Short documentation for EIFFELTIME library

Introduction

The library EiffelTime is built on three notions of time. The absolute notion (for example, what time is it? 3:45 p.m.) is used for events. It may be useful also to deal with an interval between two events (the meeting is between 3:45 p.m. and 6:00 p.m.). Finally EiffelTime has the notion of duration, which is the length of an interval (the meeting will take 2:15). The notion of absolute is linked with duration by the use of an origin.

1. ABSOLUTE TIME

The classes dealing with date and those dealing with time have almost the same construction. At the top of the hierarchy are the constants and the notion of value (TIME_VALUE, DATE_VALUE, DATE_TIME_VALUE). From this notion comes two kinds of heirs which are the absolute notion of time (classes DATE, TIME and DATE_TIME) and the notion of duration (classes DATE_DURATION, TIME_DURATION, DATE_TIME_DURATION).

DATE, TIME and DATE_TIME inherit from the deferred class ABSOLUTE. It implies that instances of these classes are used as absolutes. We can imagine an oriented axis on which are reported values. ABSOLUTE inherits COMPARABLE, there is a complete order inside the class and its heir. ABSOLUTE is a client of DURATION, so that each instance of ABSOLUTE is linked with the duration between the origin and itself. The default way to compare absolute objects is to compare their respective duration to each other.

1.1 TIME

TIME deals with hour, minute and second. Is is possible to use more precision for time (there is no limit inside the class). See below (5. More precision in TIME) for documentation. This section deals only with sec-

ond.

Creation

There are three ways to create an instance of the class TIME: by choosing the time (make), by getting the time from the system (make_now), or by choosing the number of seconds elapsed from the origin (make_by_seconds). The arguments of make and make_by_seconds have to respect the range of a day (see preconditions).

Origin and cyclic representation

The origin is 0 hour 0 minute and 0 second. Notion of time is relative to a day in a cyclic representation: days begin at 0:0:0 and end at 23:59:59. If a second is added to 23:59:59 then the result will be 0:0:0. Subtracting a minute to 0:0:0 will yield 23:59:0.

Comparison

Instances of TIME may be compared. Functions <, + > and >= are available. Function is_equal must be use to test equality. = will compare references.

Measurement

The duration linked to an instance of TIME (attribute duration) is an instance of TIME_DURATION. It is the duration from the origin until the current time. The function seconds returns the number of seconds since the origin. This function may be useful to get the number of seconds between two events. The feature - creates an interval between two instances of TIME. The duration of this interval is given by the function duration. However, this duration is not canonical (See Duration, section 2, for precisions). In TIME, the feature relative_duration returns the same duration, but more efficiently and also it is canonical.

Operations

- Set directly hour, minute and second with set_hour, set_minute and set_second. Arguments must satisfy the rules of creation.
- Adding hours, minutes and seconds with features hour_add, minute_add and second_add. Features add and + take an instance of TIME_DURATION as anPage 3 argument and add it to the current time.
- Moving to the next or the previous hour, minute or second with features hour_forth, hour_back, minute_forth, minute_back, second_forth and second_back. It is faster to use these features rather than those above (hour_back <-> hour_add (-1)).

1.2. DATE

DATE deals with year, month and day. It is more complicated since there is no regular period in dates: each month contains its own total of days and there are leap years. That is why some peculiarities appear while manipulating objects of this class. There is no limit for a date (inside the class). The only limit comes from INTEGER representation. If INTEGER size is 32 bits (most common case), and as long as the basic unit is a day, the range for a date is from (- 2^31) to 2^31 (days), i.e. 5.8 million years from the origin.

Creation

There are also three ways to create an instance of the class DATE: by choosing the date (make, make_month_day_year, make_day_month_year), by getting the date from the system (make_now), or by choosing the number of days elapsed from the origin (make_by_days). The arguments of each creation procedure have to respect the common range (See preconditions).

Origin

The origin is 01/01/1600.

Comparison

Instances of DATE may be compared. Functions <, + > and >= are available. Function is_equal must be use to test equality, = will compare references.

Measurement

Each instance of DATE has a function (duration) which returns the duration since the origin until the current date (it is an instance of DATE_DURATION). This duration is definite, i.e. it contains only days (See below). However, it may be useful to deal directly with days (no need of DATE_DURATION). In this case, the function days of DATE yields the number of days since origin.

Status Report

Much information may be gotten from functions written in this part. Most of them are used within the class, but they are exported at the same time.

Operations

DATE operations looks like TIME operations:

- Setting directly year, month and day with set_year, set_month and set_day. Arguments must satisfy the rules of creation. These rules are more complicated than those of TIME. For example it is not allowed to set day to 31 if the current month is April, whereas it is allowed if the month is January. It is the same rules as for make. The same thing for years: It is not allowed to set year to a non-leap year if the current date is February 29th of a leap year. However, two features are available to set month and year even if day is too large: set_month_cut_days and set_year_cut_days will cut day down to the largest value allowed.
- Adding years, months and days with features year_add, month_add and day_add. There is no requirement to add a year or a month. However, these features have to return a correct result, i.e. day is checked before each addition. Adding one month to August 31st will yield September 30th. 31 is cut to 30 since there are only 30 days in september. Features add and + take an instance of DATE_DURATION in argument and add it to the current date. It is written so that years and months are added first, the days last. (see DATE_DURATION below)
- Moving to the next or the previous year, month or day with features year_forth, year_back, month_forth, month_back, day_forth and day_back. It is the same but faster to use these features rather than those upper (year_back <-> year_add (-1)).
- Features relative_duration and definite_duration return the duration between the current date and the argu-

ment. The first one returns a result which is canonical (See definitions below), while the second one returns a definite result but may be not canonical.

For example, date1 is April 20th and date2 is May 28th. Both features will yield instances of DURATION; however, relative_duration will yield 1 month and 8 days whereas definite_duration will yield 38 days.

1.3. DATE_TIME

The aim is to gather the time with the date. DATE_TIME is client of TIME and DATE (see inheritance relation). Some features from DATE and TIME are re-written since they are useful within the class. Many other features may be called indirectly with the correct attribute (time or date).

Creation

There are still several ways to create an instance:

- by choosing value for all the attributes of the date and the time (make).
- by getting the time from the system (make_now).
- by gathering an instance of DATE with an instance of TIME (make_by_date_time). This feature copies only the references of its arguments, so that if the time (or the date) is changed, the instance previously initialized will be also changed. If this effect has to be avoided, cloning the arguments is required.
- by encapsulating an instance of DATE (make_by_date). The attribute time is set to the origin, i.e. 0:0:0. The attribute date is set with the same reference as the argument (See comment of the previous section).

Access

To make it easier calls to features of TIME and DATE, the most useful access features are written as features in DATE_TIME (days, seconds and their associated duration date_duration and time_duration).

