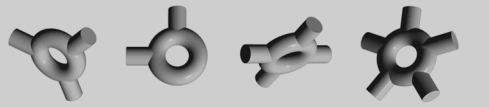
elements 

Dates Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

The Dates package provides the core classes for dates, date printing and date arithmetic. These classes form the basic structures to which more complex business rules (such as rolling to a business day) can be added. The Date package also provides the classes for performing term arithmetic over non-Actual/Actual date bases.

Currently, only the standard 'common era' (CE) calendar is supported. Structural differences in calendar conventions (eg. for the Islamic or Asian calendars) are not supported. Days of the week follow the ISO 8601 conventions, rather than the more common Sunday = 1 convention.[2]

Most financial transactions occur at times in the future. For example, the maturity date of a spot FX deal is, normally, two business days after the date the deal was made on. Other transactions are composed of complex sequence of transactions at several future dates. For example, the coupon payments and the return of principal of a bond.

Periods encode the specification of these future dates. Periods add rolling conventions to the ordinary addition of an amount of time to some date. Periods may also be composite — describing the addition of a series of periods to a date — or repeating — describing a sequence of dates.

When a date falls on a non-business day, or a day that is unacceptable for some other reason, then the date needs to be *rolled* to a new date that is acceptable.

The basic mechanisms for date rolling are also discussed in this package. This package allows extended date rolling conventions, where the conventions can be constructed from a general program-like model, allowing date rolling conventions of arbitrary complexity.

Date classification for weekends and holidays is also modelled.

Basic weekends consist of lists of the days of the week that are considered to be weekends. Holidays and more complex weekends can be constructed in a number of ways: as a particular date within some period, relative to another holiday or as an arbitrary list of dates.

In some cases, groups of holidays and weekends can interact. Holidays which fall on weekends can be moved to business days. Holidays which fall on another holiday can be moved to other days. Working out which holidays move under

Unit	Amount	12-Jan-98	1-Jun-98	1-Dec-98	25-Dec-98
days	1	13-Jan-98	2-Jun-98	2-Dec-98	28-Dec-98
days	-1	12-Jan-98	1-Jun-98	30-Nov-98	24-Dec-98
days	38	19-Feb-98	8-Jul-98	8-Jan-99	1-Feb-99
bdays	2	14-Jan-98	3-Jun-98	3-Dec-98	29-Dec-98
bdays	-1	9-Jan-98	29-May-98	30-Nov-98	24-Dec-98
bdays	7	21-Jan-98	10-Jun-98	10-Dec-98	5-Jan-99
nwdays	1	12-Jan-98	2-Jun-98	2-Dec-98	28-Dec-98
nways	-1	9-Jan-98	29-May-98	30-Nov-98	24-Dec-98
nways	12	28-Jan-98	17-Jun-98	17-Dec-98	12-Jan-99
weeks	1	19-Jan-98	8-Jun-98	8-Dec-98	4-Jan-99
weeks	-1	5-Jan-98	25-May-98	24-Nov-98	18-Dec-98
months	1	12-Feb-98	1-Jul-98	2-Jan-99	25-Jan-99
months	-1	12-Dec-97	1-May-98	2-Nov-98	25-Nov-98
months	5	12-Jun-98	2-Nov-98	3-May-99	25-May-99
years	1	12-Jan-99	1-Jun-99	1-Dec-99	27-Dec-99
years	-1	13-Jan-97	2-Jun-97	1-Dec-97	26-Dec-97
years	10	14-Jan-08	2-Jun-08	1-Dec-08	26-Dec-08

Table 1: Example Date Arithmetic

such circumstances can lead to cycles of interacting holidays. To prevent cycles, holidays and weekends are assigned an *order*.

Holidays and weekends are considered to fall on single dates from midnight to midnight. Holidays which run from (for example) evening to evening are assumed to fall on the day that contains the business portion of the holiday. The dates that holidays fall on are expected to be determinable before the holiday actually occurs.

1 Use Cases

1.1 Simple Date Arithmetic

Assuming Saturday and Sunday as weekends, the first of January and 25th of December as holidays and a following date rolling convention, example date arithmetic is shown in table 1.

1.2 Day Count Conventions

The elapsed time between two dates is often calculated by using some convention other than the actual number of days. Common day count conventions are:

Actual The actual number of days between two dates.

NL Non-leap year. Any leap-days are ignored when calculating the elapsed time.

30 30 day month. All months are treated as being 30 days long. If dates fall on the 31st, then the date is adjusted.

30E 30 day month (European convention). Dates falling on the 31st obey different adjustment rules to the standard 30 day basis.

30 PSA 30 day month (Public Securities Association convention). Dates falling on the 31st obey different adjustment rules to the standard 30 day basis.

Sample day count calculations for each convention are shown in table 2.

1.3 Date Rolling Conventions

Date rolling conventions describe the process used to move a date that falls on a holiday or weekend to a new business day. Common conventions are:

Following Move the date on to the closest business day after the date.

Preceding Move the date on to the closest business day before the date.

Modified Following Move the date on to the closest business day after the date, unless the new date crosses a month boundary, in which case move the date back to the closest business day before the date.

Modified Preceding Move the date on to the closest business day before the date, unless the new date crosses a month boundary, in which case move the date forwards to the closest business day after the date.

Null Leave the date as it is.

From	To	Actual	NL	30	30E	30 PSA
1-Jan-1986	1-Feb-1986	31	31	30	30	30
1-Jan-1986	1-Jan-1987	365	365	360	360	360
15-Jan-1986	1-Feb-1986	17	17	16	16	16
1-Feb-1986	1-Mar-1986	28	28	30	30	30
15-Feb-1886	1-Apr-1986	45	45	46	46	46
15-Mar-1986	15-Jun-1986	92	92	90	90	90
31-Mar-1986	1-Apr-1986	1	1	1	1	1
31-Mar-1986	30-Apr-1986	30	30	30	30	30
31-Mar-1986	31-Dec-1986	275	275	270	270	270
15-Jul-1986	15-Sep-1986	62	62	60	60	60
21-Aug-1986	11-Apr-1987	233	233	230	230	230
1-Nov-1986	1-Mar-1987	120	120	120	120	120
15-Dec-1986	30-Dec-1986	15	15	15	15	15
15-Dec-1986	31-Dec-1986	16	16	16	15	16
31-Dec-1986	1-Feb-1987	32	32	31	31	31
1-Feb-1988	1-Mar-1988	29	28	30	30	30
30-Aug-1991	31-Aug-1991	1	1	0	0	0
31-Aug-1991	27-Feb-1992	180	180	177	177	177
31-Aug-1991	28-Feb-1992	181	181	178	178	178
31-Aug-1991	29-Feb-1992	182	181	179	179	179
31-Aug-1991	1-Mar-1992	183	182	181	181	181
15-Jan-1992	28-Feb-1992	44	44	43	43	43
15-Jan-1992	29-Feb-1992	45	44	44	44	44
15-Jan-1992	1-Mar-1992	46	45	46	46	46
1-Apr-1992	15-Jul-1992	105	105	104	104	104
29-Feb-1992	2-Mar-1992	2	2	3	3	2

Table 2: Example Day Count Calculations

1.4 Date Rolling Example - Modified Following

The *modified following* date rolling convention has the following definition:

If the current date is not a business day, then move the date forward to the closest following date that is a business day. If, as a result of rolling the date forward, the date passes into another month, then roll the date backwards instead.

1.5 Date Rolling Example - Following

The *following* date rolling convention has the following definition:

If the current date is not a business day, then move the date forward to the closest following date that is a business day.

1.6 Date Rolling Example - FX Following

The *FX following* date rolling convention has the following definition:

An FX period is calculated from the spot date (eg. FX 1 month is spot date + 1 month). If the date is not a business day, then move the date forward to the closest following date that is a business day. If the spot date, before any other period was added, falls at the end of the business month, then force the rolled date to the end of the business month.[3]

Note that the FX following convention should not be used by periods of less than a month.

1.7 FX Spot

The spot date for foreign exchange transactions is normally two business days after the date on which the agreement was made.

Deals involving USD or CAD have different spot date rules. The USD spot date is two non-weekend days after the trade date. The CAD spot date is one business day after the trade date.

1.8 FX One Month

The one month period for foreign exchange transactions involves first adding the FX Spot period to the current date and then adding one month to that date. If the resulting date is on a non-business day, then the date is rolled forward to the following business day. If the date calculated after the addition of the spot date falls at the end of a month, then the rolled date is also forced to the end of the month. eg. Deal Date = 29th May. Spot = 31st May = End of Month. +1M = 30 June.

If necessary, the forward value date is bought back to the nearest previous business day in order to stay within the same calendar month rather than move forward to the beginning of the next month.

eg. Deal Date = 28th Jan. Spot = 30th Jan. +1M = 1st Mar, roll this back to 28th (or 29th) Feb.

1.9 Six one month payments

One month is added repeatedly to the current date, with each resulting date being rolled forward to a business day.

1.10 3 month payments until 12-Dec-2000

Three months are added to the current date until 12-Dec-2000 is reached or passed. The last payment date is 12-Dec-2000.

1.11 Saturday and Sunday

The standard Western weekend. Any day falling on a Saturday or Sunday is considered to be a weekend day.

1.12 Friday and Saturday

The extended Islamic weekend. Any day which falls on a Friday or Saturday is considered to be a weekend.

1.13 Friday, Saturday and Sunday

The result of a trade between two countries, one of which observes a Friday and Saturday weekend, the other of which observes a Saturday and Sunday weekend.

1.14 Malaysian Weekends

In Malaysia, Sundays and the first Saturday in the month are weekend days.

1.15 Taiwanese Weekends

In Taiwan, the weekend consists of each Sunday and the second and fourth Saturday of each month.

1.16 Lithuanian Holiday Weekends

In Lithuania, if there is a holiday on Thursday, then Friday is a weekend day and Saturday a working day, in contrast to the normal Saturday and Sunday weekend.

1.17 Christmas

Christian Religious Holiday. Falls on the 25th of December each year.

1.18 Easter (Western)

Western Christian Religious Holiday.

Easter tends to be somewhere in either March or April. Definitionally, this holiday is held on the Sunday following the full moon on or after the vernal equinox.

Rather than use the astronomical definition, the church uses standardized tables of ecclesiastical moons so that the dates of Easter can be determined in advance.

See http://aa.usno.navy.mil/AA/faq/docs/easter.html

1.19 Good Friday (Western)

Western Christian Religious Holiday.

The Friday immediately before Easter Sunday.

1.20 Yom Kippur

Jewish Religious Holiday.

Beginning at sunset on Tishri 9 of the Jewish calendar and lasting until three stars are visible on Tishri 10.

1.21 Independence Day

United States National Holiday.

Held on the 4th of July each year.

1.22 Melbourne Cup Day

Victorian (Australia) State Holiday.

Held on the first Tuesday of November each year.

1.23 Greenery Day

Japanese National Holiday.

Falls on the 29th of April of each year. If the 29th of April is a Sunday then the holiday is moved to the following Monday.

1.24 New Year (Coptic)

Coptic Christian Religious Holiday.

Falls on the 11th of September, or the 12th of September during leap-years.

1.25 New Year (Chinese)

Chinese and East Asian National Holiday.

Celebrated on the first day of the Chinese lunar calendar.

1.26 Beltane

Wiccan Religious Holiday.

Either the 30th of April or the 1st of May each year, depending upon Wiccan convention.

1.27 Noruz

Persian Calendar New Year.

Occurs on the day of the vernal equinox - the 20th to the 22nd of March.

1.28 Islamic Holidays

Islamic Religious Holidays.

The holidays of Islam, as well as the start of the month are decided by religious authorities, based on the actual observation of the moon. Holiday dates may vary from country to country.

2 Interfaces

2.1 Date

Date objects represent dates without any additional time information being attached. For most financial calculations, date calculations are more useful than time calculations.

Dates obey value semantics. Any operation on a date such as addition or subtraction produces a new date, rather than altering the internal structure of the date object.

The date specifications generally follow ISO 8601[2], except where more configurability is allowed.

2.1.1 Relationships

	Class	Description	Notes
⇑	Datestamp		
⇑	ValueSemantics		
↓	DateModel §3.17		

⇑:Inherits ↓:Realized by

2.1.2 Operations

Boolean equals(Object arg) equals
arg: Object

Equality test. Two dates are equal if they refer to the same date.

Boolean lessThanOrEqualTo(Date) lessThanOrEqualTo
Date

Less than or equal to relationship. Date a is less than or equal to date b if a precedes b or a and b fall on the same date.

Integer day() day

Day of month. Return the day within the month, with the first day of the month being 1.

MonthEnum month() month

Month of year. Return the month that the date falls into.

Integer year() year

Year number. Return the year number, including the century. 1 CE returns 1, 1 BCE returns 0.

Integer dayOfYear() dayOfYear

Day within year. Return the number of days from the start of the year, with 1st January being 1 and 31st of December being 365 or 366, depending on whether

this year is a leap year or not.

DayOfWeekEnum dayOfWeek() dayOfWeek

Day of week. Return the day of the week that this date falls upon.

Integer century() century

Century number. Return the century number in CE terms. Years 1-100 CE return 1 (1st century), the year 2000 returns 20 (20th century).

QuarterEnum quarter() quarter

Quarter of the year. Return the quarter of the year, with January - March being the first quarter through to October–December being the fourth.

print(OutputStream stream, DateFormat format) print

stream: OutputStream The stream to print onto.
format: DateFormat The format to use when printing the date.

Print a string representation of this date according to a date format. Add this date to the output stream, formatted by the supplied format.

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Standard text representation. Print using the DateFormat.local() date format.

printShort(OutputStream stream) printShort

stream: OutputStream The stream to print onto.

Short text representation. Print this date using the DateFormat.localShort() date format.

printLong(OutputStream stream) printLong

stream: OutputStream The stream to print onto.

Long text representation. Print this date using the DateFormat.localLong() date format.

printVeryLong(OutputStream stream) printVeryLong

stream: OutputStream The stream to print onto.

Full text representation. Print this date using the DateFormat.localVeryLong() date format.

Date add(PeriodWithRoller periodWithRoller, DateClassifier classifier)

add

periodWithRoller: PeriodWithRoller A period with roller contains a period to add to the date and the roller indicating the rolling to be done for the date.
classifier: DateClassifier Date classifier for date rolling. The default value is NullDateClassifierModel.default().
Raises: ImmobileDateException

Add an amount of a period (with a rolling convention). The returned date is this date plus the amount given. Once calculated, the date is rolled to a suitable date according to the supplied date roller and classifier.

Date nextBusinessDay(DateClassifier classifier)

nextBusinessDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the next business day from this date. Return the closest date (that is not this date) that is a business day following this date.

Date previousBusinessDay(DateClassifier classifier)

previousBusinessDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the previous business day to this date. Return the closest date (that is not this date) that is a business day preceding this date.

Date nextNonWeekendDay(DateClassifier classifier)

nextNonWeekendDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the next non-weekend day from this date. Return the closest date (that is not this date) that is a non-weekend day following this date.

Date previousNonWeekendDay(DateClassifier classifier)

previousNonWeekendDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the previous non-weekend day to this date. Return the closest date (that is not this date) that is a non-weekend day preceding this date.

Date closestBusinessDay(DateClassifier classifier)

closestBusinessDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the closest business day to this date. Return the closest date (that is not this date) that is a business day either preceding or following this date. If both dates are equally close, then the following date is returned.

Date closestNonWeekendDay(DateClassifier classifier) closestNon-WeekendDay

classifier: DateClassifier Classifier to determine the type of day.

Date which is the closest non-weekend day to this date. Return the closest date (that is not this date) that is a non-weekend day either preceding or following this date. If both dates are equally close, then the following date is returned.

Integer daysInMonth() daysInMonth

Number of days in the month for this date.

> 30 days hath September,
> April, June and November.
> All the rest have 31.
> Except February alone,
> Which has 28 days,
> And 29 days on each leap year.

Boolean isLeapYear() isLeapYear

This date falls in a leap year?

The rules for leap years are as follows:

Any year evenly divisible by 4 is a leap year, except any year evenly divisible by 100 is not a leap year, except any year evenly divisible by 400 is a leap year.

(An additional rule for every 4000 year is being considered)

Examples:

1999 is not a leap year. 1996 is a leap year. 2100 is not a leap year. 2000 is a leap year.

Integer daysInYear() daysInYear

Number of days in this date's year. Return 366 if the year is a leap year, 365 otherwise.

2.2 DateBasis

A date basis encodes the conventions used when calculating the elapsed time between two dates, usually for interest rate calculations.

The most obvious date basis is Actual/Actual; you calculate the actual number of days between the two dates to get the term in days and divide it by the actual

year length to get the term in years. However, certain financial instruments use other conventions.

Date bases are essentially stateless and obey ValueSemantics.

2.2.1 Relationships

	Class	Description	Notes
⇑	ValueSemantics		
↓	DateBasisModel §3.1		

⇑:Inherits ↓:Realized by

2.2.2 Operations

Boolean equals(Comparable arg) equals
arg: Comparable

The equality relationship. Two date bases are equal if they would return the same day and year counts for any arbitrary pair of dates. In practical terms, equality means the same day- and year-count conventions.

print(OutputStream stream) print
stream: OutputStream The stream to print onto.

Print the object. Date bases are conventionally printed by first printing the day count convention, printing a slash and then the year count convention.

Eg. “30E/Actual” or “30/360”.

DayBasis dayBasis() dayBasis

The day count convention. Return the day count convention used by this date basis.

YearBasis yearBasis() yearBasis

Year length convention. Return the year length convention used by this date basis.

Integer termInDays(Date from, Date to) termInDays
from: Date The start date of the period.
to: Date The end date of the period.

Day count between two dates. Returns the number of days between from and to, according to this date basis’ conventions.

Double termInYears(Date from, Date to) termInYears
from: Date The start date of the period.
to: Date The end date of the period.

Year count between two dates. Returns the number of years between from and to, according to this date basis' conventions.

2.3 DateClassifier

In business date calculations, it is necessary to distinguish between business days and non-business days. Non-business days may be either weekend days or holidays. What is a business day or non-business day will change from location to location and financial instrument type to instrument type. Date classifiers provide the means of deciding whether a day is a business day or not.

A classifier can class a date as both a holiday and a weekend.

Date classifiers may be used recursively in some holiday definitions, so that holidays which depend on other holidays or which are moved when they conflict with another holiday can be correctly handled. Date classifiers are partially ordered by *precedence*. A date classifier which is "less than" another date classifier by precedence is moved to avoid a clash. The *environment* of a date classifier is the set of potentially clashing holidays within a location (or, in the case of a union of two classifiers, locations).

Usually, date classifiers classify according to a single location. However, some instruments (eg. FX) are dependent upon the business conventions of a number of different locations. As a result, classifiers need to be composable; two classifiers can be combined to provide a single classifier. Classifiers that are not composites are *simple*.

There are two methods for joining date classifiers together. The composition of two date classifiers merges the weekends and holidays of a single location, taking precedence into account. Within a location, certain holidays have precedence over other holidays. This can cause holidays to move to the following day. Thus, composition can be used to construct all of the holidays, correctly rolled, within a location.

The union of two date classifiers merges the holidays and weekends of two locations together, ignoring precedence issues. With FX deals, two currencies hence, two locations, are involved in a transaction. Both locations' holidays and weekends need to be known when determining the matuirty date of a deal. There are no precedence concerns between holidays and weekends in two different locations The union of two date classifiers will contain both locations' holidays and weekends.

