

User Manual 4.15 Time

De Wiki

Aller à : [navigation](#), [rechercher](#)
[User Manual 4.15 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 `fr.cnes.sirius.patrius.time`.

Library

Javadoc

Patrius [\[\[\[:Modèle:JavaDoc4.15\]\]/fr/cnes/sirius/patrius/time/package-summary.html]([[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/package-summary.html) Package `fr.cnes.sirius.patrius.time`

Patrius [\[\[\[:Modèle:JavaDoc4.15\]\]/fr/cnes/sirius/patrius/utils/package-summary.html]([[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/utils/package-summary.html) Package `fr.cnes.sirius.patrius.utils`

Links

Time architecture description, [Orekit site](#).

Useful Documents

None as of now.

Package Overview

Please note that not all implementations are present in the following diagram for the sake of clarity.



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

Features Description

Dates

In Patrius, the class `AbsoluteDate` represents a specific instant in time. There are several 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)`
- through a CNES Julian date and associated timescale: `AbsoluteDate(double, TimeScale)`.
Note that the reversed conversion (from `AbsoluteDate` to CNES JD is also available with method `toCNESJulianDate()`

✘ 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, `Patrius` contains 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();`

Note that for UTC scale, some data are required (the UTC-TAI shift). This data can be handled through the `TimeScalesFactory` class. By default, some loaders are added to `TimeScalesFactory` in the following order:

- `UTCTAIHistoryFilesLoader`: UTC-TAI file in IERS format (file `UTC-TAI.history`)

There is no included data in `PATRIUS` by default Once loaded, data are stored in static variables and are used for UTC time scale retrieval.

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 `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)
```

- create a list of equally spaced dates within an interval:
 - `List<AbsoluteDate> datesList = interval.getDateList(double)` (if providing a step)
 - `List<AbsoluteDate> datesList = interval.getDateList(int)` (if providing a number of dates)

Boundaries type (open, closed) are taken into account (more information in the javadoc of these methods)

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();
```

Please see the Javadoc for more information.

Local time and solar time

Local time is represented by the angle between the projections of the Earth-Sun vector and the Earth-Spacecraft vector in the equatorial plane. Local time is expressed in hours and its value is 12h when this angle is 0 rad. Beware, in PATRIUS, local time is always expressed as an angle in the range $[-\pi; \pi]$ and always represents the angle between the projections of the Earth-Sun vector and the Earth-Spacecraft vector in the equatorial plane. If one wants to retrieve local time (in hours) from the local time angle provided by PATRIUS, the following formula should be used:

$$\text{Local time} = 12\text{h} + \text{local time angle} * 12 / \pi.$$

Local time increases for prograde orbits and decreases for retrograde orbits.

Similarly, solar time is represented by the angle between the Earth-Sun projection in the orbital plane and the Earth-Spacecraft vector. Solar time is expressed in hours and its value is 12h when this angle is 0 rad. Beware, in PATRIUS, solar time is always expressed as an angle in the range $[-\pi; \pi]$ and always represents the angle between the Earth-Sun projection in the orbital plane and the Earth-Spacecraft vector. If one wants to retrieve solar time (in hours) from the solar time angle provided by PATRIUS, the following formula should be used:

$$\text{Solar time} = 12\text{h} + \text{solar time angle} * 12 / \pi.$$

Solar time always increases with time. It is measured around the orbit momentum.

The class `LocalTimeAngle` provides methods to compute:

- True local time angle using `computeTrueLocalTimeAngle()`. Local time is always computed in TIRF frame. True local time is represented by 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 $[-\pi; \pi]$.
- Mean local time angle using `computeMeanLocalTimeAngle()`. Local time angle is always computed in TIRF frame. Mean local time is the sum of true local time and equation of time (EOT):

$$\text{Mean local time} = \text{True local time} + \text{EOT}$$

- 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:

$$\text{EOT} = -(\text{Local time GMST} - \text{sec})$$

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



- RAAN from local true/mean time with methods `computeRAANFromMeanLocalTime` and `computeRAANFromTrueLocalTime`.

Getting Started

AbsoluteDateInterval

Here is given a code example on how to use the AbsoluteDateInterval:

```
    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.);
```

Contents

Interfaces

| Interface | Summary | Javadoc |
|--------------------|---|--|
| TimeScale | Interface for time scales. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TimeScale.html ...] |
| TimeStamped | This interface represents objects that have a AbsoluteDate date attached to them. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TimeStamped.html ...] |

Classes

| Class | Summary | Javadoc |
|-------|---------|---------|
|-------|---------|---------|

| | | |
|----------------------------------|---|--|
| AbsoluteDate | This class represents a specific instant in time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/AbsoluteDate.html ...] |
| AbsoluteDateInterval | This class implements an interval based on the AbsoluteDate class. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/AbsoluteDateInterval.html ...] |
| AbsoluteDateIntervalsList | This class represents a list of AbsoluteDateInterval objects. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/AbsoluteDateIntervalsList.html ...] |
| ChronologicalComparator | Comparator for TimeStamped instance. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/ChronologicalComparator.html ...] |
| ComparableInterval | Class describing an interval of Comparable data. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/math/interval/ComparableInterval.html ...] |
| DateComponents | Class representing a date broken up as year, month and day components. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/DateComponents.html ...] |
| DateTimeComponents | Holder for date and time components. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/DateTimeComponents.html ...] |
| GalileoScale | Galileo system time scale. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/GalileoScale.html ...] |
| GenericInterval | Generic class to describe an interval. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/utis/GenericInterval.html ...] |
| GMSTScale | Greenwich Mean Sidereal Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/GMSTScale.html ...] |
| GPSScale | GPS time scale. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/GPSScale.html ...] |
| TAIScale | International Atomic Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TAIScale.html ...] |
| TCBScale | Barycentric Coordinate Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TCBScale.html ...] |
| TCGScale | Geocentric Coordinate Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TCGScale.html ...] |
| TDBScale | Barycentric Dynamic Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TDBScale.html ...] |
| TimeComponents | Class representing a time within the day broken up as hour, minute and second components. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TimeComponents.html ...] |
| TimeScalesFactory | Factory for predefined time scales. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TimeScalesFactory.html ...] |
| TTScale | Terrestrial Time as defined by IAU(1991) recommendation IV. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/TTScale.html ...] |
| UT1Scale | Universal Time 1. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/UT1Scale.html ...] |
| UTCScale | Coordinated Universal Time. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/UTCScale.html ...] |
| LocalTimeAngle | Local time angle computation methods. | [[[:Modèle:JavaDoc4.15]]/fr/cnes/sirius/patrius/time/LocalTimeAngle.html ...] |

Tutorials

Tutorial 1

[Tutoriel sur les dates](#)

Récupérée de « http://patrius.cnes.fr/index.php?title=User_Manual_4.15_Time&oldid=3857 »
 Catégorie :

- [User Manual 4.15 Flight Dynamics](#)

Menu de navigation

Outils personnels

- [3.144.45.187](#)
- [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

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 21 novembre 2024 à 15:31.
- [Politique de confidentialité](#)
- [À propos de Wiki](#)
- [Avertissements](#)

- 