Comparison

Instances of DATE_TIME are totally ordered (the class inherit from ABSOLUTE). Functions <, + > and >= are available. Function is_equal must be used to test equality, = will compare references.

Measurement

Function duration gathers functions duration from the attributes time and date. The result is an instance of DATE_TIME_DURATION.

Element change

It is possible to change reference of time and date with the features set_time and set_date. To change only one element (for example hour), features from TIME or DATE have to be used.

Operations

Addition of hours, minutes and seconds are available directly in the class. The reason is that adding one second may have a consequence on the date. Using second_add from TIME is also possible but the date will not be mod-

ified in the case time makes a cycle. It is of course the same for minute and hour. day_add is also available directly since it is frequently used within the class.

Features + and add take an instance of DATE_TIME_DURATION in arguments. The date duration is added first then the time duration. Adding the time duration first would have yield some different result as in this example: the current date is August 30th 23:59:59. The duration to add is one month and one second. Feature add returns October 1st 0:0:0, whereas adding the second first would return September 30th 0:0:0! The same difference occurs with leap years.

Feature relative_duration and definite_duration returns the duration between the current date (with time) and the argument. The first one returns a result which is canonical (see definitions below), while the second one returns a result definite but may be not canonical. It is the same notion than in DATE.

1.4. DATE and DATE_TIME

Another way to process would have been to make DATE_TIME inherit from DATE, as long as DATE_TIME is a DATE, with more precision. The choice was to have a client relation between them. Otherwise DATE should have known the existence of DATE_TIME, and many assignment attempts would have been useful in features such as infix +. So DATE_TIME is client of DATE.

However, it could be useful to mix instances of DATE of DATE_TIME. As DATE_TIME is client of DATE with its attribute date, it is easy to get only the date from instances of DATE_TIME. On the other way features are available to convert objects from DATE to DATE_TIME. In class DATE, feature to_date_time builds an instance of DATE_TIME with the origin of time (0,0,0). In the class DATE_TIME, the creation procedure make_by_date has the same effect. (The same feature exists for duration, replacing origin by zero).

2. DURATION

TIME_DURATION, DATE_DURATION AND DATE_TIME_DURATION

The classes dealing with duration inherit DURATION, which inherits GROUP_ELEMENT and PART_COMPARABLE. An instance of TIME_DURATION, DATE_DURATION or DATE_TIME_DURATION is an element of a group, i.e. there is a zero and addition operations (infix +, infix -, prefix + and prefix -). Duration is used as an amount of time, without link to an origin. It may be added to the respective absolute notion (time + time_duration is possible, not time + date_time_duration nor date_time + time_duration...see classes TIME, DATE and DATE_TIME).

Attributes are allowed to take negative values or values which do not stand in the usual range (hour = -1, minute = 75, day = 40...). However, features are available in each class to convert instances into the usual format: functions canonical and to_canonical are present in each class. An instance is canonical (canonical = True) if its attributes stand into the usual range. For example, an instance of TIME_DURATION such as 12:-10:60 is not canonical. to_canonical will return 11:51:0. In DATE_DURATION and DATE_TIME_DURATION, these features are also present.

The order is partially implemented. TIME_DURATION has a complete order whereas DATE_DURATION and DATE_TIME_DURATION are more specific.

2.1. TIME_DURATION

Creation

Two ways are possible: by choosing the value of each attributes hour, minute and second (feature make), or by giving an amount of seconds (make_by_seconds). Any integer value is accepted. It is possible to create a duration with 1 hour and -60 minutes.

Access

Zero is a once feature with 0 hour, 0 minute and 0 second. The total amount of second of the current duration is the result of feature seconds_count.

Comparison

Instances of TIME_DURATION may be compared easily. The order is the order of the respective total amount of second. 1:-40:0 is less than 0:0:1800, etc... Functions <, >, <= and >= are available. Is_equal tests equality, = will compare references.

Element change

Set directly hour, minute and second with set_hour, set_minute and set_second. Arguments do not need to satisfy any range rule.

Operations

- Adding hours, minutes and seconds with features hour_add, minute_add and second_add.
- TIME_DURATION inherits from GROUP_ELEMENT. infix and prefix +, infix and prefix - are available to compose instances of each other.

Conversion

Two features ensure a link with the notion of day: to_days returns the number of days equivalent to the current duration. For example, a duration such as 23:60:0 is equivalent to one day. For negative duration, the result is never 0. -1 hour is equivalent to -1 day (i.e. the result of the function is - 1). To_days is associated with time_modulo_day. This second function returns an instance of TIME_DURATION. The result represents the difference between the current duration and the number of days yielded by to_days. It implies that the result is always positive and less than one day.

For example, the current duration is 25:70:600. to_days will returns 1 (one day) and time_modulo_day will return 2:20:0:. If the current duration is negative: -23:-80:300, to_days will return -2 (minus two days) and time_modulo_day will return 23:45:0.

Durations may be canonical or not canonical (BOOLEAN canonical). That means the features hour, minute and second are included in a particular range, or not. An instance of TIME_DURATION is canonical if:
- in the case of a positive duration (> zero), all of the three features have to be positive or 0, minute and second less than 60.
- in the case of a negative duration (< zero), all of the three features have to be negative or 0, minute and second strictly greater than -60. The function canonical tests if the duration is canonical or not, the function to_canonical yields a new duration equivalent to the current one and canonical.

2.2. DATE_DURATION

Dealing with the gregorian calendar is not so easy because of irregularities. A duration of one month may be equal to 28 up to 31 days, depending on the current date! On the other hand, it could be useful to deal with precise duration. This point leads to an original design of the class: A separation is made between two kinds of instances. The definite ones and the relative ones. The function definite which returns a BOOLEAN, is true for definite duration and false otherwise. An instance is definite if and only if its attributes month and year are 0. Then only the number of days is used. Relative (non definite) durations have their attributes year, month and day meaningful but it is then impossible to compare them to each other (is one month greater than 30 days?, is one year greater than 365 days?). The main difference appears when a duration is added to a date. In the case of a definite duration, there is no ambiguity. A given number of days are added to the date, taking care of the calendar. In the other case, the result is relative to the origin date. For example, a one month duration may be equal to 28 days if the date is in February or 31 days if the date is in August. A duration becomes definite when its attributes year and month become 0. However it is possible to deal with instances of DATE_DURATION without taking care of this distinction.

2.2.1. Relative date_duration.

Relative duration can not be compared with any other durations (including zero). The reason is simple. It is not possible to say if 30 days are less than 1 month: it depends on the date: it is true in August (in a 31 days month) and it is false in February.