2.3.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇓	SimpleDateClassifier §2.4		
↓	NullDateClassifierModel §3.44		
↓	DateClassifierWeekendModel §3.5		
↓	DateClassifierUnionModel §3.3		
↓	DateClassifierReferenceData-Model §3.2		
↔	DateClassifierReferenceData-Model §3.2	model 0..1	
↔	DateClassifierUnionModel §3.3		
↔	DateClassifierUnionModel §3.3	components 0..n	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◇:Aggregate ♦:Composite

2.3.2 Operations

Boolean isHoliday(Date date, DateClassifier environment) isHoliday
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

Date falls on a holiday? Return true if this classifier classifies date as a holiday, false otherwise.

Boolean isWeekend(Date date, DateClassifier environment) isWeekend
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

Date falls on a weekend? Return true if this classifier classifies date as a weekend, false otherwise.

Boolean isNonBusinessDay(Date date, DateClassifier environment) isNonBusiness-Day
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

A date that falls on a holiday or a weekend is classified as a non-business day, and will return true.

$$isNonBusinessDay \Leftrightarrow isHoliday \vee isWeekend$$

Boolean isBusinessDay(Date date, DateClassifier environment) isBusinessDay
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

Date falls on a business day? A business day is classified as a day which does not fall on either a holiday nor a weekend.

$$isBusinessDay \Leftrightarrow \neg isNonBusinessDay$$

Boolean isSimple() isSimple

Return true if this date classifier is neither a composite or union of other date classifiers. Return false otherwise.

DateClassifier compose(DateClassifier arg) compose
arg: DateClassifier The date classifier to compose this classifier with.

Compose two date classifiers so that the two classifiers are affected by precedence considerations. This operation has similar semantics to the union() operation, however, certain holidays may be moved if they clash with other holidays.

Boolean precedes(DateClassifier other) precedes
other: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. Return true if this date classifier has a higher precedence than the other date classifier.

DateClassifier predecessorsOf(DateClassifier test) predecessorsOf
test: DateClassifier The date classifier to test against.

The predecessors of a date classifier. Return a date classifier that is equivalent to removing all the basic date classifiers that make up this classifier that do not preceed the test classifier. If there are no such classifiers, return an instance of the NullDateClassifier.

DateClassifier union(DateClassifier arg) union

arg: DateClassifier The date classifier to combine this classifier with.

Join two date classifiers so that the classifiers are unaffected by precedence considerations. Let c be the join of this classifier (a) and the argument date classifier, (b).

$$c.isHoliday(date) \Leftrightarrow a.isHoliday(date) \vee b.isHoliday(date)$$

$$c.isWeekend \Leftrightarrow a.isWeekend(date) \vee b.isWeekend(date)$$

2.4 SimpleDateClassifier

A variant of date classifier that computes a date for a holiday or weekend, relative to a supplied date and tests the supplied date against the holiday.

2.4.1 Relationships

	Class	Description	Notes
⇑	DateClassifier §2.3		
⇑	Identifiable		
↓	SimpleDateClassifierModel §3.63		
↔	SimpleDateClassifierModel §3.63	predecessors 0..n	
↔	DateClassifierHolidayRelative-Model §3.66	reference holiday 1..1	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.4.2 Operations

Boolean isWeekend(Date date, DateClassifier environment) isWeekend

date: Date The date to test.

environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a weekend? Return true if weekendFlag() is true and the argument date is the same as the reference date for date and environment, false otherwise.

Boolean isHoliday(Date date, DateClassifier environment) isHoliday

date: Date The date to test.

environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a holiday? Return true if weekendFlag() is false and the argument date is the same as the date that a holiday is found to fall on.

Date nthClosestHolidayTo(Date date, Integer n, DateClassifier environment) nthClosestHolidayTo

date: Date The reference date for computing the holiday.
n: Integer The nth closest holiday to a date.
environment: DateClassifier The set of potentially clashing holidays within a location. The default value is NullDateClassifierModel.defaultInstance().

Return the nth closest holiday to the date, aDate. If there are no holidays, return null.

When *n* is positive, it indicates that you are looking for the nth closest holiday $after$ the date, aDate. When *n* is negative, it indicates that you are looking for the nth closest holiday $before$ the date, aDate.

Return the date that this holiday specifies, for the supplied date (eg. if the holiday falls, annually, on the 27th of July then the closest holiday for the 12th of June 2023 is the 27th of July 2023).

Boolean weekendFlag() weekendFlag

The date is a weekend, instead of a holiday? Return true if this date classifier classifies a weekend, rather than a holiday.

Boolean isSimple() isSimple

This classifier is simple? Return true.

DateClassifier compose(DateClassifier arg) compose

arg: DateClassifier The date classifier to compose this classifier with.

Compose two date classifiers so that the two classifiers are affected by precedence considerations. Return a DateClassifierCompositeModel §3.4 with this classifier and the argument classifier as components. Thus, all holidays within a location have been consolidated and rolled where two holidays within the one location fall on the same day.

DateClassifier union(DateClassifier arg) union

arg: DateClassifier The date classifier to combine this classifier with.

Join two date classifiers so that the classifiers are unaffected by precedence

considerations. Return a DateClassifierUnionModel §3.3 with this classifier and the argument classifier as components.

2.5 DateFormat

An object that implements the DateFormat interface can be used to print dates in whatever form is appropriate for a locality.

2.5.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
↓	DateFormatModel §3.15		
↓	DateFormatReferenceData-Model §3.16		
↔	DateFormatReferenceData-Model §3.16	model 0..1	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.5.2 Operations

String dayName(Integer day) dayName
day: Integer Day number.

Long day of month. Return the day of the month in text format, with a suitable suffix appended.

Eg. 1 becomes “1st”, 12 becomes “12th”, 22 becomes “22nd”

String dayOfYearName(Integer day) dayOfYear-Name
day: Integer Day number.

Long day of year. Return the day of the year in text format, with a suitable suffix appended.

Eg. 1 becomes “1st”, 120 becomes “120th”, 22 becomes “22nd”

String dayOfWeekName(DayOfWeekEnum day) dayOfWeek-Name
day: DayOfWeekEnum Day of week

Short day of the week name. Return the day of the week in an abbreviated text format.

Eg. In English, monday becomes “Mon”, thursday becomes “Thu”, etc.

String longDayOfWeekName(DayOfWeekEnum day) longDay-OfWeekName
day: DayOfWeekEnum Day of week.

Full day of the week name. Return the day of the week in an full text format.

Eg. In English, monday becomes “Monday”, thursday becomes “Thursday”, etc.

String monthName(MonthEnum month) monthName
month: MonthEnum The month in the year.

Short month name. Return the month name in an abbreviated text format.

Eg. In English, january becomes “Jan”, april becomes “Apr”, etc.

String longMonthName(Integer month) longMonth-Name
month: Integer Month number. (1=January to 12=December)

Full month name. Return the full month name.

Eg. In English, january becomes “January”, april becomes “April”, etc.

String yearName(Integer year) yearName
year: Integer Year number.

Possibly abbreviated year name. Returns the conventionally abbreviated form of the year if the year falls within 50 years of the current date, otherwise return the result of longYearName().

Eg. in English, during the year 2000, 1998 will result in “98”, 2035 in “35”, 1937 in “1937” and 1815 in “1815”.

String longYearName(Integer year) longYearName
year: Integer Year number.

Full year name. Returns the full 4-digit year in string format. For years under 1000, preceding 0s are added to make a 4-digit year.

Eg. in English, during the year 2000, 1998 will result in “1998”, 2035 in “2035”, 1937 in “1937” and 1815 in “1815”.

String centuryName(Integer century) centuryName
century: Integer Century number. (1st century = 1, 20th century = 20)

Name of the century. Return the century in text format, with a suitable suffix appended.

Eg. 1 becomes “1st”, 20 becomes “20th”, 22 becomes “22nd”

String quarterName(QuarterEnum quarter) quarterName

quarter: QuarterEnum Quarter number.

Quarter name. Return the name of the quarter (eg. first returns “Q1”, etc.)

String eraName(Integer year) eraName

year: Integer Year number.

Era of date. Returns the dating convention.

Eg. in English, 2037 will return “AD” or “CE”

String format() format

Date format. Returns the format to use when printing a date. The format is a string into which appropriate date components can be inserted. All characters in the string are interpreted as themselves, apart from a character preceded (escaped) by ”Escape sequences are summarized in table 3, and are based on the escape sequences provided in Unix when specifying dates.

Some examples are shown in table 4.

printDate(OutputStream stream, Date date, String format) printDate

stream: OutputStream The stream to print onto.

date: Date The date to print.

format: String The format (see format()) to use when printing. The default value is self.format().

Print a date in a suitable format. Print the date supplied onto the stream, using the result of format() to guide printing.

Date parseDate(InputStream stream, String format) parseDate

stream: InputStream The stream to parse.

format: String The format to parse the date against. The default value is self.format().

Raises: ParseException

Parse a date by loosely matching it against the format supplied in format.

Some formats cannot be used as a parsing format, due to the presence of alphanumeric characters that make parsing ambiguous. A ParseException should be raised in the case of a bad format.

Loose parsing is accepted:

Any whitespace or punctuation character in the format can match any sequence of zero or more whitespace and punctuation characters.

The Escape sequences supported during parsing are summarised in table 5.

Sequence	Description
%%	A single '%' character.
%a	The abbreviated weekday name, as returned by dayOfWeekName().
%A	The full weekday name, as returned by longDayOfWeekName().
%b	The abbreviated month name, as returned by monthName().
%B	The full month name, as returned by longMonthName().
%c	The century number.
%C	The century number, padded with a preceding 0 if necessary to make a two digit number.
%d	The day of month number.
%D	The day of month number, padded with a preceding 0 if necessary to make a two digit number.
%E	The day of month number, as returned by dayName().
%f	The day of year number.
%F	The day of the year number, as returned by dayOfYearName().
%m	The month number.
%M	The month number, padded with a preceding 0 if necessary to make a two digit number.
%q	The quarter number.
%Q	The quarter number, as returned by quarterName().
%w	The weekday number.
%y	The year number, as returned by yearName().
%Y	The full year number, as returned by longYearName().
%z	The year number within a century, padded with a preceding 0 if necessary to make a two digit number.
%Z	The era name, as returned by eraName().

Table 3: Date Format Escape Sequences

Date	%D-%b-%y	%D/%M/%z	%E %B %Y
23-Apr-1999	23-Apr-99	23/04/99	23rd April 1999
01-Jan-2005	01-Jan-05	01/01/05	1st January 2005
12-Jun-1830	12-Jun-1830	12/06/30	12th June 1830

Table 4: Example Date Formats

Sequence
%b
%B
%D
%E
%F
%M
%y
%Y

Table 5: parseDate Format Escape Sequences

The '%y' format interprets a two digit number as being the year with those last two digits closest to the current date (eg. in 1999, '45' is interpreted as '2045' and '78' is interpreted as '1978').

Date parseDate(String dateString, String format) parseDate
dateString: String
format: String
Raises: ParseException

Parses the date given in the dateString parameter. See the parseDate(inputStream, format) method for full documentation.

Date parseDateStrictly(InputStream stream, String format) parseDateStrictly
stream: InputStream The stream to parse.
format: String The format to parse the date against. The default value is self.format().
Raises: ParseException

Parse a date by matching it against the format supplied in format.

Apart from surrounding whitespace, and whitespace in the format matching 0 or more whitespace characters, the supplied stream must match the format exactly.

Some formats cannot be used as a parsing format, due to the presence of alphanumeric characters that make parsing ambiguous. A ParseException should be raised in the case of a bad format.

2.6 DatePosition

A specification of where a date should fall within some period. Example date positions are *the first Sunday in the month* or *the last business day of the year*.

2.6.1 Relationships

	Class	Description	Notes
↓	DatePositionModel §3.18		

↓:Realized by

2.6.2 Operations

Date force(Date date, DateClassifier classifier) force
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
Raises: ImmobileDateException

Force a date to a position within some span. Subclasses of DatePositionModel provide differing semantics for this operation. In general, however, the date is forced to a date that is the nth instance of something within a given periodUnit. Counting can be either forwards (from the start of the period) or backwards (from the end of the period).

If no such date exists, then an ImmobileDateException is raised.

Examples of date forcing would include:

(i) Force a date to the 3rd business day of the month. (ii) Force a date to the 7th tuesday of the year. (iii) Force a date to the 3rd-last weekday of the quarter.

Boolean testOn(Date date, DateClassifier classifier) testOn
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.default().
Raises: ImmobileDateException

Test a date to see whether it falls on the date that this datePosition specifies. Test date to see whether it is the same as the forced date:

$$date = force(date, classifier)$$

Boolean testBefore(Date date, DateClassifier classifier) testBefore
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.default().
Raises: ImmobileDateException

Test a date to see whether it falls before or on the date that this datePosition specifies. Test date to see whether it precedes, or is the same as the forced date:

$$date \leq force(date, classifier)$$

Boolean testAfter(Date date, DateClassifier classifier) testAfter
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.default().
Raises: ImmobileDateException

Test a date to see whether it falls on or after the date that this datePosition specifies. Test date to see whether it follows, or is the same as the forced date:

$$date \geq force(date, classifier)$$

2.7 DateRoller

Date rollers move a date to a suitable business date according to some convention.

Date rolling can be applied to a date even if that date falls on a business day.

2.7.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇓	DateRollerProgram §2.8		
↓	NullDateRollerModel §3.45		
↓	DateRollerReferenceDataModel §3.36		
↓	DateRollerFollowingModel §3.26		
↓	DateRollerPrecedingModel §3.30		
↓	DateRollerModifiedFollowing-Model §3.28		
↓	DateRollerModifiedPreceeding-Model §3.29		
↓	DateRollerForeignExchange-Model §3.27		
↔	DateRollerReferenceDataModel §3.36	model 0..1	
↔	PeriodWithRollerModel §3.47	roller 0..n	
↔	DateClassifierHolidayRelative-Model §3.66	roller 0..n	
↔	DateClassifierHolidayRegular-Model §3.65	position 0..n	
↔	RepeatedPeriodModel §3.57	roller 0..n	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.7.2 Operations

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll

classifier: DateClassifier The classifier for determining whether this date falls on an acceptable date.

date: Date The date to roll.

startDate: Date In the case of dates constructed by addinga period to another date, the original date before any addition occurred. This is relevant, for example, in the case of the Foreign Exchange rolling convention. The default value is date.

lastDate: Date In the case of dates constructed by adding a complex series of periods to a start date, last date refers to the date immediately prior to this date being constructed. The default value is startDate.

Raises: ImmobileDateException

Roll the supplied date.

The behavior of roll is essentially undefined. Generally, if a classifier classifies date as falling on a business day then date is returned, otherwise the date is moved in some manner to some date that classifier regards as a business day. This behavior is not necessary, however; a null date roller may simply return date whether it is on a business day or not.

An ImmobileDateException is raised if the date roller is unable to apply itself to that date. An example is a date roller which continually moves the date to an end of month position that is a holiday.

The roll method takes three date parameters: (i) 'date'. This is the date to be rolled (ii) If the date to be rolled has been determined by adding a period to another date, then we are sometimes interested in this other date, which is referred to as 'lastDate'. (iii) If the date to be rolled has been determined by adding a sequence of periods to another date, then we are sometimes interested in this original date, which is referred to as 'startDate'.

For example, in foreign exchange markets, periods are defined from spot date. Thus a 1-month period would produce a date 1 month from spot date. In this situation, the 1-month date would be 'date', the spot date would be 'lastDate', and today's date would be 'startDate'.

2.8 DateRollerProgram

A generalized date roller that uses a general program to compute a rolled date. DateRollerProgramModel allows date rollers to be defined using the scripting language in the 'Program' package, together with primitive date rolling tests and actions defined in this package.

Note that it is not necessary to define date rollers using this scripting language. Date rollers can be coded directly if this is preferred. The most common date rollers are defined separately in this package, without reference to the 'Program' package scripting language.

2.8.1 Relationships

	Class	Description	Notes
⇑	DateRoller §2.7		
↓	DateRollerProgramModel §3.35		

⇑:Inherits ↓:Realized by

2.8.2 Operations

Function program() program

The rolling program. Return the function that this roller uses to roll a supplied date.

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll

classifier: DateClassifier The classifier for determining whether this date falls on an acceptable date.
date: Date The date to roll.
startDate: Date In the case of dates constructed by adding to another date, the original date before any addition occurred. The default value is date.
lastDate: Date In the case of dates constructed by adding a complex series of elements to a start dates, last date refers to the date immediately prior to this date being constructed. The default value is startDate.
Raises: ImmobileDateException

Roll the supplied date. Evaluate the function returned by program() and return the result of the function. The associated function should expect 4 arguments: classifier, date, start date and last date. The result should be a date.

2.9 DayBasis

The day basis interface provides an interface to the conventions for calculating day counts between two dates. Day bases are essentially stateless and obey ValueSemantics.

2.9.1 Relationships

	Class	Description	Notes
⇑	ValueSemantics		
↓	DayBasisModel §3.37		

⇑:Inherits ↓:Realized by

2.9.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Print the conventional name for this day count convention onto the output stream. Day counts that include the last day when calculating day counts have a "+" appended to them.

Integer termInDays(Date from, Date to) termInDays

from: Date The start date of the calculation.

to: Date The end date of the calculation.

Elapsed time in days between two dates. Return the elapsed period in days between from and to. See the subclasses of DayBasisModel §3.37 for more detail on these calculations.

Boolean includeLastDay() includeLastDay

Include the last day in determining day counts?

Generally, the final day is not counted when calculating day counts — the elapsed number of days is counted. However, in certain cases, such as accrued interest calculations for Italian and Spanish bonds, the final day is included in the count.[1] This day simply adds an extra day to the day count, it does not shift the end date on an extra day.

2.10 Period

An amount of some time unit for date arithmetic purposes. Periods obey ValueSemantics, making them suitable for use as attributes.

2.10.1 Relationships

	Class	Description	Notes
⇑	ValueSemantics		
⇑	Identifiable		
⇓	PeriodWithRoller §2.11		
↓	PeriodModel §3.46		
↓	PeriodReferenceDataModel §3.48		
↔	PeriodReferenceDataModel §3.48	model 0..1	
↔	RepeatedPeriodEndPeriod-Model §3.61	endPeriod	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.10.2 Operations

PeriodUnit unit() unit

The unit of the period. Return the period unit that this period is expressed in. Examples of period units are days, weeks, months, etc...

Number amount() amount

The amount of period. Return the amount of period unit to add/subtract.

Period add(Period arg) add

arg: Period The amount to add to this period.
Raises: DateArithmeticException

Add two periods together. Both periods must have the same units; if the two units differ, then a DateArithmeticException is raised. The resulting Period, a has the property that, when applied to a date d, $d + a = d + this + arg$.

Period negated() negated

The negation of this amount. The resulting Period a, has the property that, when added to a period, d, the resulting period $r = d + a$ has $r + this = d$.

Period subtract(Period arg) subtract

arg: Period The amount to subtract from this period.
Raises: DateArithmeticException

Subtract one period from another. Both periods must have the same units; if the two units differ, then a DateArithmeticException is raised. The resulting Period, a has the property that, when applied to a period d, $d + a = d + this - arg$.

