- ...

Time Interval

This implementation for time intervals, `AbsoluteDateInterval`, uses the Orekit `AbsoluteDate` class as the endpoint value type.

Infinite endpoints (`AbsoluteDate.PAST_INFINITY` and `AbsoluteDate.FUTURE_INFINITY`) are supported, but as open endpoints only. Empty intervals are also forbidden.

This implementation extends the class `ComparableInterval`, inheriting the following operations:

- check if two intervals are equal:

```
boolean equal = interval.equals(Object)
```

- check if two intervals overlap:

```
boolean overlaps = interval.overlaps(ComparableInterval<T>)
```

- check if the interval includes another interval:

```
boolean includes = interval.includes(ComparableInterval<T>)
```

- check if the interval is connected to another interval (its lower point coincides with the upper point of the input interval, and one point is closed and the other open):

```
boolean connected = interval.isConnectedTo(ComparableInterval<T>)
```

- compare the lower end point with the lower end point of another interval:

```
int compare = interval.compareLowerEndTo(ComparableInterval<T>)
```

- compare the upper end point with the upper end point of another interval:

```
int compare = interval.compareUpperEndTo(ComparableInterval<T>)
```

In addition to inherited capabilities, the class can also:

- compute the duration of the interval in seconds (as computed in a regular timescale- the duration has a physical meaning):

```
double duration = interval.getDuration()
```

- compute the duration between intervals in seconds (that is : the duration between the end of the earlier interval and the beginning of the later one):

```
double duration = interval.durationFrom(AbsoluteDateInterval nextInterval)
```

- merge the interval with another interval if possible (the intervals must overlap or be connected to be merged):

```
AbsoluteDateInterval mergedInterval = interval.mergeTo(AbsoluteDateInterval)
```

- get the intersection between two intervals when overlapping:

```
AbsoluteDateInterval intersection =
interval.getIntersectionWith(AbsoluteDateInterval)
```

- compare the duration of an interval with the duration of another interval:

```
int compare = interval.compareDurationTo(AbsoluteDateInterval)
```

List of time intervals

The class `AbsoluteDateIntervalsList` represents a list of time intervals (whose elements are `AbsoluteDateInterval` instances).
As `AbsoluteDateInterval` objects implement the `Comparable` interface via the class `ComparableInterval`, the time intervals in the list are automatically ordered by their lower/upper dates.

```
        final AbsoluteDateIntervalsList list = new
AbsoluteDateIntervalsList();
        IntervalEndpointType open = IntervalEndpointType.OPEN;
```

```
        IntervalEndpointType closed = IntervalEndpointType.CLOSED;
        // set up the time intervals to add:
        final AbsoluteDateInterval i1 = new AbsoluteDateInterval(open, date1,
date2, closed);
        final AbsoluteDateInterval i2 = new AbsoluteDateInterval(closed,
date1, date3, open);
        list.add(i1);
        list.add(i2);
```

In addition to the methods inherited from the `TreeSet` class, `AbsoluteDateIntervalsList` can also:

- tell which time intervals in the list contain a predetermined date:

```
AbsoluteDateIntervalsList intervals = list.getIntervalsContainingDate(date)
```

- compute the shortest interval containing all the intervals belonging to the list:

```
AbsoluteDateInterval inclusiveInterval = list.getInclusiveInterval();
```

- compute the list of complementary intervals of the given intervals list:

```
AbsoluteDateIntervalsList complementaryList =
list.getComplementaryIntervals();
```