If feature > (or <, + + is called with at least one non definite member (the current instance or the argument), the result will be always False. We may only know if two durations are equal, with the feature is_equal. It compares field by field the two durations. When adding a relative date_duration to a date, the years and the months are added first, then the date may be cut (June 31 -> June 30) and finally the days are added. For example, if one month is added to the date August 31st, the result is September 30th.

Nevertheless there is a way to compare relative durations: a relative date_duration may be canonical. It means that the duration has its attributes month and day in a fixed range. month must be between 1 and 12, and day larger than 1 and less than a value between 27 and 30. This value is fixed simply: (in the case of a positive duration) when setting day to 0 and adding one more month, the addition of the start date and this new duration must yield a date strictly after the final date (yielded by adding date and tested duration). For example is 0/0/30 (i.e. 0 year, 0 month and 30 days) canonical?

- If the origin date is 01/15 (15th of January), the final date is 02/14. We can not convert 30 days into 1 month in this case. The duration is canonical.
- If the origin date is 04/15 (15th of april), the final date is 05/15. The duration is not canonical since it is possible to convert days into 1 month.

The origin date is used to determine whether the duration is positive or not. If the final date is after the origin date the duration is positive, otherwise it is negative. That does not mean we can compare it to zero, that is only used to determine the sign of the canonical standard. If the duration is negative, it is canonical only if all the attributes are negative.

A way to compare two relative durations is to make them canonical from the same date, and then to compare the fields. It is the same as adding the durations to the same date, and to compare the final dates to each other.

2.2.2. Definite date_duration.

Definite durations are characterized by the attribute day. Whenever a duration has its attributes year and

month equal to 0, this duration is then definite. On the other hand, if one of these two attributes is not 0, the duration is not definite anymore.

The number of days between an origin date and the result of (date + duration) does not depend on the origin date. It is possible to compare definite date_duration to each other. The order is the one of day.

A definite duration may be canonical or not. It is canonical if the number of day is small enough.

2.2.3. General description.

Creation

Two creation features are available: make takes three arguments (year, month and day). If year and month are null, the duration will be definite; make_by_days takes only the number of day. The duration is automatically definite.

Comparison

Features <, >, <= and >= are available. If both instances are definite, numbers of days are compared. If one of them is non definite, the result is False.

Element change

Features set_day, set_month and set_year are available to set one of these three attributes day, month, year.

Operation

- Add years, months and days with features year_add, month_add and day_add.
- DATE_DURATION inherits from GROUP_ELEMENT. infix and prefix +, infix and prefix - are available to compose instances of each other.

Conversion

- to_canonical is used to get a new duration equivalent to the current one and canonical. It needs an argument from class DATE, which is the origin of calculations.
- to_definite is used to get a new duration equivalent to the current one and definite. As with the previous feature, one argument is needed. - to_date_time is used to get an instance of DATE_TIME_DURATION. It will have the same date of the current duration and time set to zero.

2.3. DATE_TIME_DURATION

DATE_TIME_DURATION is client of DATE_DURATION and TIME_DURATION. Most of the common features described in DATE_DURATION are present in the class. The class deals with its attributes date and time in the same way as DATE_TIME.

There are, as in DATE_DURATION, definite and non definite durations. It is the date part which gives the definite / non definite status. Features canonical and to_canonical are present in DATE_TIME_DURATION. They have

to deal with the attributes time.

Creation

There are still several ways to create an instance:
- by choosing values for all the attributes of date and time (make). - by choosing a value for day and values for all the attributes of time. The instance is then definite (make_definite).
- by gathering an instance of DATE with an instance of TIME (make_by_date_time). This feature copies the references of its arguments, so that if the time (or the date) is changed, the instance previously initialized will be also changed. If this effect has to be avoided, the use of clones is required.
- by encapsulating an instance of DATE (make_by_date). The attribute time is set to zero, i.e. 0:0:0. The attribute date is set with the same reference than the argument.

Access

Seconds_count is the amount of seconds of the time part only. To get the total amount of seconds of the current duration, first shift it to a definite duration, then multiply day by the number of seconds in day and add to it seconds_count. Take care that the duration is not more than 68 years. If it is, the number of seconds will be larger than 2 billion, which is the upper limit for INTEGER (4 bytes).

Comparison

The rules are the same than those for DATE_DURATION. Features <, >, <= and >= are available. If both instances are definite, numbers of days are compared. If one of them is non definite, the result is False.

Element change

It is possible to change reference of time and date with the features set_time and set_date. To change only one element (for example hour), features from TIME_DURATION or DATE_DURATION have to be used.

Operation

- DATE_TIME_DURATION inherits from GROUP_ELEMENT. infix and prefix +, infix and prefix - are available to compose instances to each other.
- Only day_add is present. To add only one element, features from TIME_DURATION or DATE_DURATION have to be used.

Conversion

- canonical and to_canonical are available in the class. To be canonical an instance of the class must have its attributes time and date canonical. Then time must have the same sign than the one of the current duration. The sign of the current instance is determined by adding it to the argument (from DATE_TIME). That will yield a final date. If this final date is after the origin (= the argument), the current duration is considered positive. Otherwise, it is considered negative. Finally time must be less than one day (if positive) or more than minus one day (if negative). To_canonical returns a duration equivalent to the current one (for the argument) and canonical.
- time_to_canonical looks like to_canonical but focuses mainly on time. It requires a definite duration so that it is possible to compare it to zero. It yields a definite duration equivalent to the current one with a canonical time. hour

is then cut so that it stands in the range of one day (0 to 23 if positive and -23 to 0 if negative). The attributes day is also modified to keep the result equivalent to the current duration. time_to_canonical does not need any argument because only time and day are modified.

3. INTERVAL

Class INTERVAL deals with intervals between two instances of the same class which conform to ABSOLUTE (DATE, TIME, DATE_TIME). The notions of interval is directly linked with the notion of order. The start_bound must be before the end_bound.

Creation

The features make, set_start_bound and set_end_bound take clones of their arguments, so that if these arguments are changed, the interval previously created is not.
It would have been possible to create intervals with references to date or time, but a modification of the dates would have been effective in the interval so that only the invariant would have been able to check if the start_bound is still before the end_bound.

Interval measurement

The measure of intervals is made by duration: the result is an instance of the class DURATION. However, as DURATION is the common parent of TIME_DURATION, DATE_DURATION and DATE_TIME_DURATION, it does not have many features available. Some features in class TIME, DATE, DATE_TIME return the same result and are more efficient to use. DURATION has to be use as the last solution.