2.11 PeriodWithRoller

2.11.1 Relationships

	Class	Description	Notes
⇑	Period §2.10		
↓	PeriodWithRollerModel §3.47		

⇑:Inherits ↓:Realized by

2.11.2 Operations

DateRoller roller() roller

2.12 PeriodUnit

A unit of elapsed time, in the form of days, months, years, etc. PeriodUnits obey ValueSemantics, making them suitable for use as attributes.

2.12.1 Relationships

	Class	Description	Notes
⇑	ValueSemantics		
↓	PeriodUnitModel §3.49		
↔	DatePositionModel §3.18	within 1..1	
↔	RepeatedPeriodModel §3.57	units 0..n	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.12.2 Operations

Date addToDate(Date date, Number n, DateClassifier classifier) addToDate
date: Date The date to add to.
n: Number The amount of this unit to add to the date. This amount may be a non-integer.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is NullDateClassifierModel.defaultInstance().

Add n units of this unit to the date.

If n is a non-integer value, then the integer value is added to the date and the fractional value is then added as the rounded number of days that makes up that fraction for the period length. If adding that number of days would cause the date to cross this period's boundary, then the fractional part is allocated to each part of the period by calculating the number of days to take the date over the period, subtracting the ratio of those number of days and the period length from the fractional part and then adding the remaining fractional part to the new period.

Eg. Suppose we add 1.5 calendar months to 27-Sep-1999.

1. Add the integer number of months first to give 27-Oct-1999.

2. Adding (0.5×31) days to 27-Oct-1999 will take the date into November, so:

3. Add 5 days to 27-Oct-1999 to take it to 1-Nov-1999.

4. Add $(0.5 - (5/31)) \times 30 = 10.16 = 10$ days to 1-Nov-1999 to take it to 11-Nov-1999.

Date firstCalendarDay(Date date, DateClassifier classifier) firstCalendar-Day
date: Date The date to add to.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The first day of the unit that a date falls within. Return the first calendar day which falls within the period given by this unit and this date.

Eg. The first calendar day of the week for 1-Jan-1999 is 27-Dec-1998.

Date lastCalendarDay(Date date, DateClassifier classifier) lastCalendar-Day
date: Date The date to add to.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The last day of the unit that a date falls within. Return the last calendar day which falls within the period given by this unit and this date.

Eg. The last calendar day of the week for 1-Jan-1999 is 2-Jan-1999

Date firstNonWeekendDay(Date date, DateClassifier classifier) firstNonWeek-endDay
date: Date The date to add to.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The first non-weekend day of the unit that a date falls within. Return the first non-weekend day which falls within the period given by this unit and this date.

Eg. The first non-weekend day of the month for 8-May-1999 is 3-May-1999, given a Saturday/Sunday weekend.

Date lastNonWeekendDay(Date date, DateClassifier classifier) lastNonWeek-endDay
date: Date The date to add to.

classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The last non-weekend day of the unit that a date falls within. Return the last non-weekend day which falls within the period given by this unit and this date.

Eg. The last non-weekend day of the month for 8-May-1999 is 31-May-1999, given a Saturday/Sunday weekend.

Date firstBusinessDay(Date date, DateClassifier classifier) firstBusiness-Day
date: Date The date to add to.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The first business day of the unit that a date falls within. Return the first business day which falls within the period given by this unit and this date.

Eg. The first business day of the year for 8-May-1999 is 2-Jan-1999, given a Saturday/Sunday weekend and the 1st of January as a Holiday.

Date lastBusinessDay(Date date, DateClassifier classifier) lastBusiness-Day
date: Date The date to add to.
classifier: DateClassifier The classifier to use for determining whether the date is a business day, non-weekend day, etc. The default value is Null-DateClassifierModel.defaultInstance().

The last business day of the unit that a date falls within. Return the last business day which falls within the period given by this unit and this date.

Eg. The last business day of the year for 8-May-1999 is 31-Dec-1999, given a Saturday/Sunday weekend and the 1st of January as a Holiday.

Date startDate(Date date) startDate
date: Date

What is the start date of the period unit in which the parameter date falls?

Date endDate(Date date) endDate
date: Date

What is the end date of the period unit in which the parameter date falls?

Boolean testDatesInSamePeriod(Date testDate, Date baseDate) testDatesIn-SamePeriod
testDate: Date
baseDate: Date

Do the two parameter dates fall in the same period unit? e.g. Are two dates in the same month?

If the start date of the period unit for the testDate is on or before the baseDate, and the end date of this periodUnit is on or after baseDate, then return true. Otherwise return false.

2.13 RepeatedPeriod

A repeated period represents the repeated application of a period to a date to provide either a single end date or a series of dates.

Repeated periods can be *chained*. In a chained repeated period, each date is generated by adding the period once to the $previous$ date and then rolling the result.

In an unchained repeated period, the nth date in the sequence of dates generated by the repeated period is calculated by adding the period n times and then rolling the resulting date.

Using the notation $d + n \cdot u$ for the application of n units of period unit u to date d and $d_0 = date$, then:

If the repeated period is chained, then

$$d_i = d_{i-1} + sizeOfRollPeriod \cdot unit$$

If the repeated period is not chained, then

$$d_i = d_0 + (sizeOfRollPeriod \times i) \cdot unit$$

For example, a chained roller:
period = 1 Month
sizeOfRollPeriod = 2
numberOfRollPeriods = 3

$$\begin{aligned} d_0 &= StartDate \\ d_1 &= StartDate + (1M \times 2) \;\; roll \\ d_2 &= d_1 + (1M \times 2) \;\; roll \\ d_3 &= d_2 + (1M \times 2) \;\; roll \end{aligned}$$

Unchained:

$$
\begin{aligned}
d_0 &= StartDate \\
d_1 &= StartDate + (1 \times (1M \times 2)) \quad roll \\
d_2 &= StartDate + (2 \times (1M \times 2)) \quad roll \\
d_3 &= StartDate + (3 \times (1M \times 2)) \quad roll
\end{aligned}
$$

2.13.1 Relationships

	Class	Description	Notes
↓	RepeatedPeriodModel §3.57		
↓	RepeatedPeriodReferenceData-Model §3.62		
↔	RepeatedPeriodReferenceData-Model §3.62	model 0..n	

↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.13.2 Operations

Date stopDate(Date date, DateClassifier classifier) stopDate
date: Date
classifier: DateClassifier
Raises: ImmobileDateException

Apply this period to the supplied date. In the case of a repeated period, this operation returns the last date in the sequence of dates that this repeated period generates.

Collection<Date> dates(Date date, DateClassifier classifier) dates
date: Date The date to start the repeated period at.
classifier: DateClassifier The date classifier for date rolling.
Raises: ImmobileDateException

Returns a sorted collection of dates that represents the application of this repeated period to an initial date. Implementations of this interface provide the behaviour for this operation.

Boolean isChained() isChained

Is this a chained repeated period? Return true if the dates generated by this repeated period are to be chained, false otherwise.

2.14 YearBasis

A year basis calculates the number of years in some elapsed time, based on the year convention. Year bases are essentially stateless and implement ValueSemantics.

2.14.1 Relationships

	Class	Description	Notes
⇑	ValueSemantics		
↓	YearBasisModel §3.67		

⇑:Inherits ↓:Realized by

2.14.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Print the conventional name for this year count convention onto the output stream.

Double termInYears(Date from, Date to, DayBasis basis) termInYears

from: Date The start date of the calculation.

to: Date The end date of the calculation.

basis: DayBasis The day count convention to use.

Elapsed time in years between two dates. Return the elapsed period in years between from and to. This calculation may be fairly complex, due to the need to handle variable length years. See the subclasses of YearBasisModel §3.67 for more detail on these calculations.

3 Classes

3.1 DateBasisModel

A concrete implementation of the DateBasis interface.

3.1.1 Relationships

	Class	Description	Notes
↑	DateBasis §2.2		

↑:Realizes

3.1.2 Attributes

dayBasis: DayBasis The day count convention.

yearBasis: YearBasis The year length convention.

3.2 DateClassifierReferenceDataModel

DateClassifierReferenceDataModel allows date classifiers to be manipulated in the form of reference data. The actual data classifier model is held in an associated DateClassifier.

Note that the model may be DateClassifierCompositeModel §3.4 or DateClassifierUnionModel §3.3, allowing grouping of related holidays.

3.2.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	DateClassifier §2.3		
↔	DateClassifier §2.3	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.3 DateClassifierUnionModel

The union of two date classifiers merges the holidays and weekends for two locations, disregarding precedence considerations. Thus a date classifier union is built by combining a series of holidays and weekends, with the resulting classifier representing the union of its components.

This kind of composite date classifier can be used to represent the combined holidays of two different locations, in which case the relative holidays in each location do not affect each other.

3.3.1 Relationships

	Class	Description	Notes
↑	DateClassifier §2.3		
⇓	DateClassifierCompositeModel §3.4		
↔	DateClassifier §2.3		→
↔	DateClassifier §2.3	components 0..n	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.3.2 Attributes

identifier: String The unique identifier for this object, if one is needed. If this object is an anonymous combination of holidays, this attribute may be nil.

An example of an anonymous combination of holidays would be in the situation where an FX deal between USD and FRF is done between parties in Melbourne and Havana. You don't want to name all of the holiday combinations that result.

3.3.3 Operations

Boolean isWeekend(Date date, DateClassifier environment) isWeekend
date: Date The date to test.
environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a weekend? Return true if there exists a component date classifier that returns true for isWeekend with the supplied date and environment, false otherwise.

Boolean isHoliday(Date date, DateClassifier environment) isHoliday
date: Date The date to test.
environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a holiday? Return true if there exists a component date classifier that returns true for isHoliday with the supplied date and environment, false otherwise.

Boolean isSimple() isSimple

This classifier is simple? Return false since this classifier is made up of more than one classifier.

DateClassifier compose(DateClassifier arg) compose
arg: DateClassifier The date classifier to compose this classifier with.

Compose two date classifiers so that the two classifiers are affected by precedence considerations. Return a DateClassifierCompositeModel §3.4 with this classifier and the argument classifier as components.

Boolean precedes(DateClassifier other) precedes

other: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. Return false.

DateClassifier predecessorsOf(DateClassifier test) predecessorsOf

test: DateClassifier The date classifier to test against.

The predecessors of a date classifier.

Return $\bigcup_{c \in components} c.predecessorsOf(test)$ where $\bigcup$ represents the union operation.

DateClassifier union(DateClassifier arg) union

arg: DateClassifier The date classifier to combine this classifier with.

Join two date classifiers so that the classifiers are unaffected by precedence considerations. Return a DateClassifierUnionModel §3.3 with the union of the components of this classifier and the arg classifier as components.

«Static Method» DateClassifier local() local

Local date classifier. Return the date classifier which encodes the holidays and weekends of the location running the system.

3.4 DateClassifierCompositeModel

A date classifier built by combining a series of holidays and weekends. The resulting classifier represents the composite of its components. Composite date classifiers allow the composite to affect the placement of other elements within the date classifier so as to reflect precedence relationships.

This kind of composite date classifier can be used to represent the combined holidays of a single location.

3.4.1 Relationships

	Class	Description	Notes
⇑	DateClassifierUnionModel §3.3		

⇑:Inherits

3.4.2 Operations

Boolean isWeekend(Date date, DateClassifier environment) isWeekend

date: Date The date to test.

environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a weekend? Let env be the composite of the environment and this date classifier. Return true if there exists a component date classifier that returns true for isWeekend with the supplied date and env, false otherwise.

Boolean isHoliday(Date date, DateClassifier environment) isHoliday
date: Date The date to test.
environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifierModel.defaultInstance().

Date falls on a holiday? Let env be the composite of the environment and this date classifier. Return true if there exists a component date classifier that returns true for isHoliday with the supplied date and env, false otherwise.

DateClassifier compose(DateClassifier arg) compose
arg: DateClassifier The date classifier to compose this classifier with.

Compose two date classifiers so that the two classifiers are affected by precedence considerations. Return a DateClassifierCompositeModel §3.4 with the union of the components of this classifier and the arg classifier as components.

3.5 DateClassifierWeekendModel

A basic implementation of the most common form of fixed weekend. A number of days during the week are set aside for the weekend.

This form of weekend takes precedence over any other holiday model.

3.5.1 Relationships

	Class	Description	Notes
↑	DateClassifier §2.3		

↑:Realizes

3.5.2 Attributes

weekendDays: Set<DayOfWeekEnum> The days of the week that constitute this weekend.

identifier: String The identifier for this weekend type (may be nil if this is an anonymous weekend classifier).

3.5.3 Operations

Boolean isHoliday(Date date, DateClassifier environment) isHoliday
date: Date
environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifier.defaultInstance().

Test a date for being a holiday. Always returns false.

Boolean isWeekend(Date date, DateClassifier environment) isWeekend
date: Date
environment: DateClassifier The surrounding date classifier context. The default value is NulDateClassifier.defaultInstance().

Test a date for being a weekend day. Return true if weekendDays includes the result returned by date.dayOfWeek().

Boolean isBasic() isBasic

This classifier is basic? Return true.

DateClassifier compose(DateClassifier arg) compose
arg: DateClassifier The date classifier to compose this classifier with.

Compose two date classifiers so that the two classifiers are affected by precedence considerations. Return a DateClassifierCompositeModel §3.4 with this classifier and the argument classifier as components.

Boolean precedes(DateClassifier other) precedes
other: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. Return true if the argument classifier is a DateClassifierWeekendModel, false otherwise.

DateClassifier predecessorsOf(DateClassifier test) predecessorsOf
test: DateClassifier The date classifier to test against.

The predecessors of a date classifier. If the test argument is a DateClassifierWeekendModel, return an instance of the NullDateClassifier, otherwise return this date classifier.

DateClassifier union(DateClassifier arg) union
arg: DateClassifier The date classifier to combine this classifier with.

Join two date classifiers so that the classifiers are unaffected by precedence considerations. Return a DateClassifierUnionModel §3.3 with this classifier and the argument classifier as components.

«Static Method» DateClassifier local() local

Local date classifier. Return the date classifier which encodes the holidays and weekends of the location running the system.

3.6 DateConditionPrimitiveModel

An abstract superclass for the various date rolling tests.

3.6.1 Relationships

	Class	Description	Notes
↑	SystemFunction		
⇓	DateConditionDayTypeModel §3.8		
⇓	DateConditionPositionModel §3.11		
⇓	DateConditionCrossModel §3.7		

⇓:Inherited by ↑:Realizes

3.7 DateConditionCrossModel

Test to see whether a date rolling operation has crossed some boundary, such as a month boundary.

3.7.1 Relationships

	Class	Description	Notes
⇑	DateConditionPrimitiveModel §3.6		

⇑:Inherits

3.7.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith

passedParameters: Collection<Object> The function parameters.

Execute this function. Three parameters are passed into the operation: a Date §2.1 called the test date, a Date called the base date and a PeriodUnit §2.12.

Pass these parameters to the 'testCrossingPeriodUnit' method and return the result.

String identifier() identifier

The unique identifier for the function. Return "dateCrosses"

«Static Method» Boolean testCrossingPeriodUnit(Date date, Date baseDate, PeriodUnit periodUnit) testCrossingPeriodUnit

date: Date

baseDate: Date

periodUnit: PeriodUnit

Test whether the testDate and the baseDate fall in different calendar period units. This is done by sending the 'testDatesInSamePeriod' method to the PeriodUnit with testDate and baseDate as parameters.

For example, testing 23-Jan-1999 against 29-Jan-1999 and a month periodUnit returns false, as both dates fall within the same month. Testing the same dates against a week period returns true, as the dates fall into different weeks.

3.8 DateConditionDayTypeModel

An abstract superclass for the various day-type tests.

3.8.1 Relationships

	Class	Description	Notes
⇑	DateConditionPrimitiveModel §3.6		
⇓	DateConditionNonBusinessDayModel §3.9		
⇓	DateConditionWeekendDayModel §3.10		

⇑:Inherits ⇓:Inherited by

3.9 DateConditionNonBusinessDayModel

Test to see if this day is a non-business day.

Note that this test is the negation of the usual business day tests. In most cases, rolling will occur on a non-business day, rather than a business day and, rather than force users to constantly use a negation operation, the negation operation has been built in.

3.9.1 Relationships

	Class	Description	Notes
⇑	DateConditionDayTypeModel §3.8		

⇑:Inherits

3.9.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Two parameters are passed into the operation: a Date §2.1 and a DateClassifier §2.3.

Pass these parameters to the 'testNonBusinessDay' method, and return the result.

String identifier() identifier

The unique identifier for the function. Return "isNonBusinessDay"

«Static Method» Boolean testNonBusinessDay(Date date, DateClassifier classifier) testNonBusinessDay
date: Date
classifier: DateClassifier

If the date classifier classifies the date as a non-business day, then return true, otherwise return false.

3.10 DateConditionWeekendDayModel

Test to see if this day is a weekend day.

Note that this test is the negation of the usual non-weekend tests. In most cases, rolling will occur on a weekend day, rather than a non-weekend day and, rather than force users to constantly use a negation operation, the negation operation has been built in.

3.10.1 Relationships

	Class	Description	Notes
⇑	DateConditionDayTypeModel §3.8		

⇑:Inherits

3.10.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Two parameters are passed into the operation: a Date §2.1 and a DateClassifier §2.3.

Pass these parameters to the 'testWeekendDay' method and return the result.

String identifier() identifier

The unique identifier for the function. Return "isWeekendDay"

«Static Method» Boolean testWeekendDay(Date date, DateClassifier classifier) testWeekend-Day
date: Date
classifier: DateClassifier

If the date classifier classifies the date as a weekend day, then return true, otherwise return false.

3.11 DateConditionPositionModel

Test to see whether a date satisfies some position requirement; whether the date is before, on or after some reference date.

3.11.1 Relationships

	Class	Description	Notes
⇑	DateConditionPrimitiveModel §3.6		
⇓	DateConditionOnModel §3.14		
⇓	DateConditionBeforeModel §3.13		
⇓	DateConditionAfterModel §3.12		

⇑:Inherits ⇓:Inherited by

3.12 DateConditionAfterModel

Test to see whether a date is on or after a reference date.

3.12.1 Relationships

	Class	Description	Notes
⇑	DateConditionPositionModel §3.11		

⇑:Inherits

3.12.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Three parameters are passed into the operation: a Date §2.1, a DatePosition §2.6 and a a DateClassifier §2.3.

Pass these parameters to the 'testDateAfterPosition' method, and return the result.

String identifier() identifier

The unique identifier for the function. Return "isDateBefore"

«Static Method» Boolean testDateAfterPosition(DatePosition datePosition, Date date, DateClassifier classifier) testDateAfterPosition
datePosition: DatePosition
date: Date
classifier: DateClassifier

Send the testAfter message to the datePosition and return the result.

3.13 DateConditionBeforeModel

Test to see whether a date is on or before a reference date.

3.13.1 Relationships

	Class	Description	Notes
⇑	DateConditionPositionModel §3.11		

⇑:Inherits

3.13.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Three parameters are passed into the operation: a Date §2.1, a DatePosition §2.6 and a a DateClassifier §2.3.

Pass these parameters to the 'testDateBeforePosition' method, and return the result.

String identifier() identifier

The unique identifier for the function. Return "isDateBefore"

«Static Method» Boolean testDateBeforePosition(DatePosition datePosition, Date date, DateClassifier classifier) testDateBeforePosition
datePosition: DatePosition
date: Date
classifier: DateClassifier

Send the testOn message to the datePosition and return the result.