Examples

```
        final AbsoluteDate t1 = new AbsoluteDate("1969-11-03",
TimeScalesFactory.getTT());
        final AbsoluteDate t2 = new AbsoluteDate("1969-11-04",
TimeScalesFactory.getTT());
        final AbsoluteDate t3 = new AbsoluteDate("1969-11-06",
TimeScalesFactory.getTT());
        final AbsoluteDate t4 = new AbsoluteDate("1969-11-07",
TimeScalesFactory.getTT());
        final IntervalEndpointType open = IntervalEndpointType.OPEN;
        final double dayInSeconds = Constants.JULIAN_DAY;
        // Two separated intervals, two days of separation :
        // ..] t1 ; t2 [....] t3 ; t4 [..
        final AbsoluteDateInterval ad1A = new AbsoluteDateInterval(open, t1,
t2, open);
        final AbsoluteDateInterval ad1B = new AbsoluteDateInterval(open, t3,
t4, open);
        final double dur11 = ad1B.durationFrom(ad1A);
        final double dur12 = ad1A.durationFrom(ad1B);
        // ad1B begins 2 days after ad1A ends : duration from is + 2 days
        Assert.assertEquals(2.* dayInSeconds, dur11, 0.);
        // ad1A ends 2 days before ad1B begins : duration from is- 2 days
        Assert.assertEquals(-2.* dayInSeconds, dur12, 0.);
```

Please see the Javadoc for more information.

Local time

Local time is the angle between the projections of the Earth-Sun vector and the Earth-spacecraft vector in the equatorial plane. Local time angle is zero at 12.00h and is PI at 0.00h.
Local time increases for prograde orbits and decreases for retrograde orbits.
Solar time is the angle between the Earth-Sun projection in the orbital plane and the Earth-spacecraft vector. Solar time angle is zero at 12.00h and is PI at 0.00h.
Solar time always increases with time. It is mesured around the orbit momentum.

The class `LocalTime` provides methods to compute:

- True local time using `computeTrueLocalTime()`. Local time is always computed in TIRF frame. True local time is the angle between projection of satellite position and Sun position over the equatorial plane in provided frame. Returned time is expressed in seconds in the range [0s; 86400s[.
- Mean local time using `computeMeanLocalTime()`. Local time is always computed in TIRF frame. Mean local time is the difference between true local time and equation of time (EOT):

[math]Mean local time = True local time + EOT[/math]

- Equation of time using `computeEquationOfTime()`:

Equation of time is true local time of GMST ([0; 0; 1] vector in TIRF frame) minus seconds in the date:

[math]EOT =-(Local time GMST - sec)[/math]

Equation of time is periodic over one year in the range [-16min; +14min]:

Getting Started

TBD

Contents

Interfaces

Interface	Summary	Javadoc
TimeScale	Interface for time scales.	...
TimeStamped	This interface represents objects that have a AbsoluteDate date attached to them.	...

Classes

Class	Summary	Javadoc
AbsoluteDate	This class represents a specific instant in time.	...
AbsoluteDateInterval	This class implements an interval based on the AbsoluteDate class.	...
AbsoluteDateIntervalsList	This class represents a list of AbsoluteDateInterval objects.	...

ChronologicalComparator	Comparator for TimeStamped instance.	...
ComparableInterval	Class describing an interval of Comparable data.	...
DateComponents	Class representing a date broken up as year, month and day components.	...
DateTimeComponents	Holder for date and time components.	...
GalileoScale	Galileo system time scale.	...
GenericInterval	Generic class to describe an interval.	...
GMSTScale	Greenwich Mean Sidereal Time.	...
GPSScale	GPS time scale.	...
TAIScale	International Atomic Time.	...
TCBScale	Barycentric Coordinate Time.	...
TCGScale	Geocentric Coordinate Time.	...
TDBScale	Barycentric Dynamic Time.	...
TimeComponents	Class representing a time within the day broken up as hour, minute and second components.	...
TimeScalesFactory	Factory for predefined time scales.	...
TTScale	Terrestrial Time as defined by IAU(1991) recommendation IV.	...
UT1Scale	Universal Time 1.	...
UTCScale	Coordinated Universal Time.	...
LocalTime	Local time computation methods.	...

Tutorials

Tutorial 1

[Tutoriels_Flight_Dynamics.Dates Tutoriel sur les dates]

Tips & Tricks

None yet !

Menu de navigation

Outils personnels

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 2 mars 2018 à 14:49.

- Politique de confidentialité
- À propos de Wiki
- Avertissements

-