Comparison

It includes intersection, inclusion and a special comparison.
- is_equal is present and compare values, not references.
- Feature intersects returns the mathematical result of the intersection of two intervals.
- is_strict_included_by, strict_includes, is_included_by and includes are connected to the same notion of inclusion.
- <, + > and >= use a special rule to compare intervals. int1 < int2 is true if int1 starts and ends strictly before int2. The other features use the same rule and is_equal if needed.
- overlaps looks like intersects but the argument has to be after the current interval. is_overlapped is the opposite.
- meets and is_met are used to test if two intervals have a common bound.

Status Report

The main part of the functions need an argument from the same generic type and return a BOOLEAN value. - empty tests if the bounds are equal. - has, strict_before, strict_after, before and after test the position of an element relatively to the current interval.

Element change

set_start_bound and set_end_bound are available to change the bounds.

Operations

Union and intersection are the mathematical functions. gather requires that two intervals meet each other and then yields the union.

4. Format

The aim of classes dealing with format is to get a printable representation of date (or time) which conforms to the local habits. It means that for each country and for each habit a different set of data has to be used. For this reason data and formatting features which use the data are completely separated in EiffelTime. This makes it easy to switch from one set of data to another.
Data are organized in classes which inherit from LOCALIZER. They are classified by their type: BOOLEAN, STRING, ARRAY [STRING] and INTEGER. Each piece of information is stored with a key and manipulated with it. Formatting features are organized in classes which inherit from FORMAT: FORMAT [TIME], FORMAT [DATE], FORMAT [DATE_TIME], they are clients of LOCALIZER.

4.1. Storing data

4.1.1. LOCALIZER.

The class is written so that all the common operations are available to manipulate data. Four types of data are available: BOOLEAN, STRING, ARRAY [STRING] and INTEGER. For each of them features are present. - to record data: record_boolean_value, record_integer_value, record_string_array_value and record_string_value. A value is stored with a key. If the key is already present, nothing happens. To force the system to record a new value, features force_boolean, force_integer, force_string and force_string_array are available, with the same signature.
- to test the presence of data: has_boolean_entry, has_integer_entry, has_string_array_entry and has_string_entry.
- to remove data: remove_boolean, remove_integer, remove_string and remove_string_array.
- to access data: boolean_value, integer_value, string_array_value and string_value. To keep the access process convenient, a default value is required for each access. This is to avoid the client of the class testing for each request. It is a little bit disturbing for arrays but much time is saved while accessing data.

LOCALIZER is not specific to EiffelTime. It will be useful for other libraries and users applications.

4.1.2. TIME_LOCALIZER.

The class represents a more specific type of data structure. It inherits from LOCALIZER. Invariants are present to ensure that data needed in classes FORMAT and its heirs are well defined.

4.2. Formatting date and time4.2.1. FORMAT.

The class is deferred and generic. It contains features to justify strings. Four options are available: left_justified, centered, right_justified and not_justified. The total width (width) has to be defined (generally during initialization). Function justify returns a new string whose length is equal to width, and which contains the string argument justified according to the current parameters. Only its heirs provide effective formatting feature: formatted. Then justify and formatted may be used together to provide a formatted and justified representation of a date (or time).

4.2.2. FORMAT [TIME] and FORMAT [DATE].

These classes have several common points.

- They are clients of LOCALIZER. LOCALIZER provides them data to format date or time. The creation procedure has its first argument which conforms to LOCALIZER. It is possible to change localizer with set_localizer.
- In these classes, feature formatted is effective and provides a formatted representation of the its argument. formatted includes many options (the attributes of the class).

4.2.3. FORMAT [DATE_TIME].

FORMAT [DATE_TIME] inherits from FORMAT. It is a client of FORMAT [TIME] and FORMAT [DATE]. An instance of this class gathers one instance of FORMAT [TIME] and one of FORMAT [DATE]. Several options are then available (which are not in FORMAT): - to have date before or after time with set_date_first and set_time_first.
- to justify the date and the time before they are formatted together with justify_date and justify_time.
- to change element such as the separator, or the attributes format_date and format_time (with set_separator_date_time, set_format_date and set_format_time). A special care has to be brought to the attribute width since it must keep larger or equal to the sum of the formatted date plus the formatted time plus the separator length.

5. More precision in TIME

TIME and TIME_DURATION are designed to deal with high precision in time. The only limit is the one from number representation.

The classes TIME and TIME_DURATION have an attribute fine_second (inherited from TIME_VALUE) which allows high precision. This attribute represents the number of seconds with fractions (it is an instance of DOUBLE). From this attribute are calculated second and fractional_second (which are functions): second is the truncated-to-integer part and fractional_second is the difference between the two previous one, so that the sum of second and fractional_second is always equal to fine_second (see invariant in TIME_VALUE).

As a result of this, when fine_second is positive (3.55 for example), second and fractional_second are also positive (3 and 0.55). When fine_second is negative (- 3.55 for example), second and fractional_second are also negative (- 3 and - 0.55).

Manipulation on second and fractional_second are in fact always made through fine_second. Users who do not want to deal with precision do not need to care about this.
Features dealing with fine_second and fractional_second are described here.

Creation (common to TIME and to TIME_DURATION)

- make_fine looks like make but it takes a DOUBLE for its third argument (instead of an INTEGER). fine_second is then set to this value.

- make_by_fine_seconds looks like make_by_seconds but it takes a DOUBLE for argument (instead of an INTEGER).Comparison (common)

There are no new features. The same ones are available since they are written to deal with precision. It is possible to compare two instances, one with precision and the other one without.

Measurement and access

In TIME:
- fine_seconds looks like seconds but it contains fractions.

In TIME_DURATION:
- fine_seconds_count looks like seconds_count but it contains fractions.

Element change

- set_fine_second allows to set directly fine_second to a DOUBLE. In TIME, a precondition requires that the argument stands in the allowed range.
- set_fractionals allows to set directly fractional_second to a DOUBLE. In TIME a precondition requires that the argument is positive and smaller than one. In TIME_DURATION the precondition is particular: it is not allowed to have an argument with a different sign than second. Otherwise, as long as fractional_second and second are linked to fine_second, such a setting would mean that second is also changed and fractional_second set to a different value. For example if fine_second = 4.5 (then second = 4 and fractional_second = 0.5) and - 0.8 is the argument of set_fractionals.

The result of that would be fine_second = 3.2 (then second = 3 and fractional_second = 0.2). It is better to prohibit that.

Comment: feature set_second (from both TIME and TIME_DURATION) will cut down fractional_second to zero.

Operations

- fine_second_add looks like second_add but takes a DOUBLE for argument.

In TIME_DURATION:
- canonical and to_canonical deals already with precision. There is nothing different.