3.14 DateConditionOnModel

Test to see if a date is the same as a reference date.

3.14.1 Relationships

	Class	Description	Notes
⇑	DateConditionPositionModel §3.11		

⇑:Inherits

3.14.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Three parameters are passed into the operation: a Date §2.1, a DatePosition §2.6 and a a DateClassifier §2.3.

Pass these parameters to the 'testDateOnPosition' method, and return the result.

String identifier() identifier

The unique identifier for the function. Return "isDateOn"

«Static Method» Boolean testDateOnPosition(DatePosition datePosition, Date date, DateClassifier classifier) testDateOnPosition

datePosition: DatePosition
date: Date
classifier: DateClassifier
Send the testOn message to the datePosition and return the result.

3.15 DateFormatModel

The DateFormatModel class provides a concretization of the DateFormat interface. Dictionaries of names and suffixes are used to provide a language and location dependent format.

The operations specified in DateFormat are implemented by appropriate lookups on the supplied dictionaries.

If more language modeling is done in future releases, then some parts of this model will probably be moved to the language model.

3.15.1 Relationships

	Class	Description	Notes
↑	DateFormat §2.5		
↑	Validatable		

↑:Realizes

3.15.2 Attributes

weekdaysShort: Dictionary<DayOfWeekEnum, String> Holds a dictionary which maps a day of the week onto an abbreviated weekday name.

weekdaysLong: Dictionary<DayOfWeekEnum, String> Holds a dictionary which maps a day of the week onto a full weekday name.

monthsShort: Dictionary<MonthEnum, String> Holds a dictionary which maps a month onto an abbreviated month name.

monthsLong: Dictionary<MonthEnum, String> Holds a dictionary which maps a month onto a full month name.

eras: Dictionary<EraEnum, String> Holds a dictionary which maps the elements of EraEnum §5.3 onto appropriate era strings (eg. before maps onto "BC").

quartersShort: Dictionary<QuarterEnum, String> Holds a dictionary which maps a quarter onto an abbreviated quarter name.

quartersLong: Dictionary<QuarterEnum, String> Holds a dictionary which maps a quarter onto a full quarter name.

suffixes: Dictionary<Integer, String> Holds a partial dictionary that maps integers in the range 0–99 onto strings. Suffixes are added to numbers to indicate placement (eg. 6 has 'th' added to become '6th', 1 has 'st' added to become '1st'). In general, the default suffix is used; this dictionary contains exceptional cases.

defaultSuffix: String Holds the suffix to use if a number is not found inside the suffix dictionary.

zeroFlag: Boolean If true, this flag indicates that this date format counts the year 0 as 0, rather than as 1 from the previous era. (eg. in the common era calendar, year 0 is set to be 1 BCE).

identifier: String The unique identifier for the date format.

format: String This stores the format to use when printing a date. For example, if the 23rd of april 1999 is to be prined as 23/04/99, then the format would be "

3.15.3 Operations

Reportable validate() validate

Validate this piece of data.
A DateFormatModel is valid if the following conditions hold:

- weekdaysShort and weekdaysLong have a mapping for all elements of DayOfWeekEnum §5.2. Two different key values may not map onto the same string.
- monthsShort and monthsLong have a mapping for all elements of MonthEnum §5.4. Two different key values may not map onto the same string.
- Eras must have a mapping for all elements of EraEnum §5.3. Two different key values may not map onto the same string.
- quartersShort and quartersLong have a mapping for all elements of QuarterEnum §5.5. Two different key values may not map onto the same string.
- defaultSuffix is a non-empty string.
- Suffixes should only have keys in the range 0–99.

Additionally, if format has an alphanumeric character in it that is not part of a "%" escape sequence, then a warning should be added, indicating that the format may not be parsable.

«Static Method» DateFormat local() local

Local common date format. Return the date format that encodes the common local convention used when printing dates.

«Static Method» DateFormat localShort() localShort

Local common short date format. Return the date format that encodes the common local convention used when printing dates in an abbreviated form.

«Static Method» DateFormat localLong() localLong

Local common long date format. Return the date format that encodes the common local convention used when printing dates in a long form.

«Static Method» DateFormat localVeryLong() localVeryLong

Local common very long date format. Return the date format that encodes the common local convention used when printing dates in a very long form.

3.16 DateFormatReferenceDataModel

The DateFormatReferenceDataModel provides a means of storing and using date formats as reference data. Essentially, this class acts as a wrapper around an actual DateFormatModel, although it responds to the same methods and can be used in place of a DateFormatModel instance.

3.16.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	DateFormat §2.5		
↔	DateFormat §2.5	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.16.2 Operations

Reportable validate() validate

Validate this piece of data. Compose the normal reference data validation with the results of validating the associated DateFormatModel.

3.17 DateModel

The Date Model class implements the Date class by placing the count of days into a year and the year number.

The range of this class is dependent on the implementation of Integer within the target language. In most cases, implementation limits will not be a problem, as even a 16 bit year will extend to about 32,000 CE.

3.17.1 Relationships

	Class	Description	Notes
↑	Date §2.1		

↑:Realizes

3.17.2 Attributes

dayOfYear: Integer The day count into the year, with the 1st of January being day 1.

year: Integer The year number in the CE calendar.

3.18 DatePositionModel

A concrete implementation of the DatePosition interface. Subclasses encode the type of position that we wish to force a date to.

The start and end dates of the period within which the date is to be forced are supplied, relative to the reference date, by the associated PeriodUnit §2.12.

3.18.1 Relationships

	Class	Description	Notes
↑	DatePosition §2.6		
⇓	DatePositionCalendarDayModel §3.20		
⇓	DatePositionNonWeekend-Model §3.23		
⇓	DatePositionBusinessDayModel §3.19		
⇓	DatePositionWeekModel §3.25		
⇓	DatePositionMonthModel §3.22		
⇓	DatePositionDayOfWeekModel §3.21		
⇓	DatePositionQuarterModel §3.24		
↔	PeriodUnit §2.12	within 0..*	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.18.2 Attributes

direction: DateDirectionEnum = forwards If set to forwards, the position is found by moving forward from the start of the associated period. If set to backwards, the position is found by moving backward from the end of the period.

nth: Integer Move to the nth date that satisfies the criteria set out in the subclass. Counting starts with 1 being the first/last date that satisfies the subclass.

3.19 DatePositionBusinessDayModel

A position in terms of business days. Eg. the 3rd business day in the month.

3.19.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.19.2 Operations

Date force(Date date, DateClassifier classifier) force
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
Raises: ImmobileDateException

Force a date to a position within a date unit. Move forwards/backwards to the nth business day, as specified by classifier, from the start/end of the associated periodUnit.

3.20 DatePositionCalendarDayModel

A position in terms of calendar days. Eg. The 25th calendar day in the year.

3.20.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.20.2 Operations

Date force(Date date, DateClassifier classifier) force

date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
Raises: ImmobileDateException

Force a date to a position within a date unit. Move forwards/backwards to the nth calendar day from the start/end of the associated date amount.

3.21 DatePositionDayOfWeekModel

Force a date to the nth occurrence of particular day of the week within the chosen period. For example, the 4th Monday in the month, the 5th Sunday in the year.

3.21.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.21.2 Attributes

day: DayOfWeekEnum The day of the week.

3.21.3 Operations

Date force(Date date, DateClassifier classifier, Integer targetOrder) force
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
targetOrder: Integer The order of date classifier to use. The default value is MaxInteger.
Raises: ImmobileDateException

Force a date to a position within a date unit. Move forwards/backwards to the nth date which corresponds to the same day of the week from the start/end of the associated date unit.

For example, the 2nd Saturday of the month would have nth = 1, direction = forwards, day = saturday and a date amount of 1 month. Applying this position to 23-Jul-1998 would force the date to 11-Jul-1998.

3.22 DatePositionMonthModel

A date position in terms of months. Eg. the 3rd month before the end of the year.

3.22.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.22.2 Operations

Date force(Date date, DateClassifier classifier) force
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
Raises: ImmobileDateException

Force a date to a position within a period unit. Move forwards/backwards to the nth month from the start/end of the associated date unit. The day of the month remains the same.

If the day of the month would cause an illegal date (eg. forcing the 30th of June to February) then the day of month is adjusted to be the last day of the month.

3.23 DatePositionNonWeekendModel

A position in terms of non-weekend days. Eg. The 2nd non-weekend day in the week.

3.23.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.23.2 Operations

Date force(Date date, DateClassifier classifier) force
date: Date The date to force.
classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().
Raises: ImmobileDateException

Force a date to a position within a date unit. Move forwards/backwards to the nth non-weekend day, as specified by classifier, from the start/end of the associated date unit.

3.24 DatePositionQuarterModel

A date position in terms of a number of calendar quarters. Eg: the third quarter in a year.

3.24.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.24.2 Attributes

quarter: QuarterEnum

3.24.3 Operations

Date force(Date date, DateClassifier classifier) force
date: Date

classifier: DateClassifier The default value is NullDateClassifierModel.defaultInstance().

Force a date to a position within a period unit. Move forwards/backwards to the nth quarter from the start/end of the associated period unit.

Force a date to a position within a period unit. Move forwards/backwards to the nth quarter from the start/end of the associated date unit. The day of the quarter remains the same.

The first day of the quarter is the first day of the first month of the quarter. The last day of the quarter is the last day of the last month of the quarter.

If the day of the quarter would cause an illegal date then the day of month is adjusted to be the last day of the quarter. For example, forcing 31 December (the 92nd day of the october quarter) to the january quarter would give 31 March (the 90th day of the January quarter), since this is the last day of the quarter.

3.25 DatePositionWeekModel

A position in terms of weeks. Eg. the 3rd week of the year.

3.25.1 Relationships

	Class	Description	Notes
⇑	DatePositionModel §3.18		

⇑:Inherits

3.25.2 Operations

Date force(Date date, DateClassifier classifier) force

date: Date The date to force.

classifier: DateClassifier The date classifier for business/non-weekend day classification, if needed. The default value is NullDateClassifierModel.defaultInstance().

Raises: ImmobileDateException

Force a date to a position within a date unit. Move forwards/backwards to the nth week from the start/end of the associated date unit. The relative position within the week (Sunday, Monday, etc.) remains the same as that of the supplied date.

For a month, the first week of the month is the week into which the first day of the month falls and the last week of the month is the week before the first week of the next month.

For a quarter, the first week of the quarter is the first week of the first month of the quarter and the last week is the last week of the last month of the quarter..

For a year, the first week of the year is the week into which the 4th of January falls.[2] The last week of the year is the week before the first week of the next year.

3.26 DateRollerFollowingModel

Under the 'following' date rolling convention, if the day occurs on a non-business day, the day is rolled to the next business day.

For example, 4 August 1999 + 1 month = 4 September 1999 (a Saturday), then roll forward to 6 September 1999.

3.26.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.26.2 Operations

String identifier() identifier

Return "Following".

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll

classifier: DateClassifier
date: Date
startDate: Date
lastDate: Date

test whether the dateClassifier classifies date as a non business day. If it does, then add a period of 1 calendar day to date. Repeat the process until we get a business day.

3.27 DateRollerForeignExchangeModel

Under the 'foreign exchange' date rolling convention, if the day occurs on a non-business day, the day is rolled to the next business day. No adjustment in the

forward value date is made for any weekends or public holiday between the spot date and the forward delivery date. An exception to this rule is when the spot value is the last working day of the month. Then the forward value date is the last working day of the corresponding forward month. However, if necessary, the forward value date is brought back to the nearest previous business day in order to stay in the same calendar month, rather than moved forward to the beginning of next month.

For example, Dealing date: 26 June 1996 Spot date: 28 June 1996 (last working day of June) 1 month: 31 July 1996 2 months: 30 August 1996

Additionally, even if the spot value date is earlier than the last working day of the month, but the forward value date would fall on a non-business day, this is still brought back rather than moved later, if necessary to keep it in the appropriate month.

3.27.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.27.2 Operations

String identifier() identifier

Return "ForeignExchange".

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll
classifier: DateClassifier
date: Date
startDate: Date
lastDate: Date

test whether lastDate (the spot date) is the last business day of the month. This is done by creating a DatePositionBusinessDayModel with direction = 'backwards' and nth = 1, then sending the testOn() method with 'lastDate' as parameter.

If lastDate is not the last business day of the month then use the 'modifiedFollowing' rollowing convention. i.e. roll the date forward to the next business date, unless this crosses a month boundary, in which case roll the date backwards to the previous business date.

If lastDate is the last business day of the month then the required date is the last business day of the month in which 'date' falls. This is done by creating a DatePositionBusinessDayModel with direction = 'backwards' and nth = 1, then sending the force() method with 'date' as parameter.

3.28 DateRollerModifiedFollowingModel

Under the 'modified following' date rolling convention, if the day occurs on a non-business day, the day is rolled to the next business day unless the next business day crosses a month boundary, in which case the day is rolled back to the previous business day.

For example, 30 April 1998 + 1 month = 30 May 1998 (a Saturday), then roll forward to 1 June 1998. This crosses a boundary month so the date will roll back to 29 May 1998

3.28.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.28.2 Operations

String identifier() identifier

Return "ModifiedFollowing"

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll
classifier: DateClassifier
date: Date
startDate: Date
lastDate: Date

test whether the dateClassifier classifies date as a non business day. If it does, then add a period of 1 calendar day to date. Repeat the process until we get a business day.

Test whether this resulting business day falls in the same month as the original date. This is done by sending 'testDatesInSamePeriod' to a MonthPeriodUnit-

Model with the two dates as parameters. If this is false, then add a period of -1 day to 'date' until a business day is reached.

3.29 DateRollerModifiedPreceedingModel

Under the 'modified preceding' date rolling convention, if the day occurs on a non-business day, the day is rolled to the previous business day unless the previous business day crosses a month boundary, in which case the day is rolled forward to the next business day.

For example, 1 December 1998 + 1 month = 1 January 1999 (a public holiday), then roll back to 31 December 1998. This crosses a boundary month so the date will roll forward to 4 January 1999.

3.29.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.29.2 Operations

String identifier() identifier

Return "ModifiedPreceding".

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll
classifier: DateClassifier
date: Date
startDate: Date
lastDate: Date

test whether the dateClassifier classifies date as a non business day. If it does, then add a period of -1 calendar day to date. Repeat the process until we get a business day.

Test whether this resulting business day falls in the same month as the original date. This is done by sending 'testDatesInSamePeriod' to a MonthPeriodUnitModel with the two dates as parameters. If this is false, then add a period of 1 day to 'date' until a business day is reached.

3.30 DateRollerPrecedingModel

Under the 'preceding' date rolling convention, if the day occurs on a non-business day, the day is rolled to the previous business day.

For example, 4 August 1999 + 1 month = 4 September 1999 (a Saturday), then roll backward to 3 September 1999.

3.30.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.30.2 Operations

String identifier() identifier

Return "Preceding".

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll
classifier: DateClassifier
date: Date
startDate: Date
lastDate: Date

test whether the dateClassifier classifies date as a non business day. If it does, then add a period of -1 calendar day to date. Repeat the process until we get a business day.

3.31 DateRollerPrimitiveModel

An abstract superclass of the various primitive date rolling functions.

3.31.1 Relationships

	Class	Description	Notes
↑	SystemFunction		
⇓	DateRollerAdditionModel §3.32		
⇓	DateRollerForcingModel §3.34		
⇓	DateRollerCallingModel §3.33		

⇓:Inherited by ↑:Realizes

3.32 DateRollerAdditionModel

Add (or subtract) to a date.

3.32.1 Relationships

	Class	Description	Notes
⇑	DateRollerPrimitiveModel §3.31		

⇑:Inherits

3.32.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Two parameters are passed into the operation: a Date §2.1 and a Period §2.10.

Pass these parameters to the addPeriodToDate method, and return the result.

String identifier() identifier

The unique identifier for the function. Return “dateRollerAdd”

«Static Method» Date addPeriodToDate(Date date, Period period) addPeriodTo-Date
date: Date
period: Period

Return the result of adding the period parameter to the date parameter.

3.33 DateRollerCallingModel

Roll a date by calling another date roller.

3.33.1 Relationships

	Class	Description	Notes
⇑	DateRollerPrimitiveModel §3.31		

⇑:Inherits

3.33.2 Operations

Object executeWithOn(Collection<Object> passedParameters, Object targetObject) executeWithOn
passedParameters: Collection<Object> The function parameters.
targetObject: Object

Execute this operation. Four parameters are passed into the operation: a DateClassifier §2.3 Date §2.1 called the date, a Date called the last date and

The target object must be a DateRoller §2.7.

Pass these parameters to the 'callDateRoller' static method on the receiver, and return the result.

String identifier() identifier

The unique identifier for the function. Return "dateRollerCall"

«Static Method» Date callDateRoller(DateRoller targetRoller, DateClassifier classifier, Date date, Date lastDate) callDateRoller
targetRoller: DateRoller
classifier: DateClassifier
date: Date
lastDate: Date

If the start date is absent, it defaults to the last date. If the last date is absent, it defaults to the date.

The date, lastDate and classifier parameters are passed to the roll() operation on the targetRoller object and the result returned.

3.34 DateRollerForcingModel

Force a date to a suitable position. This function can be used to force dates to, eg. the end of the quarter, or the 15th of the month.

3.34.1 Relationships

	Class	Description	Notes
⇑	DateRollerPrimitiveModel §3.31		

⇑:Inherits

3.34.2 Operations

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object> The function parameters.

Execute this function. Three parameters are passed into the operation: a Date §2.1, a DatePosition §2.6 and a DateClassifier §2.3. Pass these parameters to the 'forceDateToPosition' method and return the result.

String identifier() identifier

The unique identifier for the function. Return "dateRollerForce"

«Static Method» Date forceDateToPosition(Date date, DatePosition position, DateClassifier classifier) forceDateToPosition
date: Date
position: DatePosition
classifier: DateClassifier

Three parameters are passed into the operation: a Date §2.1, a DatePosition §2.6 and a DateClassifier §2.3. Return the result of applying the date position to the date.

3.35 DateRollerProgramModel

A date roller that allows access to the generic programming structures from the Program package. The date roller can then use these structures to build arbitrarily complex date rolling specifications. The date to roll and the resulting rolled date are passed to the program via a variable.

3.35.1 Relationships

	Class	Description	Notes
↑	DateRollerProgram §2.8		
↔	Function	program 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.35.2 Attributes

identifier: String The unique identifier for the date roller. This attribute may be nil if this is an anonymous date roller.

3.35.3 Operations

Function program() program

The program to perform the date roll. Return the associated program.

3.36 DateRollerReferenceDataModel

Date rollers tend to be applied in a large number of areas. This class provides a wrapper so that common date rollers can be managed as reference data. The reference data object implements the DateRoller interface by delegating to the held DateRoller instance.

Generally, only program-based date rollers need to be managed as reference data.

3.36.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	DateRoller §2.7		
↔	DateRoller §2.7	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.37 DayBasisModel

The DayBasisModel provides an abstract superclass for the various day count conventions.

3.37.1 Relationships

	Class	Description	Notes
↑	DayBasis §2.9		
⇓	DayBasisActual §3.42		
⇓	DayBasis30Abstract §3.38		
⇓	DayBasisNL §3.43		

⇓:Inherited by ↑:Realizes

3.37.2 Attributes

includeLastDay: Boolean = false Include the final day in the day count?