6. INTERFACES

6.1. Cluster: time

6.1.1. *ABSOLUTE*

indexing

description: "*absolute temporal notion*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:53 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

deferred class interface
ABSOLUTE

feature -- Access

origin: ***like*** *Current*
-- Place of start for recording objects
ensure
result_exists: *Result /= void*

feature -- Measurement

duration: *DURATION*
-- Lenght of the interval of time since *origin*

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* attached to an object of the same type
-- as current object and identical to it?
-- (from *COMPARABLE*)
require -- from *GENERAL*
other_not_void: *other /= void*
ensure -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*
ensure then -- from *COMPARABLE*
trichotomy: *Result* = (***not*** (*Current* < *other*) ***and not*** (*other* < *Current*))

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
current_if_not_smaller: *Current >= other* ***implies*** *Result = Current*;
other_if_smaller: *Current < other* ***implies*** *Result = other*

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
current_if_not_greater: *Current <= other* ***implies*** *Result = Current*;
other_if_greater: *Current > other* ***implies*** *Result = other*

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0;
-- if smaller, -1; if greater, 1
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
equal_zero: (*Result = 0*) = *is_equal* (*other*);
smaller_negative: (*Result = -1*) = (*Current < other*);
greater_positive: (*Result = 1*) = (*Current > other*)

infix "<" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current object before *other*?
require -- from *PART_COMPARABLE*

```
                other_exists: other /= void
            ensure  -- from COMPARABLE
                asymmetric: Result implies not (other < Current)

        infix "<=" (other: like Current): BOOLEAN
                -- Is current object less than or equal to other?
                -- (from COMPARABLE)
            require  -- from PART_COMPARABLE
                other_exists: other /= void
            ensure  -- from COMPARABLE
                definition: Result = ((Current < other) or is_equal (other))

        infix ">" (other: like Current): BOOLEAN
                -- Is current object greater than other?
                -- (from COMPARABLE)
            require  -- from PART_COMPARABLE
                other_exists: other /= void
            ensure  -- from COMPARABLE
                definition: Result = (other < Current)

        infix ">=" (other: like Current): BOOLEAN
                -- Is current object greater than or equal to other?
                -- (from COMPARABLE)
            require  -- from PART_COMPARABLE
                other_exists: other /= void
            ensure  -- from COMPARABLE
                definition: Result = (other <= Current)

feature  -- Basic operations

        relative_duration (other: like Current): DURATION
            require
                other_exists: other /= void
            ensure
                result_exists: Result /= void

        infix "-" (other: like Current): INTERVAL [like Current]
                -- Interval between current object and other
            require
                other_exists: other /= void;
                other_smaller_than_current: other <= Current
            ensure
                result_exists: Result /= void;
                result_set: Result.start_bound.is_equal (other) and then Result.end_bound.is_equal (Cur-
rent)

invariant

            -- from GENERAL
```

reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
-- from *COMPARABLE*
irreflexive_comparison: ***not*** (*Current* < *Current*);

end -- class *ABSOLUTE*

6.1.2. *DATE*

indexing
description: "*absolute date*"
status: "*See notice at end of class*"
date: "*$Date: 1998/04/01 17:37:08 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
DATE

creation

make (*y, m, d*: *INTEGER*)
-- Set *year*, *month* and *day* to *y, m, d* respectively.
require
month_large_enough: *m >= 1*;
month_small_enough: *m <= months_in_year*;
day_large_enough: *d >= 1*;
day_small_enough: *d <= days_in_i_th_month* (*m, y*);
year_small_enough: *y <= 65535*
ensure
year_set: *year = y*;
month_set: *month = m*;
day_set: *day = d*

make_month_day_year (*m, d, y*: *INTEGER*)
-- Set *month*, *day* and *year* to *m*, *d* and *y* respectively.
require
month_large_enough: *m >= 1*;
month_small_enough: *m <= months_in_year*;
day_large_enough: *d >= 1*;
day_small_enough: *d <= days_in_i_th_month* (*m, y*)
ensure
year_set: *year = y*;
month_set: *month = m*;
day_set: *day = d*

make_day_month_year (*d, m, y*: *INTEGER*)
-- Set *day, month* and *year* to *d, m* and *y* respectively.
ensure
year_set: *year = y*;

month_set: *month = m*;
day_set: *day = d*

make_now
-- Set the current object to today's date.

make_by_days (*n*: *INTEGER*)
-- Set the current date with the number of days *n* since *origin.*
ensure
days_set: *days = n*

make_from_string (*s*: *STRING*; *code*: *STRING*)
-- Initialise from a "standard" string of form
-- *code*
require
s_exists: *s /= void*;
c_exists: *code /= void*;
date_valid: *date_valid* (*s*, *code*)

make_by_compact_date (*c_d*: *INTEGER*)
-- Initialize from a *compact_date.*
require
c_d_not_void: *c_d /= void*;
c_d_valid: *compact_date_valid* (*c_d*)
ensure
compact_date_set: *compact_date = c_d*

feature -- Initialization

make (*y*, *m*, *d*: *INTEGER*)
-- Set *year*, *month* and *day* to *y*, *m*, *d* respectively.
require
month_large_enough: *m >= 1*;
month_small_enough: *m <= months_in_year*;
day_large_enough: *d >= 1*;
day_small_enough: *d <= days_in_i_th_month* (*m*, *y*);
year_small_enough: *y <= 65535*
ensure
year_set: *year = y*;
month_set: *month = m*;
day_set: *day = d*

make_by_compact_date (*c_d*: *INTEGER*)
-- Initialize from a *compact_date.*
require
c_d_not_void: *c_d /= void*;
c_d_valid: *compact_date_valid* (*c_d*)
ensure
compact_date_set: *compact_date = c_d*

make_by_days (*n*: *INTEGER*)
-- Set the current date with the number of days *n* since *origin*.
ensure
days_set: *days* = *n*

make_day_month_year (*d*, *m*, *y*: *INTEGER*)
-- Set *day*, *month* and *year* to *d*, *m* and *y* respectively.
ensure
year_set: *year* = *y*;
month_set: *month* = *m*;
day_set: *day* = *d*

make_from_string (*s*: *STRING*; *code*: *STRING*)
-- Initialise from a "standard" string of form
-- *code*
require
s_exists: *s* /= *void*;
c_exists: *code* /= *void*;
date_valid: *date_valid* (*s*, *code*)

make_from_string_default (*s*: *STRING*)
-- Initialise from a "standard" string of form
-- *date_default_format_string*
require
s_exists: *s* /= *void*;
date_valid: *date_valid* (*s*, *date_default_format_string*)

make_month_day_year (*m*, *d*, *y*: *INTEGER*)
-- Set *month*, *day* and *year* to *m*, *d* and *y* respectively.
require
month_large_enough: *m* >= *1*;
month_small_enough: *m* <= *months_in_year*;
day_large_enough: *d* >= *1*;
day_small_enough: *d* <= *days_in_i_th_month* (*m*, *y*)
ensure
year_set: *year* = *y*;
month_set: *month* = *m*;
day_set: *day* = *d*

make_now
-- Set the current object to today's date.