3.38 DayBasis30Abstract

The 30 day date basis calculates day counts based on all months having an even length of 30 days, with a year, therefore, having 360 days. The last days of the month tend to be treated as a special case, with several different conventions.

3.38.1 Relationships

	Class	Description	Notes
⇑	DayBasisModel §3.37		
⇓	DayBasis30E §3.40		
⇓	DayBasis30 §3.39		
⇓	DayBasis30PSA §3.41		

⇑:Inherits ⇓:Inherited by

3.39 DayBasis30

The Day 30 Model reflects the International Swap Dealers Association (ISDA) convention; the most common form of 30 day date basis.

3.39.1 Relationships

	Class	Description	Notes
⇑	DayBasis30Abstract §3.38		

⇑:Inherits

3.39.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Add “30” to the output stream. If includeLastDay is true, add a “+” to the output stream.

Integer termInDays(Date from, Date to) termInDays

from: Date The start date of the calculation.

to: Date The end date of the calculation.

Elapsed time in days between two dates.

If d_1, m_1 and y_1 are, respectively, the day, month and year numbers for the from date and d_2, m_2 and y_2 are, respectively, the day, month and year numbers for the to date, then the term in days is given by:

$$360 \times (y_2 - y_1) + 30 \times (m_2 - m_1) + (z_2 - z_1)$$

where z_1 and z_2 are defined by the following table:

d_1	d_2	z_1	z_2
< 30		d_1	d_2
≥ 30	< 30	30	d_2
≥ 30	≥ 30	30	30

Note that z_2 can, on occasion, be greater than 30.

If includeLastDay is true, add 1 to the calculated term in days.

3.40 DayBasis30E

The DayBasis30E model reflects the common European interpretation of the 30 day basis.

3.40.1 Relationships

	Class	Description	Notes
⇑	DayBasis30Abstract §3.38		

⇑:Inherits

3.40.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Add "30E" to the output stream. If includeLastDay is true, add a "+" to the output stream.

Integer termInDays(Date from, Date to) termInDays

from: Date The start date of the calculation.

to: Date The end date of the calculation.

Elapsed time in days between two dates.

If d_1, m_1 and y_1 are, respectively, the day, month and year numbers for the from date and d_2, m_2 and y_2 are, respectively, the day, month and year numbers for the to date, then the term in days is given by:

$$360 \times (y_2 - y_1) + 30 \times (m_2 - m_1) + (z_2 - z_1)$$

where z_1 and z_2 are defined by the following table:

d_1	d_2	z_1	z_2
< 30	< 30	d_1	$d2$
< 30	≥ 30	d_1	30
≥ 30	< 30	30	d_2
≥ 30	≥ 30	30	30

If includeLastDay is true, add 1 to the calculated term in days.

3.41 DayBasis30PSA

The Day30PSA model reflects the common Public Securities Association (PSA) interpretation of the 30 day basis.

3.41.1 Relationships

	Class	Description	Notes
⇑	DayBasis30Abstract §3.38		

⇑:Inherits

3.41.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Add "30PSA" to the output stream. If includeLastDay is true, add a "+" to the output stream.

Integer termInDays(Date from, Date to) termInDays

from: Date The start date of the calculation.

to: Date The end date of the calculation.

Elapsed time in days between two dates.

If d_1, m_1 and y_1 are, respectively, the day, month and year numbers for the from date and d_2, m_2 and y_2 are, respectively, the day, month and year numbers for the to date, then the term in days is given by:

$$360 \times (y_2 - y_1) + 30 \times (m_2 - m_1) + (z_2 - z_1)$$

where z_1 and z_2 are defined by the following table:

m_1	y_1 leap year?	d_1	d_2	z_1	z_2
2	Y	< 29		$d1$	$d2$
2	Y	≥ 29	< 30	30	d_2
2	Y	≥ 29	≥ 30	30	30
2	N	< 28		d_1	d_2
2	N	≥ 28	< 30	30	d_2
2	N	≥ 28	≥ 30	30	30
$\neq 2$		< 30		d_1	d_2
$\neq 2$		≥ 30	< 30	30	$d2$
$\neq 2$		≥ 30	≥ 30	30	30

If includeLastDay is true, add 1 to the calculated term in days.

3.42 DayBasisActual

The actual date basis calculates day counts by simply counting the elapsed days between the two dates.

3.42.1 Relationships

	Class	Description	Notes
⇑	DayBasisModel §3.37		

⇑:Inherits

3.42.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Add “Actual” to the output stream. If includeLastDay is true, add a “+” to the output stream.

Integer termInDays(Date from, Date to) termInDays
from: Date The start date of the calculation.
to: Date The end date of the calculation.

Elapsed time in days between two dates.

The term in days is simply the actual number of days between the two dates. If includeLastDay is true, add 1 to the calculated term in days.

3.43 DayBasisNL

The DayBasisNL class is essentially an actual day count, with the existence of leap years ignored.

3.43.1 Relationships

	Class	Description	Notes
⇑	DayBasisModel §3.37		

⇑:Inherits

3.43.2 Operations

print(OutputStream stream) print
stream: OutputStream The stream to print onto.

Print the object. Add "NL" to the output stream. If includeLastDay is true, add a "+" to the output stream.

Integer termInDays(Date from, Date to) termInDays
from: Date The start date of the calculation.
to: Date The end date of the calculation.

Elapsed time in days between two dates.

If d_1, and y_1 are, respectively, the day of year and year numbers for the from date and d_2, and y_2 are, respectively, the day of year and year numbers for the to date, then the term in days is given by:

$$365 \times (y_2 - y_1) + (z_2 - z_1)$$

where z_1 and z_2 are defined by the following table:

month	leap year?	d	z
1			d
2		≤ 28	d
2	Y	29	$d-1$
> 2	N		d
> 2	Y		$d-1$

If includeLastDay is true, add 1 to the calculated term in days.

3.44 NullDateClassifierModel

This classifies all dates as business days.

3.44.1 Relationships

	Class	Description	Notes
↑	DateClassifier §2.3		

↑:Realizes

3.44.2 Operations

Boolean isHoliday(Date date, DateClassifier environment) isHoliday
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

Date falls on a holiday? Return false.

Boolean isWeekend(Date date, DateClassifier environment) isWeekend
date: Date The date to test.
environment: DateClassifier The set of date classifiers that may affect this date classifier. The default value is this.

Date falls on a weekend? Return false.

DateClassifier compose(DateClassifier arg) compose
arg: DateClassifier

Compose two date classifiers. Return arg.

String identifier() identifier

Unique identifier. Return “Null”

Boolean precedes(DateClassifier other) precedes

other: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. Return false.

DateClassifier union(DateClassifier arg) union

arg: DateClassifier

Create the union of two date classifiers. Return arg.

Boolean isSimple() isSimple

This is a simple date classifier? Return true.

DateClassifier predecessorsOf(DateClassifier test) predecessorsOf

test: DateClassifier The date classifier to test against.

The predecessors of a date classifier. Return this classifier.

«Static Method» DateClassifier defaultInstance() defaultInstance

The default null date classifier. Return an instance of this class.

«Static Method» DateClassifier local() local

Local date classifier. Return the date classifier which encodes the holidays and weekends of the location running the system.

3.45 NullDateRollerModel

A date roller that ignores business calendars.

3.45.1 Relationships

	Class	Description	Notes
↑	DateRoller §2.7		

↑:Realizes

3.45.2 Operations

Date roll(DateClassifier classifier, Date date, Date startDate, Date lastDate) roll

classifier: DateClassifier The classifier for determining whether this date falls on an acceptable date.
date: Date The date to roll.
startDate: Date In the case of dates constructed by adding to another date, the original date before any addition occurred. The default value is date.
lastDate: Date In the case of dates constructed by adding a complex series of elements to a start dates, last date refers to the date immediately prior to this date being constructed. The default value is startDate.
Raises: ImmobileDateException

Roll the supplied date. Return date.

String identifier() identifier

The unique identifier. Return "Null".

3.46 PeriodModel

A concrete implementation of the Period interface.

3.46.1 Relationships

	Class	Description	Notes
↑	Period §2.10		
↑	Identifiable		
⇓	PeriodWithRollerModel §3.47		

⇓:Inherited by ↑:Realizes

3.46.2 Attributes

amount: Number The amount of the unit to add.

unit: PeriodUnit The time unit to add/subtract.

identifier: String The identifier for the period. In the case of anonymous periods, this may be nil.

3.47 PeriodWithRollerModel

A period which consists of an amount of time to add to the date and a date rolling convention to roll the resulting date to an acceptable date.

3.47.1 Relationships

	Class	Description	Notes
⇑	PeriodModel §3.46		
↑	PeriodWithRoller §2.11		
↔	DateRoller §2.7	roller 0..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

3.47.2 Operations

Date applyTo(Date date, DateClassifier classifier) applyTo
date: Date
classifier: DateClassifier
Raises: ImmobileDateException

Apply this period to the supplied date. If there is an initial period, then apply the initial period to $date$ to give $date_1$, otherwise $date_1 = date$. Add the associated amount to $date_1$ to give $date_2$. If there is an associated roller, use the associated roller to roll $date_2$, using $date$ as the start date and $date_1$ as the last date, and return the resulting date.

3.48 PeriodReferenceDataModel

A wrapper for a period so that periods can be managed as reference data. All period operations are passed through to the associated model.

3.48.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	Period §2.10		
↔	Period §2.10	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

3.49 PeriodUnitModel

A concrete realization of the PeriodUnit interface. This class is an abstract class; subclasses provide implementation behavior.

3.49.1 Relationships

	Class	Description	Notes
↑	PeriodUnit §2.12		
⇓	CalendarDayPeriodUnitModel §3.51		
⇓	NonWeekendDayPeriodUnit-Model §3.53		
⇓	BusinessDayPeriodUnitModel §3.50		
⇓	WeekPeriodUnitModel §3.55		
⇓	MonthPeriodUnitModel §3.52		
⇓	YearPeriodUnitModel §3.56		
⇓	QuarterPeriodUnitModel §3.54		

⇓:Inherited by ↑:Realizes

3.50 BusinessDayPeriodUnitModel

A period unit in terms of business days.

3.50.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.51 CalendarDayPeriodUnitModel

A period unit in terms of days, without reference to holidays or weekends.

3.51.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.52 MonthPeriodUnitModel

A period unit in terms of calendar months.

3.52.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.53 NonWeekendDayPeriodUnitModel

A period unit in terms of non-weekend days.

Deals involving USD or CAD have different spot date rules. The USD spot date is two non-weekend days after the trade date.

3.53.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.54 QuarterPeriodUnitModel

A period unit in terms of calendar quarters. Quarters are assumed to be 3 months in length, with the first quarter being January to March and the last quarter being October to December.

3.54.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.55 WeekPeriodUnitModel

A period unit of weeks. Weeks are assumed to be 7 days in length, with the first day being Monday and the last day being Sunday.

3.55.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.56 YearPeriodUnitModel

A period unit in terms of calendar years.

3.56.1 Relationships

	Class	Description	Notes
⇑	PeriodUnitModel §3.49		

⇑:Inherits

3.57 RepeatedPeriodModel

A concrete model for the RepeatedPeriod interface. Subclasses of this model allow the creation of the associated sequence of dates.

3.57.1 Relationships

	Class	Description	Notes
↑	RepeatedPeriod §2.13		
⇓	RepeatedPeriodBasicModel §3.58		
⇓	RepeatedPeriodEndModel §3.59		
↔	DateRoller §2.7	roller 0..1	→
↔	PeriodUnit §2.12	units 1..1	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

3.57.2 Attributes

isChained: Boolean = false Chain the dates together when creating this period?

sizeOfRollPeriod: Integer = 1 The amount of the associated period unit to add to a date for each step of the repeated period. For example if the period is 1 month and the sizeOfRollPeriod = 2 then each roll will add 2 months to the date.

identifier: String The unique identifier for the repeated period. This attribute may be nil for anonymous periods.

stopDate: Date The end date for this period.

3.58 RepeatedPeriodBasicModel

This is a repeated period where a sequence of dates is generated by a repeated application of the roll period. The roll period is given by multiplying the period unit by the 'sizeOfRollPeriod' attribute. The number of applications is given by the 'numberOfRollPeriods' attribute.

3.58.1 Relationships

	Class	Description	Notes
⇑	RepeatedPeriodModel §3.57		

⇑:Inherits

3.58.2 Attributes

numberOfRollPeriods: Integer = 0 The number of periods that make up this repeated period. For example. Period = 1 Month. sizeOfRollPeriod = 2 (2 months/roll). numberOfRollPeriods = 3 (roll 3 times).

3.58.3 Operations

Collection<Date> dates(Date date, DateClassifier classifier) dates
date: Date The date to start the repeated period at.
classifier: DateClassifier The date classifier for date rolling.
Raises: ImmobileDateException

Returns a sorted collection of dates that represents the application of this repeated period to an initial date.

Let $\{d_0, \ldots, d_n\}$ be the sequence of dates generated. In all cases, $d_0 = date$. Using the notation $d + n \cdot u$ for the application of n units of period unit u to date d then:

If the repeated period is chained, then

$$d_i = d_{i-1} + sizeOfRollPeriod \cdot unit$$

If the repeated period is not chained, then

$$d_i = d_0 + (sizeOfRollPeriod \times i) \cdot unit$$

For example, a chained roller:
period = 1 Month
sizeOfRollPeriod = 2
numberOfRollPeriods = 3

$$\begin{aligned} d_0 &= StartDate \\ d_1 &= StartDate + (1M \times 2) \;\; roll \\ d_2 &= d_1 + (1M \times 2) \;\; roll \\ d_3 &= d_2 + (1M \times 2) \;\; roll \end{aligned}$$

Unchained:

$$\begin{aligned} d_0 &= StartDate \\ d_1 &= StartDate + (1 \times (1M \times 2)) \;\; roll \\ d_2 &= StartDate + (2 \times (1M \times 2)) \;\; roll \\ d_3 &= StartDate + (3 \times (1M \times 2)) \;\; roll \end{aligned}$$

Duplicate dates are reduced to singletons[1]. If $d_0 > d_n$ then a single date, d_n, is returned.

Date stopDate() stopDate

$stopDate = startDate + numberOfRollPeriods / timessizeOfRollPeriod$

3.59 RepeatedPeriodEndModel

A repeated period with an explicit end point. The dates associated with this period are generated until the end point has been reached. Subclasses define the way that the end date is derived.

[1] Duplicate dates are possible where rolling conventions force several dates to the same date. eg. Friday is a US holiday. a USD deal done in Australia on Thursday or Friday have spot on following Tuesday.

3.59.1 Relationships

	Class	Description	Notes
⇑	RepeatedPeriodModel §3.57		
⇓	RepeatedPeriodEndDateModel §3.60		
⇓	RepeatedPeriodEndPeriod-Model §3.61		

⇑:Inherits ⇓:Inherited by

3.59.2 Operations

Collection<Date> dates(Date date, DateClassifier classifier) dates
date: Date The date to start the repeated period at.
classifier: DateClassifier The date classifier for date rolling.
Raises: ImmobileDateException

Returns a sorted collection of dates that represents the application of this repeated period to an initial date.

Let $\{d_0, \ldots, d_n\}$ be the sequence of dates generated. In all cases, $d_0 = date$. The end date, d_n is given by the end date provided by the applyTo() operation.

Using the notation $d + i \cdot u$ for the application of i units of period unit u to date d then:

If the repeated period is chained, then the dates $d_1, \ldots, d_{n-1}$ are generated by:

$$d_i = d_{i-1} + sizeOfRollPeriod \cdot unit$$

If the repeated period is not chained, then the dates $d_1, \ldots, d_{n-1}$ are generated by:

$$d_i = d_0 + (sizeOfRollPeriod \times i) \cdot unit$$

Duplicate dates are reduced to singletons[2].

In both cases, d_{n-1} is the last date generated before d_n is reached or passed. If $d_0 > d_n$ then a single date, d_n, is returned.

It is possible to have a partial period at the end of the sequence.

Date stopDate(Date date, DateClassifier classifier) stopDate
date: Date
classifier: DateClassifier
Raises: ImmobileDateException

[2]Duplicate dates are possible where rolling conventions force several dates to the same date.

Apply this period to the supplied date. This operation must be explicitly implemented by subclasses, as it is used to provide the stop point for the period.

3.60 RepeatedPeriodEndDateModel

A repeated period where the end date is explicitly fixed.

3.60.1 Relationships

	Class	Description	Notes
⇑	RepeatedPeriodEndModel §3.59		

⇑:Inherits

3.60.2 Operations

Date stopDate(DateClassifier classifier) stopDate
classifier: DateClassifier

The end date for the period. Return the stopDate.

3.61 RepeatedPeriodEndPeriodModel

A repeated period where the final date is determined by adding a period unrelated to the repeated period to the start date.

3.61.1 Relationships

	Class	Description	Notes
⇑	RepeatedPeriodEndModel §3.59		
↔	Period §2.10	endPeriod	→

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

3.61.2 Operations

Date stopDate(Date date, DateClassifier classifier) stopDate
date: Date
classifier: DateClassifier
Raises: ImmobileDateException

Return the result of applying the associated stop period to the supplied date. $stopDate = startDate + stopPeriod$.

3.62 RepeatedPeriodReferenceDataModel

A wrapper for a repeated period model so that repeated periods can be managed as pieces of reference data. All operations of the period unit interface are implemented by delegating to the associated model.

3.62.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	RepeatedPeriod §2.13		
↔	RepeatedPeriod §2.13	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.63 SimpleDateClassifierModel

An abstract class for grouping simple holiday specifications (ie: those not invlolving union or composition) together.

3.63.1 Relationships

	Class	Description	Notes
↑	SimpleDateClassifier §2.4		
⇓	DateClassifierHolidayArbitrary-Model §3.64		
⇓	DateClassifierHolidayRegular-Model §3.65		
⇓	DateClassifierHolidayRelative-Model §3.66		
↔	SimpleDateClassifier §2.4	predecessors 0..n	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.63.2 Attributes

weekendFlag: Boolean True if this is a weekend definition, rather than a holiday.

identifier: String The identifier for this holiday. This may be nil in the case of a anonymous holiday definition.

3.63.3 Operations

Date nthClosestHolidayTo(Date date, DateClassifier environment, Integer n) nthClosestHolidayTo

date: Date The reference date for computing the target.
environment: DateClassifier The set of potentially clashing holidays within a location.
n: Integer The nth closest holiday. This number is signed. Positive n indicates that you are looking for the nth closest holiday $after$ a date. Negative n indicates that you are looking for the nth closest holiday $before$ a date.

The nth closest holiday to the date, aDate. Implemented by subclasses.

Boolean precedes(DateClassifier argDateClassifier) precedes

argDateClassifier: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. If the argDateClassifier is not a SimpleDateClassifier then return false.

Each date classifier holds a collection of date classifiers that take precedence over it. This is a collection of predecessors. Each date classifier within the predecessors collection may have their own predecessors and each of these date classifiers may have their own predecessors and so on. The closure of all predecessors is the union of these predecessors, and their predecessors, etc....

To establish whether this date classifier has precedence over the argDateClassifier, hence returning true, both must be SimpleDateClassifiers and this date classifier must appear in the closure of predecessors of the argDateClassifier.