feature -- Access

compact_date: *INTEGER*
-- Day, month and year coded.
-- (from *DATE_VALUE*)

date_default_format_string: *STRING*
-- (from *DATE_CONSTANTS*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

day: *INTEGER*
-- Day of the current object
-- (from *DATE_VALUE*)

days_in_i_th_month (*i*, *y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
-- (from *DATE_CONSTANTS*)
require -- from *DATE_CONSTANTS*
i_large_enough: *i >= 1*;
i_small_enough: *i <= months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year
-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

month: *INTEGER*
-- Month of the current object
-- (from *DATE_VALUE*)

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

origin: *DATE*
-- Origin date
ensure -- from *ABSOLUTE*
result_exists: *Result* /= *void*

year: *INTEGER*
-- Year of the current object
-- (from *DATE_VALUE*)

feature -- Measurement

days: *INTEGER*
-- Number of days elapsed since *origin*
ensure
same_duration: *Result* = *duration.day*

duration: *DATE_DURATION*
-- Definite duration elapsed since *origin*
ensure
definite_result: *Result.definite*;
duration_set: ((*Current* - *origin*)*.duration*)*.is_equal* (*Result*)

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* attached to an object of the same type
-- as current object and identical to it?
-- (from *COMPARABLE*)
require -- *COMPARABLE*
precursor: True
require else -- from *GENERAL*
other_not_void: *other* /= *void*
ensure -- from *COMPARABLE*
trichotomy: *Result* = (***not*** (*Current* < *other*) ***and not*** (*other* < *Current*))
ensure then -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*
ensure then -- from *COMPARABLE*

trichotomy: *Result* = (***not*** (*Current* < *other*) ***and*** ***not*** (*other* < *Current*))

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other* /= *void*
ensure -- from *COMPARABLE*
current_if_not_smaller: *Current* >= *other* ***implies*** *Result* = *Current*;
other_if_smaller: *Current* < *other* ***implies*** *Result* = *other*

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other* /= *void*
ensure -- from *COMPARABLE*
current_if_not_greater: *Current* <= *other* ***implies*** *Result* = *Current*;
other_if_greater: *Current* > *other* ***implies*** *Result* = *other*

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0;
-- if smaller, -1; if greater, 1
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other* /= *void*
ensure -- from *COMPARABLE*
equal_zero: (*Result* = *0*) = *is_equal* (*other*);
smaller_negative: (*Result* = *-1*) = (*Current* < *other*);
greater_positive: (*Result* = *1*) = (*Current* > *other*)

infix "<" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current date before *other*?
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*
ensure -- from *COMPARABLE*
asymmetric: *Result* ***implies*** ***not*** (*other* < *Current*)

infix "<=" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (from *COMPARABLE*)
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*
ensure -- from *COMPARABLE*
definition: *Result* = ((*Current* < *other*) ***or*** *is_equal* (*other*))

infix ">" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (from *COMPARABLE*)

```
            require  -- from PART_COMPARABLE
                     other_exists: other /= void
            ensure   -- from COMPARABLE
                     definition: Result = (other < Current)

        infix  ">=" (other: like  Current): BOOLEAN
                     -- Is current object greater than or equal to other?
                     -- (from COMPARABLE)
            require  -- from PART_COMPARABLE
                     other_exists: other /= void
            ensure   -- from COMPARABLE
                     definition: Result = (other <= Current)

feature  -- Status report

        day_of_january_1st: INTEGER
                     -- Day of the week of january 1st of the current year
            ensure
                     day_of_the_week_definition: Result > 0 and then  Result < 8

        day_of_the_week: INTEGER
                     -- Number of day from the beginning of the week
                     -- sunday is 1, etc.., saturday is 7
            ensure
                     day_of_the_week_range: Result > 0 and then  Result < 8

        days_at_month: INTEGER
                     -- Number of days from the beginning of the year
                     -- until the beginning of the current month
            ensure
                     positive_result: Result >= 0

        days_from (y: INTEGER): INTEGER
                     -- Days between the current year and year y

        days_in_month: INTEGER
                     -- Number of days in the current month
            ensure
                     positive_result: Result > 0

        days_in_year: INTEGER
                     -- Number of days in the current year
            ensure
                     valid_result: (leap_year implies  Result = days_in_leap_year) and then  (not  leap_year
implies  Result = days_in_non_leap_year)

        leap_year: BOOLEAN
                     -- Is the current year a leap year?
```

week_of_year: *INTEGER*
-- Number of weeks from the beginning of the year
-- The first week of the year begins on the first sunday of the year
-- The week number before the first sunday of the year is 0
ensure
positive_result: *Result >= 0*;
result_small_enough: *Result < max_weeks_in_year*

year_day: *INTEGER*
-- Number of days from the beginning of the year
ensure
result_large_enough: *Result >= 1*;
result_small_enough: *Result <= days_in_year*

feature -- Element change

set_day (*d*: *INTEGER*)
-- Set *day* to *d*.
require
d_large_enough: *d >= 1*;
d_small_enough: *d <= days_in_month*
ensure
day_set: *day = d*

set_month (*m*: *INTEGER*)
-- Set *month* to *m*.
-- *day* must be small enough.
require
m_large_enough: *m >= 1*;
m_small_enough: *m <= months_in_year*;
d_small_enough: *day <= days_in_i_th_month* (*m*, *year*)
ensure
month_set: *month = m*

set_year (*y*: *INTEGER*)
-- Set *year* to *y*.
require
can_not_cut_29th_feb: *day <= days_in_i_th_month* (*month*, *y*)
ensure
year_set: *year = y*

feature -- Conversion

to_date_time: *DATE_TIME*
-- Date-time version, with a zero time component

feature -- Basic operations

add (*d*: *DATE_DURATION*)

-- Adds *d* to the current date.
-- if *d* is not definite, add years and months and then days.

day_add (*d*: *INTEGER*)
-- Add *d* days to the current date.
ensure
days_set: *days* = ***old*** *days* + *d*

day_back
-- Move to previous day.
ensure
days_set: *days* = ***old*** *days* - *1*

day_forth
-- Move to next day.
-- days is from the origin, day is current.
ensure
days_set: *days* = ***old*** *days* + *1*

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i* = *j* * *Result* + *mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result* >= *0*;
result_definition: *i* = *j* * *div* (*i*, *j*) + *Result*

month_add (*m*: *INTEGER*)
-- add *m* months to the current date.
-- Can move days backward.

month_back
-- Move to previous month.
-- Can move days backward if previous month has less days than the current month.

month_forth
-- Move to next month.
-- Can move days backward if next month has less days than the current month.