DateClassifier predecessorsOf(DateClassifier argDateClassifier) predecessorsOf

argDateClassifier: DateClassifier The date classifier to test against.

The predecessors of a date classifier. If this classifier precedes the argDateClassifier, as defined by the precedes() operation, return this classifier. Otherwise return an instance of the NullDateClassifierModel §3.44.

«Static Method» DateClassifier local() local

Local date classifier. Return the date classifier which encodes the holidays and weekends of the location running the system.

3.64 DateClassifierHolidayArbitraryModel

The DateClassifierHolidayArbitraryModel class can be used to model holidays that follow no simple rule or are essentially arbitrary. These holidays usually occur annually but the date may change from year to year. Hence, the easiest approach is to manually enter the date for the holiday, when known.

For example, the date that Easter Sunday falls upon is based upon the equinoxes and the phase of the moon, as well as an official church calendar. Thus, for 2000, Easter Sunday falls on 23rd April.

3.64.1 Relationships

	Class	Description	Notes
⇑	SimpleDateClassifierModel §3.63		

⇑:Inherits

3.64.2 Attributes

dates: Collection<Date> The dates on which the holidays fall.

3.64.3 Operations

Date nthClosestHolidayTo(Date date, Integer n, DateClassifier environment)

nthClosestHolidayTo

date: Date The reference date for computing where the holiday is.
n: Integer The nth closest holiday. This number is signed. Positive n indicates that you are looking for the nth closest holiday $after$ a date. Negative n indicates that you are looking for the nth closest holiday $before$ a date.
environment: DateClassifier The set of potentially clashing holidays within a location.

The date that this holiday falls upon.

First find the start and end points of the period that date falls within, using the period attribute. The start and end points are calculated by overlapping the period that date falls within and the preceeding and following periods.[3]

If there are no associated holiday dates within this period, then return nil. If these is a single associated holiday date in this period then return that date. If there is more than one holiday date between the start and end dates then the closest

[3] See DateClassifierHolidayRegularModel §3.65 for a discussion of this "feature".

holiday date to the supplied date is returned; if two dates are equidistant, then the earlier holiday date is returned.

Boolean isWeekend(Date date) isWeekend
date: Date The date to test.

Date falls on a weekend? Return true if weekendFlag() is true and the argument date is contained in the dates collection.

Boolean isHoliday(Date date) isHoliday
date: Date The date to test.

Date falls on a holiday?

Return true if weekendFlag() is false and the argument date is in the dates collection.

3.65 DateClassifierHolidayRegularModel

A regularly recurring holiday. That is a holiday that falls upon a given date regularly (usually annually). For example Christmas day is the 25th of December of each year. These holidays usually occur once a year. This model, however, allows arbitrary periods over which the holiday can occur. Thus a holiday can be modelled in terms of day and month, and recur according to the associated period.

Once the period within which the date falls has been identified, the date is then rolled to the correct position. Composite rollers may be required (eg. Christmas may be modeled by first taking the date to the 12th month and then the 25th day within that month.)

The date you are finding is calculated by taking the *closest* date to the test date. This convention may mean that the date you are finding and the test date may fall into different periods. Using the closest date allows sensible behaviour for the DateClassifierHolidayRelativeModel §3.66 model where relative holidays can cross period boundaries. For example, the new year is the first of January in a year. New year's eve is the preceding 31st of December. If the new year is defined as a day after New Years Eve, then the new year for 2-Jan-2005 should be 1-Jan-2005. However, without the closest date formulation, the target New Years Eve will be 31-Dec-2005 and the new year 1-Jan-2006.

3.65.1 Relationships

	Class	Description	Notes
⇑	SimpleDateClassifierModel §3.63		
↔	DateRoller §2.7	position 1..1	→

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.65.2 Attributes

period: Period The period over which the holiday repeats

3.65.3 Operations

Date nthClosestHolidayTo(Date date, DateClassifier environment) nthClosestHolidayTo
date: Date The date to test.
environment: DateClassifier The set of potentially clashing holidays.

Determine the date for the holiday, and if one exists on the date passed in, return true.

Let $e = environment.predecessorsOf(this)$. Take the date and move it to the first day of period. Then use the associated position date roller to move the date to the correct position, using e and the date, start date and last date set to the start of the period. The resulting date is d_0.

Apply the process in the above paragraph to a date created by subtracting and adding the period to the date, giving d_- and d_+.

Return the closest of $\{d_-, d_0, d_+\}$ to date. If two dates are equidistant then return d_0.

As an example, let the supplied date be 2-Jan-2003 and the target date defined as the 31st of October of each year. d_0 = 31-Oct-2003, d_- = 31-Oct-2002 and d_+ = 31-Oct-2004. The closest date to 2-Jan-2003 is d_-, which is the returned target date.

3.66 DateClassifierHolidayRelativeModel

A holiday that is relative to another holiday in terms of some period of time (the offset). For example, Good Friday falls on the Friday before the Easter holiday.

3.66.1 Relationships

	Class	Description	Notes
⇑	SimpleDateClassifierModel §3.63		
↔	SimpleDateClassifier §2.4	reference holiday 1..1	→
↔	DateRoller §2.7	roller 0..1	→

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.66.2 Attributes

offset: Period The amount to move the holiday (forwards or backwards) from the reference holiday.

3.66.3 Operations

Boolean precedes(DateClassifier other) precedes

other: DateClassifier The date classifier to compare this classifier against.

Compare two date classifiers for precedence. As well as the precedence rules of the super class, the reference holiday (and, transitively, anything that precedes the reference holiday) also precedes this classifier.

Boolean isHoliday(Date aDate) isHoliday

aDate: Date The date to test for being a holiday

Returns true if the date passed in, aDate, is found to be a holiday.

This could be implemented as follows

$n = 1$
while(true) { Holiday testHoliday;
testHoliday = this.reference.nthClosestHolidayTo(aDate, n) + this.offset();
if (testHoliday = aDate) {return (true) }
else if (testHoliday < aDate)
{return (false) };
$n = n + 1$;
};

Date nthClosestHolidayTo(Date aDate, DateClassifier environment, Integer n) nthClosestHolidayTo

aDate: Date The date from which the closest holiday will be found.

environment: DateClassifier The set of potentially clashing holidays within a location.
n: Integer The nth closest holiday to be found.

Find the nth closest holiday to the date passed in.

Use the supplied date, aDate, to compute the date for the reference holiday (using the *environment* to account for clashing holidays). Add the amount given by the offset and roll the resulting date by the associated roller, using the *environment* again, to roll for clashing holidays.

3.67 YearBasisModel

The YearBasisModel provides an abstract superclass for the various year conventions.

3.67.1 Relationships

	Class	Description	Notes
↑	YearBasis §2.14		
⇓	YearModelActual §3.70		
⇓	YearModel360 §3.68		
⇓	YearModel365 §3.69		

⇓:Inherited by ↑:Realizes

3.68 YearModel360

The 360 date basis assumes that a year has 360 days in it, irrespective of reality.

3.68.1 Relationships

	Class	Description	Notes
⇑	YearBasisModel §3.67		

⇑:Inherits

3.68.2 Operations

print(OutputStream stream) print
stream: OutputStream The stream to print onto.

Print the object. Add “360” to the output stream.

Double termInYears(Date from, Date to, DayBasis basis) termInYears
from: Date The start date of the calculation.
to: Date The end date of the calculation.
basis: DayBasis The day count convention to use.

Elapsed time in years between two dates.

The term in years can be calculated by dividing the term in days given by the day basis and the two dates by 360.

3.69 YearModel365

The 365 year count assumes that a year has 365 days in it, even if a year has a leap year.

3.69.1 Relationships

	Class	Description	Notes
⇑	YearBasisModel §3.67		

⇑:Inherits

3.69.2 Operations

print(OutputStream stream) print
stream: OutputStream The stream to print onto.

Print the object. Add “365” to the output stream.

Double termInYears(Date from, Date to, DayBasis basis) termInYears
from: Date The start date of the calculation.
to: Date The end date of the calculation.
basis: DayBasis The day count convention to use.

Elapsed time in years between two dates.

The term in years can be calculated by dividing the term in days given by the day basis and the two dates by 365.

3.70 YearModelActual

The YearModelActual performs year calculations based on an actual year length.

3.70.1 Relationships

	Class	Description	Notes
⇑	YearBasisModel §3.67		

⇑:Inherits

3.70.2 Operations

print(OutputStream stream) print

stream: OutputStream The stream to print onto.

Print the object. Add "Actual" to the output stream.

Double termInYears(Date from, Date to, DayBasis basis) termInYears

from: Date The start date of the calculation.

to: Date The end date of the calculation.

basis: DayBasis The day count convention to use.

Elapsed time in years between two dates.

The term in years needs to be calculated on a year-by-year basis, to allow for variations in the year length.

If the two dates run from year m to year n, with the year length of year i being l_i, the calendar start date of year i being s_i and the term in days between two dates, d_1 and d_2 (under the supplied day count basis) being denoted by $d_2 - d_1$ then the term in years is given by:

if $m = n$ $\quad (to - from)/l_m$

otherwise $\quad (s_{m+1} - from)/l_m + (to - s_n)/l_n + \Sigma_{i=m+1}^{n-1}(s_{i+1} - s_i)/l_i$

There is a complexity when calculating terms with some 30-day day bases and an Actual year basis. If a date falls on the end of a month, 30-day date bases usually adjust the number of days in the month according to a complex convention involving both the from and to dates in an asymmetrical manner. Applied strictly, the above formula will not reflect this adjustment when summing year segments, as the date at the start of the year will not cause the correct adjustments to occur.

As an example, consider the term calculated by a 30/Actual date basis from 31-Dec-1996 to 31-Dec-1997. The day count calculated by the 30 day basis is 360. However, the day count calculated from 31-Dec-1996 to 1-Jan-1997 is 1, and the day count from 31-Jan-1997 is 360, giving a discrepancy of 1 day.

If the sequence of year segments sums to a day count different to the directly calculated day count, then the final segment is adjusted to reflect the difference between the summed and directly calculated day counts. In the above example, the segment from 1-Jan-1997 to 31-Dec-1997 would be adjusted to be 359 days.

4 Exceptions

4.1 DateArithmeticException

The DateArithmeticException is raised when a piece of date arithmetic produces an illegal value.

4.1.1 Operations

Date date() date

Target date. Return the date that was having the date arithmetic applied to it.

Object arg() arg

The argument. Return the argument that was being applied to the date.

4.2 ImmobileDateException

The Immobile date exception is raised if it is not possible to roll a date onto a business or non-weekend day for some reason.

4.2.1 Operations

Date date() date

The immobile date.

DateRoller roller() roller

The rolling convention. Return the date roller that was attempting to move the date.

DateClassifier classifier() classifier

The date classifier. Return the date classifier that was being used to classify the date.

5 Enumerations

5.1 DateDirectionEnum

An enumeration listing the direction to move when finding a position from an associated period.

5.1.1 Relationships

	Class	Description	Notes
⇑	Enum		

⇑:Inherits

5.1.2 Operations

«Static Method» DateDirectionUnum forwards() forwards

Returns a unique instance with a name of "forwards".

«Static Method» DateDirectionEnum backwards() backwards

Returns a unique instance with a nmae of "backwards".

Collection elements() elements

Returns an ordered collection with elements returned from forwards and backwards.

5.2 DayOfWeekEnum

An enumeration of the days of the week. The days of the week start on Monday and end on Sunday.[2]

English is used to name the elements of this enumeration. These names are used for programming purposes only. The DateFormat §2.5 interface describes how to translate these elements into another language.

5.2.1 Relationships

	Class	Description	Notes
⇑	OrderedEnum		

⇑:Inherits

5.2.2 Operations

«Static Method» DayOfWeekEnum monday() monday

Return a unique instance with a name of “Monday” and an order of 1.

«Static Method» DayOfWeekEnum tuesday() tuesday

Return a unique instance with a name of “Tuesday” and an order of 2.

«Static Method» DayOfWeekEnum wednesday() wednesday

Return a unique instance with a name of “Wednesday” and an order of 3.

«Static Method» DayOfWeekEnum thursday() thursday

Return a unique instance with a name of “Thursday” and an order of 4.

«Static Method» DayOfWeekEnum friday() friday

Return a unique instance with a name of “Friday” and an order of 5.

«Static Method» DayOfWeekEnum saturday() saturday

Return a unique instance with a name of “Saturday” and an order of 6.

«Static Method» DayOfWeekEnum sunday() sunday

Return a unique instance with a name of “Sunday” and an order of 7.

«Static Method» Collection<Enum> elements() elements

Return an ordered collection of the instances returned from the monday(), tuesday(), wednesday(), thursday() friday(), saturday() and sunday() operations.

Integer isoNumber() isoNumber

The number of the day of the week, using the ISO conventions.[2]. Return the order attribute.

«Static Method» DayOfWeekEnum fromIsoNumber(Integer dayOfWeek)

fromIsoNumber

dayOfWeek: Integer The numeric day of the week.
Raises: EnumOutOfRangeException

Return the enumeration instance that has the same order as the dayOfWeek argument. If there is no such instance then raise a EnumOutOfRangeException.

5.3 EraEnum

An enumeration for the possible eras.

English is used to name the elements of this enumeration. These names are used for programming purposes only. The DateFormat §2.5 interface describes how to translate these elements into another language.

5.3.1 Relationships

	Class	Description	Notes
⇑	OrderedEnum		

⇑:Inherits

5.3.2 Operations

«Static Method» EraEnum before() before

Return an era with a name of “before” and an order of 1.

«Static Method» EraEnum after() after

Return an era with a name of “after” and an order of 2.

«Static Method» Collection<Enum> elements() elements

Return an ordered collection of the instances returned from the before() and after() operations.

5.4 MonthEnum

An enumeration listing the months of the year.

English is used to name the elements of this enumeration. These names are used for programming purposes only. The DateFormat §2.5 interface describes how to translate these elements into another language.

5.4.1 Relationships

	Class	Description	Notes
⇑	OrderedEnum		

⇑:Inherits

5.4.2 Operations

«Static Method» MonthEnum january() january

Return a unique instance with a name of “january” and an order of 1.

«Static Method» MonthEnum february() february

Return a unique instance with a name of “february” and an order of 2.

«Static Method» MonthEnum march() march

Return a unique instance with a name of “march” and an order of 3.

«Static Method» MonthEnum april() april

Return a unique instance with a name of “april” and an order of 4.

«Static Method» MonthEnum may() may

Return a unique instance with a name of “may” and an order of 5.

«Static Method» MonthEnum june() june

Return a unique instance with a name of “june” and an order of 6.

«Static Method» MonthEnum july() july

Return a unique instance with a name of “july” and an order of 7.

«Static Method» MonthEnum august() august

Return a unique instance with a name of “august” and an order of 8.

«Static Method» MonthEnum september() september

Return a unique instance with a name of “september” and an order of 9.

«Static Method» MonthEnum october() october

Return a unique instance with a name of “october” and an order of 10.

«Static Method» MonthEnum november() november

Return a unique instance with a name of “november” and an order of 11.

«Static Method» MonthEnum december() december

Return a unique instance with a name of “december” and an order of 12.

«Static Method» Collection<Enum> elements() elements

Return an ordered collection of the instances returned from the january(), february(), march(), april(), may(), june(), july(), august(), september(), october(), november() and december() operations.

Integer monthNumber() monthNumber

The number of the month from the start of the year. Return the order attribute.

«Static Method» MonthEnum fromMonthNumber(Integer month) fromMonthNumber

month: Integer The numeric month number.

Raises: EnumOutOfRangeException

Return the enumeration instance that has the same order as the month argument. If there is no such instance then raise a EnumOutOfRangeException.

5.5 QuarterEnum

An enumeration for the quarters within a year.

English is used to name the elements of this enumeration. These names are used for programming purposes only. The DateFormat §2.5 interface describes how to translate these elements into another language.

5.5.1 Relationships

	Class	Description	Notes
⇑	OrderedEnum		

⇑:Inherits

5.5.2 Operations

«Static Method» QuarterEnum firstQuarter() firstQuarter

Return an instance with a name of “first” and an order of 1.

«Static Method» QuarterEnum secondQuarter() secondQuarter

Return an instance with a name of “second” and an order of 2.

«Static Method» QuarterEnum thirdQuater() thirdQuater

Return an instance with a name of “third” and an order of 3.

«Static Method» QuarterEnum fourthQuarter() fourthQuarter

Return an instance with a name of “fourth” and an order of 4.

«Static Method» Collection<Enum> elements() elements

Return an ordered collection of the instances returned from the firstQuarter(), secondQuarter(), thirdQuarter() and fourthQuarter() operations.

6 Associations

Table 6: Dates— Associations

Association Role	Class	Card.	Notes
model			
model	DateFormat §2.5	1..1	→
reference data	DateFormatReferenceData-Model §3.16	0..1	
model			
model	DateRollerProgram §2.8	1..1	→
reference data	DateRollerReferenceDataModel §3.36	0..1	
model			
	DateRoller §2.7	1..1	→
	DateRollerReferenceDataModel §3.36	0..1	
within			
	PeriodUnit §2.12	0..*	→
	DatePositionModel §3.18	1..1	
program			
	Function	1..1	→
	DateRollerProgramModel §3.35	0..*	
unit			
unit	PeriodUnit §2.12	1..1	→
period	PeriodModelUnit	0..n	
initial			
initial period	Period §2.10	0..1	→
period	PeriodModel §3.46	0..n	
model			
model	Period §2.10	1..1	→
reference data	PeriodReferenceDataModel §3.48	0..1	
model			

Table 6: . . . continued

Association Role	Class	Card.	Notes
	Period §2.10	1..1	→
	PeriodReferenceDataModel §3.48	0..1	
roller			
	DateRoller §2.7	0..1	→
	PeriodWithRollerModel §3.47	0..n	
model			
model	PeriodUnit §2.12	1..1	→
reference data	PeriodUnitReferenceDataModel	0..1	
stopPeriod			
stop	Period §2.10	1..1	→
repeated period	RepeatedPeriodModelEndedPeriod	0..n	
unit			
unit	PeriodUnit §2.12	1..1	→
repeated period	RepeatedPeriodModel §3.57	0..n	
model			
model	RepeatedPeriod §2.13	1..1	→
reference data	RepeatedPeriodReferenceData-Model §3.62	0..n	
model			
	RepeatedPeriod §2.13	1..1	→
	RepeatedPeriodReferenceData-Model §3.62	0..n	
reference holiday			
reference	TargetedDateClassifier	1..1	→
tied holiday	DateClassifierHolidayModelRelative	0..n	
precedence			
predecessors	TargetedDateClassifier	0..n	→
successor	TargetedDateClassifierModel	0..n	
model			
	DateClassifier §2.3	1..1	→
	DateClassifierReferenceData-Model §3.2	0..1	
predecessors			
	SimpleDateClassifier §2.4	0..n	→
	SimpleDateClassifierModel §3.63	0..n	
reference holiday			

Table 6: . . . continued

Association Role	Class	Card.	Notes
	SimpleDateClassifier §2.4	1..1	→
	DateClassifierHolidayRelative-Model §3.66	1..1	
roller			
	DateRoller §2.7	0..1	→
	DateClassifierHolidayRelative-Model §3.66	0..n	
position			
	DateRoller §2.7	1..1	→
	DateClassifierHolidayRegular-Model §3.65	0..n	
components			
	DateClassifier §2.3	0..n	→
	DateClassifierUnionModel §3.3	0..n	
roller			
	DateRoller §2.7	0..1	→
	RepeatedPeriodModel §3.57	0..n	
units			
	PeriodUnit §2.12	1..1	→
	RepeatedPeriodModel §3.57	0..n	
endPeriod			
	Period §2.10		→
	RepeatedPeriodEndPeriod-Model §3.61		

→:Navigable ◊:Aggregate ♦:Composite

6.1 model

Role: model *Navigable* DateFormat, 1..1.
Role: reference data DateFormatReferenceDataModel, 0..1.
The model wrapped by the reference data object.