relative_duration (*other*: ***like*** *Current*): *DATE_DURATION*
-- Duration from *other* to the current date, expressed in year, month and day
require -- from *ABSOLUTE*

```
                other_exists: other /= void
        ensure  -- from ABSOLUTE
                result_exists: Result /= void
        ensure then
                exact_duration: (other + Result).is_equal (Current);
                canonical_duration: Result.canonical (other)

    year_add (y: INTEGER)
                -- Add y years to the current date.
                -- May cut the 29th february.
        ensure
                year_set: year = old year + y

    year_back
                -- Move to previous year.
                -- May cut the 29th february.
        ensure
                year_decreased: year = old year - 1

    year_forth
                -- Move to next year.
                -- May cut the 29th february.
        ensure
                year_increased: year = old year + 1

    year_month_add (y, m: INTEGER)
                -- Add y years and m months to the current date.
                -- Check the number of days after.

    infix "+" (d: DATE_DURATION): DATE
                -- Sum to current date the duration d
                -- if duration not define, add years and then months and then days.
        ensure
                result_exists: Result /= void;
                definite_set: d.definite implies (Result - Current).duration.is_equal (d)

    infix "-" (other: like Current): INTERVAL [like Current]
                -- Interval between current object and other
                -- (from ABSOLUTE)
        require  -- from ABSOLUTE
                other_exists: other /= void;
                other_smaller_than_current: other <= Current
        ensure  -- from ABSOLUTE
                result_exists: Result /= void;
                result_set: Result.start_bound.is_equal (other) and then Result.end_bound.is_equal (Cur-
rent)

feature -- Output
```

formatted_out (*s*: *STRING*): *STRING*
-- printable representation of *Current* with "standard"
-- Form: *s*
require
s_exists: *s /= void*

out: *STRING*
-- printable representation of *Current* with "standard"
-- Form: *date_default_format_string*

feature -- Preconditions

compact_date_valid (*c_d*: *INTEGER*): *BOOLEAN*
require
c_d_not_void: *c_d /= void*

date_valid (*s*: *STRING*; *code_string*: *STRING*): *BOOLEAN*
-- Is the code_string enough precise
-- To create an instance of type DATE
-- And does the string *s* correspond to *code_string*?
require
s_exists: *s /= void*;
code_exists: *code_string /= void*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
day_large_enough: *day >= 1*;
day_small_enough: *day <= days_in_month*;
month_large_enough: *month >= 1*;
month_small_enough: *month <= months_in_year*;
year_small_enough: *year <= 65535*;
-- from *COMPARABLE*
irreflexive_comparison: ***not*** (*Current* < *Current*);

end -- class *DATE*

6.1.3. *DATE_CONSTANTS*

indexing
description: "*universal constants about dates*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:54 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
DATE_CONSTANTS

feature -- Access

date_default_format_string: *STRING*

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

days_in_i_th_month (*i*, *y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
require
i_large_enough: *i >= 1*;
i_small_enough: *i <= months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week

days_text: *ARRAY* [*STRING*]

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?

long_days_text: *ARRAY* [*STRING*]

long_months_text: *ARRAY* [*STRING*]

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year

months_text: *ARRAY* [*STRING*]

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)

```
            ensure  -- from TIME_UTILITY
                    result_definition: i = j * Result + mod (i, j)

        mod (i, j: INTEGER): INTEGER
                    -- (i \\ j) if i positive
                    -- (i \\ j + j) if i negative
                    -- (from TIME_UTILITY)
            ensure  -- from TIME_UTILITY
                    positive_result: Result >= 0;
                    result_definition: i = j * div (i, j) + Result

invariant

            -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);

end  -- class DATE_CONSTANTS
```

6.1.4. *DATE_DURATION*

```
indexing
        description: "duration expressed in date"
        status: "See notice at end of class"
        date: "$Date: 1998/03/10 16:59:54 $"
        revision: "$Revision: 4.2 $"
        access: date, time

class  interface
        DATE_DURATION

creation

        make (y, m, d: INTEGER)
                    -- Set year, month and day to y, m and d respectively.
            ensure
                    year_set: year = y;
                    month_set: month = m;
                    day_set: day = d

        make_by_days (d: INTEGER)
                    -- Set day to d.
                    -- The duration is definite
            ensure
                    day_set: day = d;
                    definite_duration: definite

feature  -- Initialization

        make (y, m, d: INTEGER)
```

-- Set *year*, *month* and *day* to *y*, *m* and *d* respectively.
ensure
year_set: *year = y*;
month_set: *month = m*;
day_set: *day = d*

make_by_days (*d*: *INTEGER*)
-- Set *day* to *d*.
-- The duration is definite
ensure
day_set: *day = d*;
definite_duration: *definite*

feature -- Access

compact_date: *INTEGER*
-- Day, month and year coded.
-- (from *DATE_VALUE*)

date_default_format_string: *STRING*
-- (from *DATE_CONSTANTS*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

days_in_i_th_month (*i*, *y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
-- (from *DATE_CONSTANTS*)
require -- from *DATE_CONSTANTS*
i_large_enough: *i >= 1*;
i_small_enough: *i <= months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year
-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

zero: *DATE_DURATION*
-- Neutral element for "+" and "-"
-- It is a definite duration
ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Are the current object and *other* equal?
require -- from *GENERAL*
other_not_void: *other* /= *void*
ensure -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*
ensure then
result_definition: *Result* = (*year* = *other.year* ***and then*** *month* = *month* ***and then*** *day* = *other.day*)

infix "<" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current object smaller than *other*?
-- It is impossible to compare not definite duration.
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*
ensure
definite_duration: (*definite* ***and then*** *other.definite*) ***implies*** *Result* = (*day* < *other.day*);
non_definite_duration: (***not*** *definite* ***or else*** ***not*** *other.definite*) ***implies*** *Result* = ***false***

```
        infix "<=" (other: like Current): BOOLEAN
                        -- Is current object less than or equal to other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

        infix ">" (other: like Current): BOOLEAN
                        -- Is current object greater than other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

        infix ">=" (other: like Current): BOOLEAN
                        -- Is current object greater than or equal to other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

feature -- Status report

        canonical (date: DATE): BOOLEAN
                        -- Is duration expressed minimally for adding to date, i.e.
                        --      If addition will yield a date after date, then:
                        --              year positive,
                        --              month between 0 and Months_in_year - 1,
                        --              day between 0 and (number of days of the month before the yielded) - 1?
                        --      If addition will yield a date before date, then:
                        --              year negative,
                        --              month between 1 - Months_in_year and 0,
                        --              day between (number of days of the month before the yielded) and 0?
                require
                        date_exist: date /= void

        definite: BOOLEAN
                        -- Is the duration is independant of the date
                        -- on which it applies? (use of day only)?
                        -- or not (use of year, month and day)?
                ensure
                        result_definition: Result = ((year = 0) and then (month = 0))

feature -- Conversion

        to_canonical (start_date: DATE): like Current
                        -- A new duration, equivalent to current one
                        -- and canonical for date
                ensure
                        canonical_result: Result.canonical (start_date);
                        duration_not_changed: (start_date + Current).is_equal (start_date + Result)
```

to_date_time: *DATE_TIME_DURATION*
-- Date-time version, with a zero time component
ensure
result_exists: *Result* /= *void*;
year_set: *Result.year* = *year*;
month_set: *Result.month* = *month*;
day_set: *Result.day* = *day*

to_definite (*date*: *DATE*)
-- Make current duration definite.
require
date_exists: *date* /= *void*
ensure
definite_result: *definite*