6.2 model

Role: model *Navigable* DateRollerProgram, 1..1.
Role: reference data DateRollerReferenceDataModel, 0..1.
The wrapped date roller model.

6.3 model

Role: *Navigable* DateRoller, 1..1.
Role: DateRollerReferenceDataModel, 0..1.

6.4 within

Role: *Navigable* PeriodUnit, 0..*.
Role: DatePositionModel, 1..1.

6.5 program

Role: *Navigable* Function, 1..1.
Role: DateRollerProgramModel, 0..*.

The program which, when evaluated, derives the new date. The function that this date roller is associated with must accept the correct 5 arguments.

6.6 unit

Role: unit *Navigable* PeriodUnit, 1..1.
Role: period PeriodModelUnit, 0..n.

The unit of period to add.

6.7 initial

Role: initial period *Navigable* Period, 0..1.
Role: period PeriodModel, 0..n.

An initial period to add to a date before the period proper is added.

6.8 model

Role: model *Navigable* Period, 1..1.
Role: reference data PeriodReferenceDataModel, 0..1.

The period that is being modeled.

6.9 model

Role: *Navigable* Period, 1..1.
Role: PeriodReferenceDataModel, 0..1.

The period that is being modeled.

6.10 roller

Role: *Navigable* DateRoller, 0..1.
Role: PeriodWithRollerModel, 0..n.
The date rolling convention to apply to the period.

6.11 model

Role: model *Navigable* PeriodUnit, 1..1.
Role: reference data PeriodUnitReferenceDataModel, 0..1.
The underlying model for the reference data.

6.12 stopPeriod

Role: stop *Navigable* Period, 1..1.
Role: repeated period RepeatedPeriodModelEndedPeriod, 0..n.
The period over which the repeated period is to run.

6.13 unit

Role: unit *Navigable* PeriodUnit, 1..1.
Role: repeated period RepeatedPeriodModel, 0..n.
The unit that the repeated period is expressed in.

6.14 model

Role: model *Navigable* RepeatedPeriod, 1..1.
Role: reference data RepeatedPeriodReferenceDataModel, 0..n.
The mode that this piece of reference data is a wrapper for.

6.15 model

Role: *Navigable* RepeatedPeriod, 1..1.
Role: RepeatedPeriodReferenceDataModel, 0..n.
The mode that this piece of reference data is a wrapper for.

6.16 reference holiday

Role: reference *Navigable* TargetedDateClassifier, 1..1.
Role: tied holiday DateClassifierHolidayModelRelative, 0..n.
The reference holiday that this holiday moves from.

6.17 precedence

Role: predecessors *Navigable* TargetedDateClassifier, 0..n.
Role: successor TargetedDateClassifierModel, 0..n.
The date classifiers that precede this date classifier.

6.18 model

Role: *Navigable* DateClassifier, 1..1.
Role: DateClassifierReferenceDataModel, 0..1.
The wrapped model for the reference data.

6.19 predecessors

Role: *Navigable* SimpleDateClassifier, 0..n.
Role: SimpleDateClassifierModel, 0..n.
The date classifiers that precede this date classifier.

That is, if two holidays fall on the same day, and one is more important than the other, then this would mean that the less important holiday would roll to another day, according to some business rule. Thus, for a given holiday, the precedence is a collection of all the holidays that are more important than it. Also, note that each of these predecessors may have their own collection of predecessors.

6.20 reference holiday

Role: *Navigable* SimpleDateClassifier, 1..1.
Role: DateClassifierHolidayRelativeModel, 1..1.
The reference holiday that this holiday moves from.

6.21 roller

Role: *Navigable* DateRoller, 0..1.
Role: DateClassifierHolidayRelativeModel, 0..n.
The roller to use when moving the target date.

6.22 position

Role: *Navigable* DateRoller, 1..1.
Role: DateClassifierHolidayRegularModel, 0..n.
The date roller that forces a date to a position within the period (ie, the day within the period).

6.23 components

Role: *Navigable* DateClassifier, 0..n.
Role: DateClassifierUnionModel, 0..n.

The individual holidays, weekends and composites that make up a composite classifier.

6.24 roller

Role: *Navigable* DateRoller, 0..1.
Role: RepeatedPeriodModel, 0..n.

6.25 units

Role: *Navigable* PeriodUnit, 1..1.
Role: RepeatedPeriodModel, 0..n.

6.26 endPeriod

Role: *Navigable* Period.
Role: RepeatedPeriodEndPeriodModel.

The end period to apply to the date.

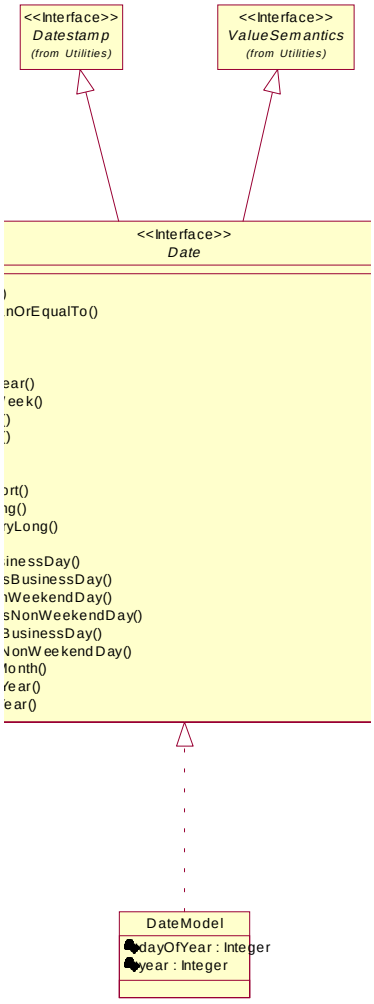


Figure 1: Class Diagram— Dates

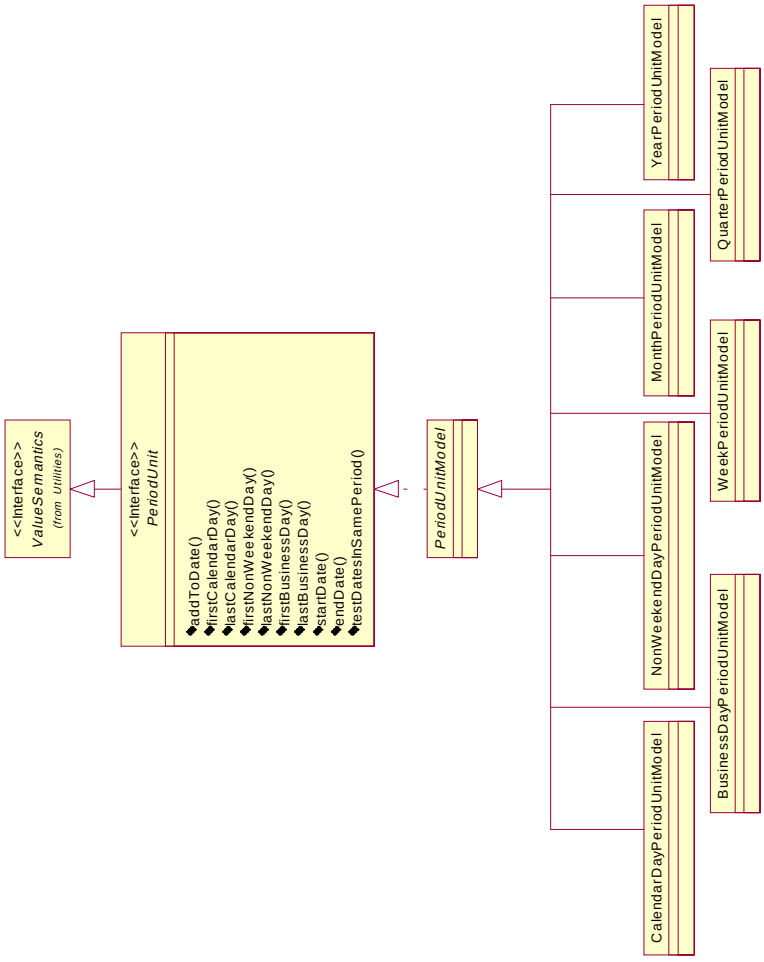


Figure 2: Class Diagram— Period Units

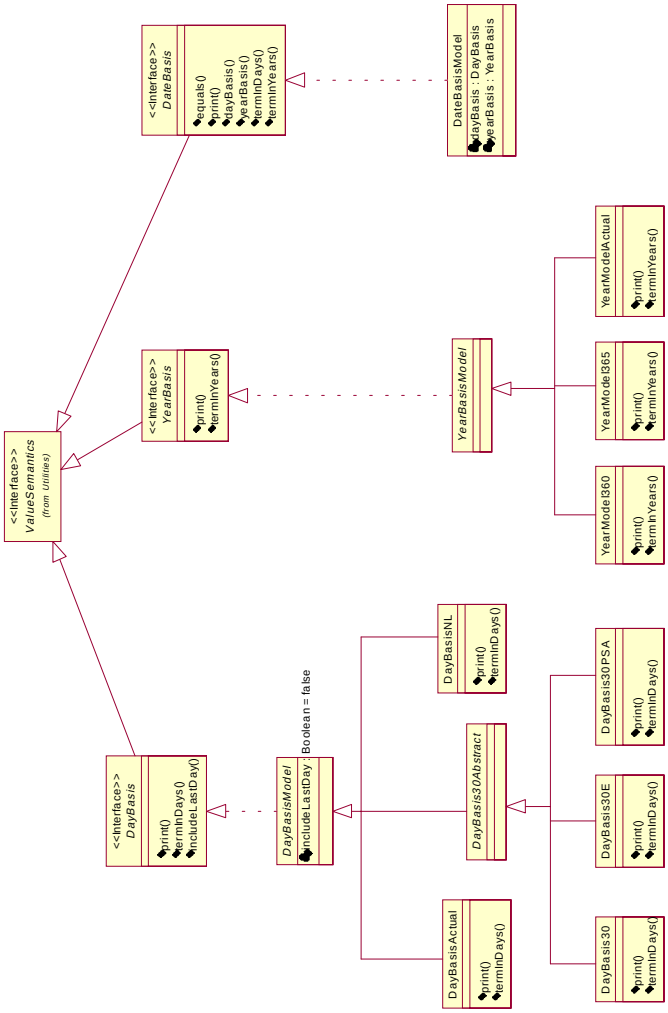


Figure 3: Class Diagram— Date Arithmetic (Date Bases)

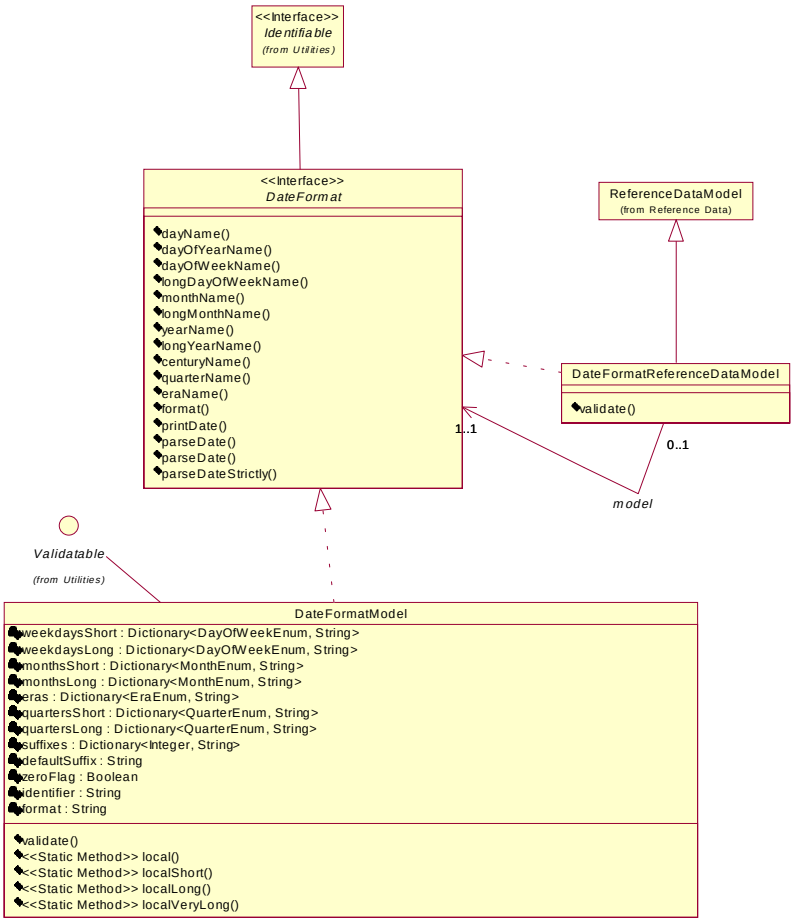


Figure 4: Class Diagram— Date Printing

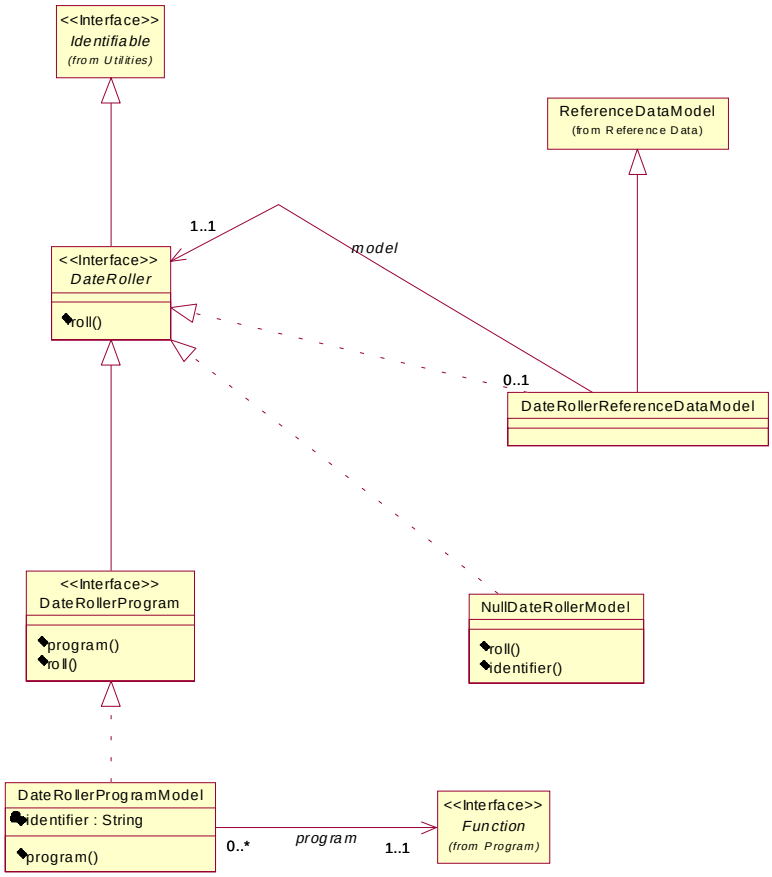


Figure 5: Class Diagram— Date Rolling

Figure 6: Class Diagram— Date Examples

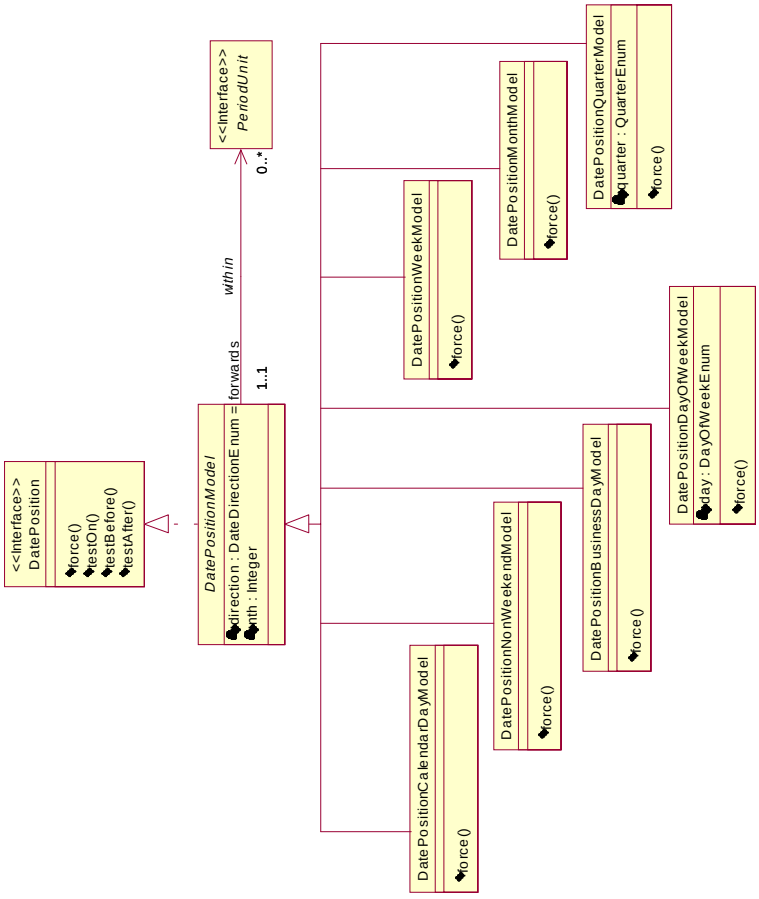
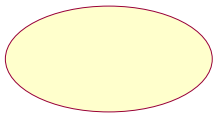
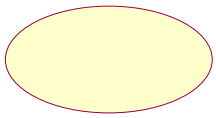


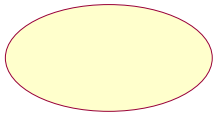
Figure 7: Class Diagram— Date Forcing



Date Rolling Example - Following



Date Rolling Example - Modified Following



Date Rolling Example - FX Following

Figure 8: Class Diagram— Date Rolling Examples

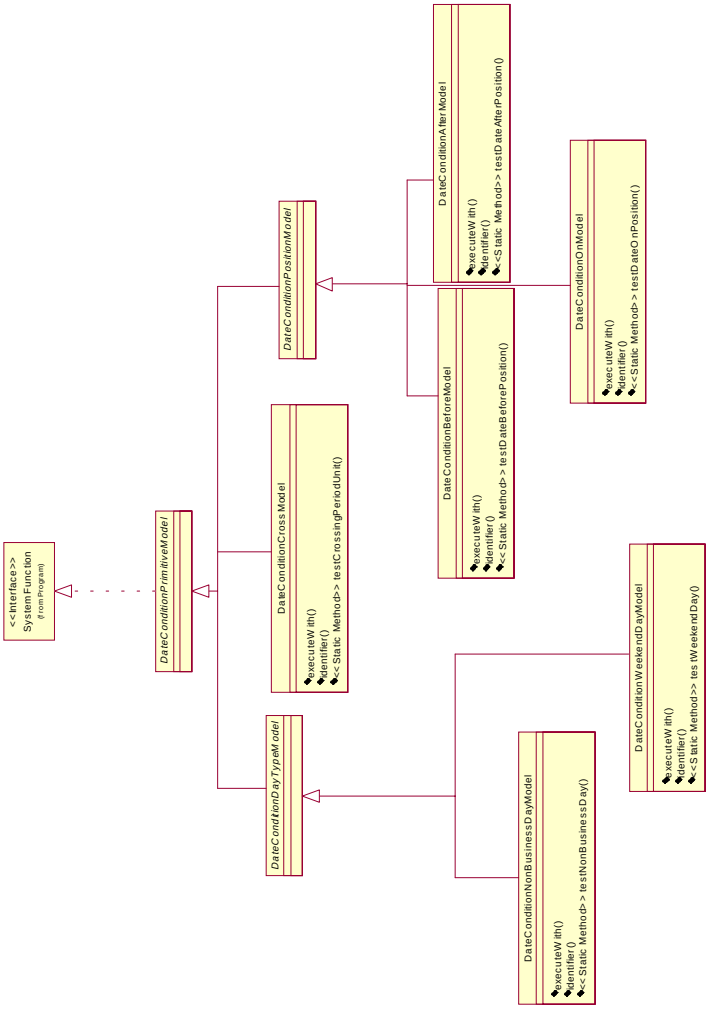


Figure 9: Class Diagram— Date Rolling - Primitive Conditions

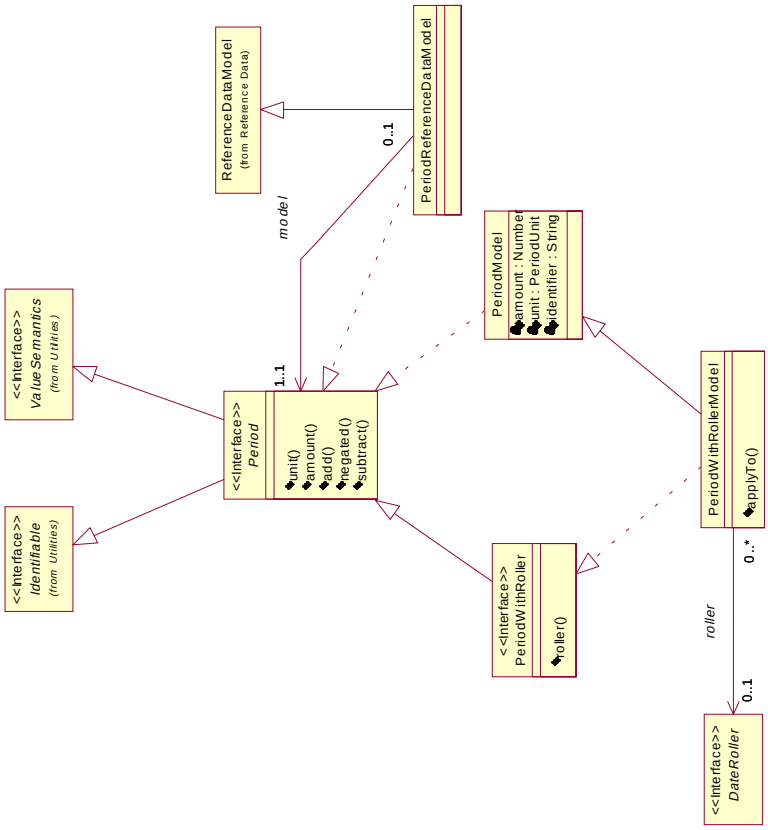


Figure 10: Class Diagram— Periods

Figure 11: Class Diagram— Period Examples

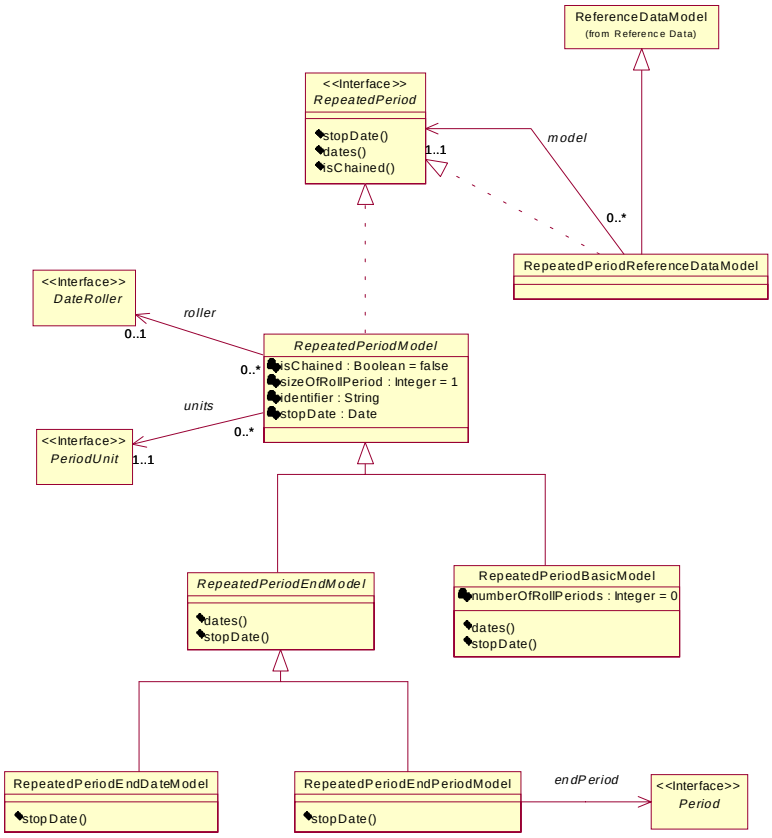


Figure 12: Class Diagram— Repeated Periods

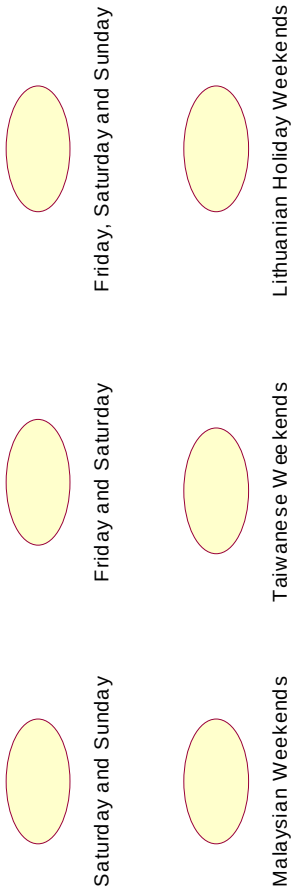


Figure 13: Class Diagram— Weekend Examples

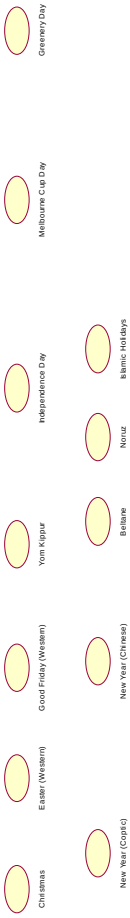


Figure 14: Class Diagram— Holiday Examples

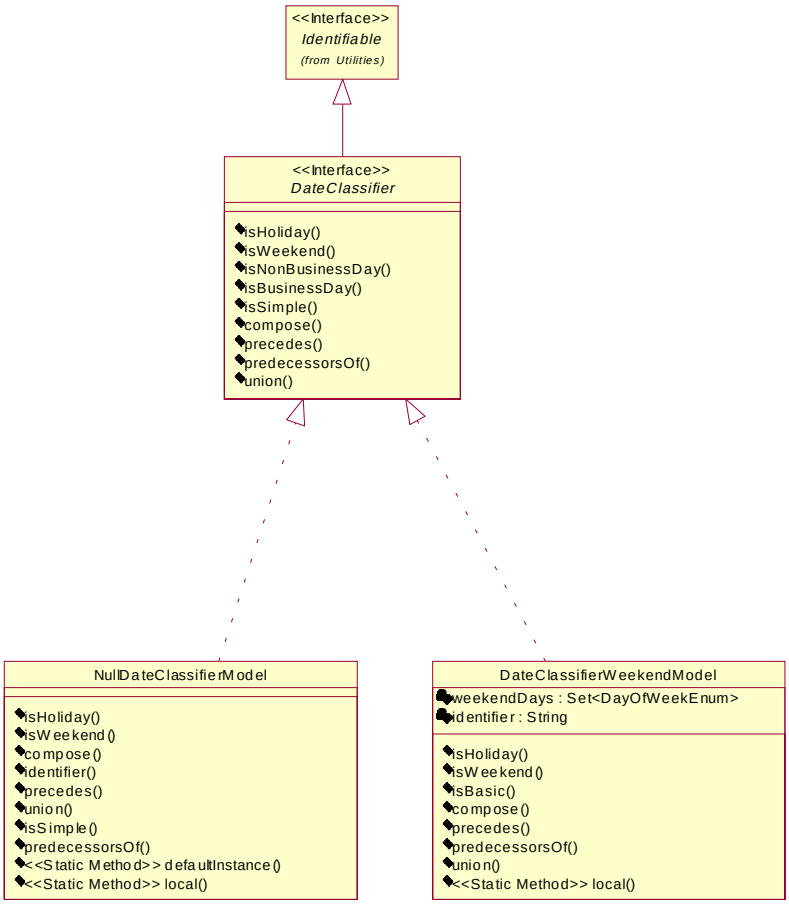


Figure 15: Class Diagram— Date Classifiers (Null and Weekend)

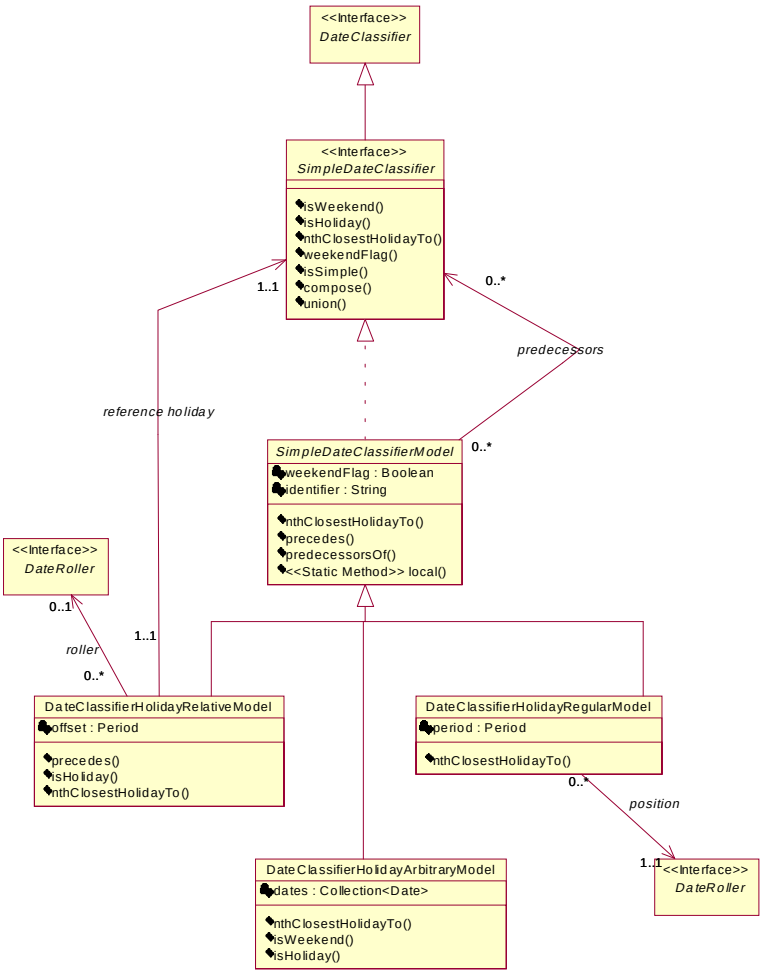


Figure 16: Class Diagram— Date Classifier (Holidays)

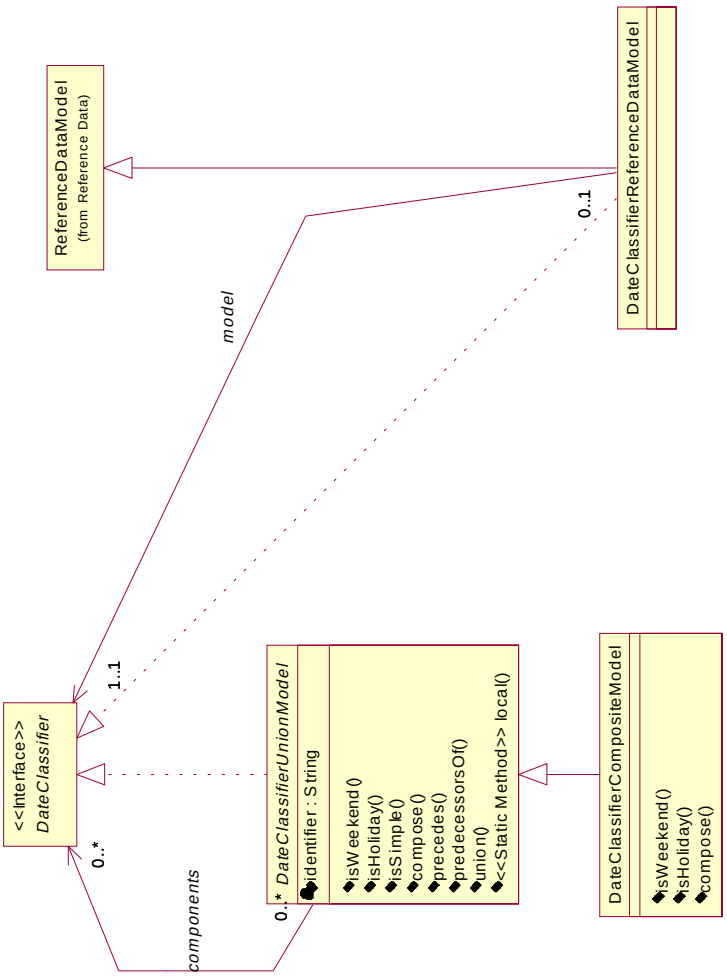


Figure 17: Class Diagram— Date Classifiers (Complex)

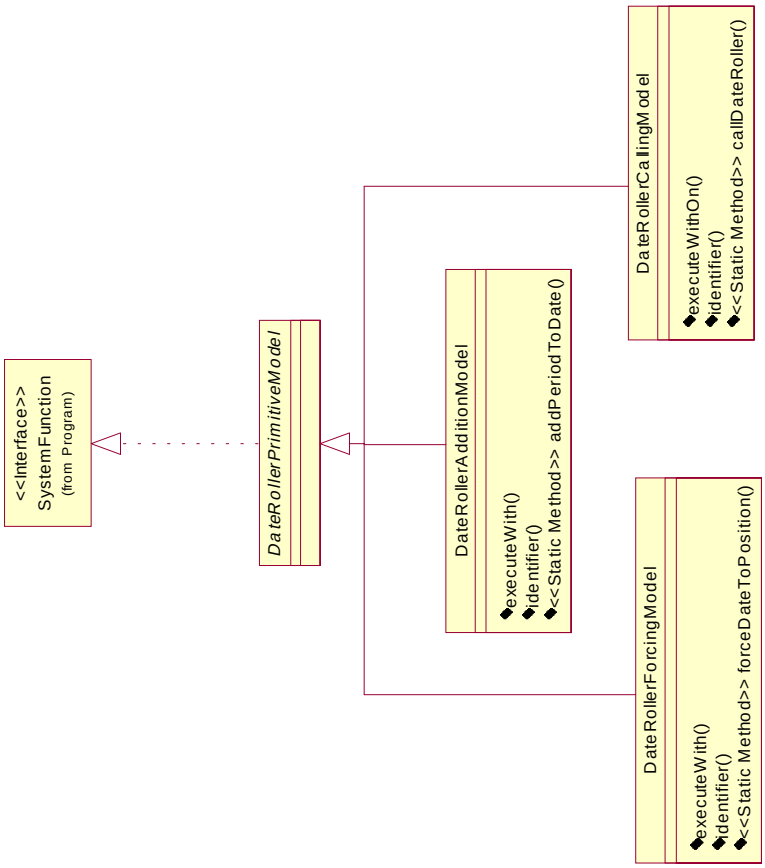


Figure 18: Class Diagram— Date Rolling - Primitive Actions

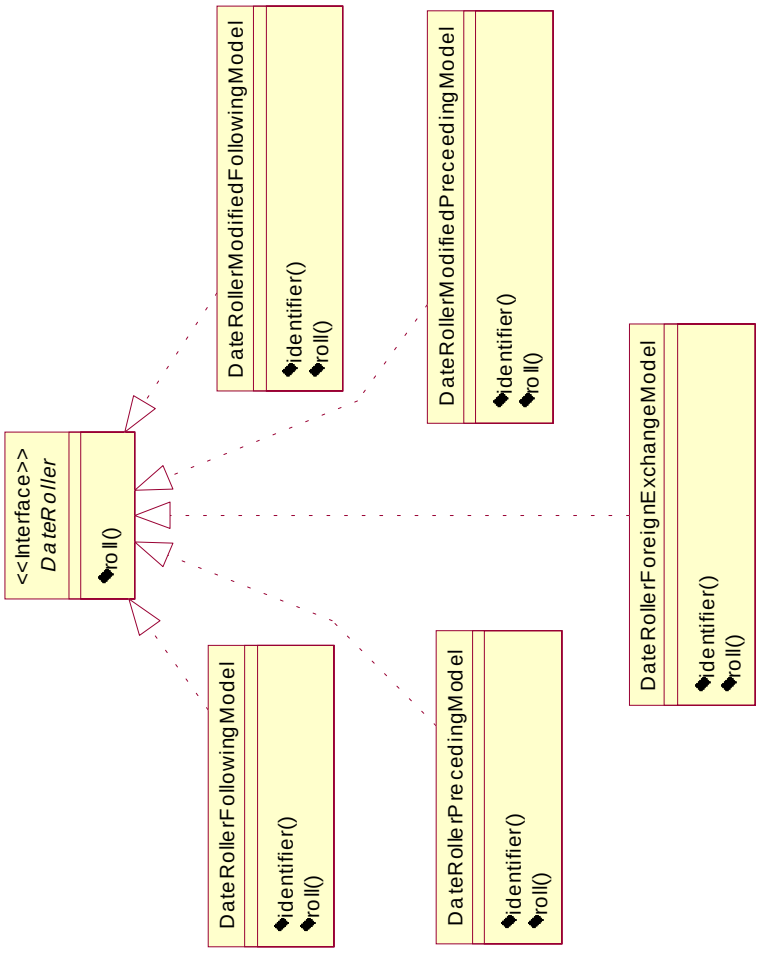


Figure 19: Class Diagram— Date Rolling - Standard Rollers

References

[1] Patrick J. Brown. *Bond Markets: Structures and Yield Calculations*. Amacom, 1998.

[2] International Organization for Standardization (ISO). *Data Elements and Interchange Formats — Information Interchange — Representation of Dates and Times*, number ISO 8601, 1988.
http://www.iso.ch/markete/8601.pdf.

[3] Robert Steiner. *Mastering Financial Calculations*. Pitman Publishing, 1998.