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i* = *j* * *Result* + *mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result* >= *0*;
result_definition: *i* = *j* * *div* (*i*, *j*) + *Result*

feature -- Attribute

day: *INTEGER*

month: *INTEGER*

year: *INTEGER*

feature -- Element Change

day_add (*d*: *INTEGER*)
-- Add *d* days to *Current*.
ensure
day_set: *day* = ***old*** *day* + *d*

month_add (*m*: *INTEGER*)

```
                -- Add m months to Current.
        ensure
                month_set: month = old month + m

    set_day (d: INTEGER)
                -- Set day to d.
        ensure
                day_set: day = d

    set_month (m: INTEGER)
                -- Set month to m.
        ensure
                month_set: month = m

    set_year (y: INTEGER)
                -- Set year to y.
        ensure
                year_set: year = y

    year_add (y: INTEGER)
                -- Add y years to Current.
        ensure
                year_set: year = old year + y

feature -- basic operation

    infix "+" (other: like Current): like Current
                -- Sum of current object with other
        require -- from GROUP_ELEMENT
                other_exists: other /= void
        ensure -- from GROUP_ELEMENT
                result_exists: Result /= void;
                commutative: Result.is_equal (other + Current)

    prefix "+ ": like Current
                -- Unary plus
        ensure -- from GROUP_ELEMENT
                result_exists: Result /= void;
                result_definition: Result.is_equal (Current)

    infix "-" (other: like Current): like Current
                -- Difference with other
        require -- from GROUP_ELEMENT
                other_exists: other /= void
        ensure -- from GROUP_ELEMENT
                result_exists: Result /= void

    prefix "- ": like Current
                -- Unary minus
```

ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*;
result_definition: (*Result* + *Current*)**.***is_equal* (*zero*)

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
-- from *GROUP_ELEMENT*
neutral_addition: *Current***.***is_equal* (*Current* + *zero*);
self_subtraction: *zero***.***is_equal* (*Current* - *Current*);

end -- class *DATE_DURATION*

6.1.5. *DATE_SET*

indexing
description: "*Sets of compactly coded dates*"
date: "*$Date: 1998/6/5 10:46 AM $*"
revision: "*$Revision: 4.3$*"

class interface
DATE_SET

creation

make (*n*: *INTEGER*)
-- Create structure for initial
-- estimate of *n* dates.
require
n_not_void: *n* /= *void*

feature -- Initialization

make_array (*minindex*, *maxindex*: *INTEGER*)
-- Allocate array; set index interval to
-- *minindex* .. *maxindex*; set all values to default.
-- (Make array empty if *minindex* = *maxindex* + 1).
-- (from *ARRAY*)
require -- from *ARRAY*
valid_indices: *minindex* <= *maxindex* ***or*** (*minindex* = *maxindex* + *1*)
ensure -- from *ARRAY*
lower = *minindex*;
upper = *maxindex*

make_from_array (*a*: *ARRAY* [*INTEGER*])
-- Initialize from the items of *a*.
-- (Useful in proper descendants of class *ARRAY,*
-- to initialize an array-like object from a manifest array.)

```
                -- (from ARRAY)
        require -- from ARRAY
                array_exists: a /= void

    setup (other: like Current)
                -- Perform actions on a freshly created object so that
                -- the contents of other can be safely copied onto it.
                -- (from ARRAY)
        ensure -- from GENERAL
                consistent (other)

feature -- Access

    area: SPECIAL [INTEGER]
                -- Special data zone
                -- (from TO_SPECIAL)

    entry (i: INTEGER): INTEGER
                -- Entry at index i, if in index interval
                -- Was declared in ARRAY as synonym of item, @ and entry.
                -- (from ARRAY)

    has (v: INTEGER): BOOLEAN
                -- Does v appear in array?
                -- (Reference or object equality,
                -- based on object_comparison.)
                -- (from ARRAY)
        ensure -- from CONTAINER
                not_found_in_empty: Result implies not empty

    item (i: INTEGER): DATE
                -- Element at index i
        require
                i_not_void: i /= void

    frozen item_array (i: INTEGER): INTEGER
                -- Entry at index i, if in index interval
                -- Was declared in ARRAY as synonym of item, @ and entry.
                -- (from ARRAY)
        require -- from TABLE
                valid_key: valid_index (k)

    last: INTEGER
                -- Index of the last element inserted

    frozen infix "@" (i: INTEGER): INTEGER
                -- Entry at index i, if in index interval
                -- Was declared in ARRAY as synonym of item, @ and entry.
                -- (from ARRAY)
```

require -- from *TABLE*
valid_key: *valid_index* (*k*)

feature -- Measurement

additional_space: *INTEGER*
-- Proposed number of additional items
-- (from *RESIZABLE*)
ensure -- from *RESIZABLE*
at_least_one: *Result >= 1*

capacity: *INTEGER*
-- Number of available indices
-- Was declared in ARRAY as synonym of *count* and *capacity.*
-- (from *ARRAY*)

count: *INTEGER*
-- Number of available indices
-- Was declared in ARRAY as synonym of *count* and *capacity.*
-- (from *ARRAY*)

Growth_percentage: *INTEGER* **is** *50*
-- Percentage by which structure will grow automatically
-- (from *RESIZABLE*)

lower: *INTEGER*
-- Minimum index
-- (from *ARRAY*)

Minimal_increase: *INTEGER* **is** *5*
-- Minimal number of additional items
-- (from *RESIZABLE*)

occurrences (*v*: *INTEGER*): *INTEGER*
-- Number of times *v* appears in structure
-- (from *ARRAY*)
ensure -- from *BAG*
non_negative_occurrences: *Result >= 0*

upper: *INTEGER*
-- Maximum index
-- (from *ARRAY*)

feature -- Comparison

is_equal (*other*: **like** *Current*): *BOOLEAN*
-- Is array made of the same items as *other*?
-- (from *ARRAY*)
require -- from *GENERAL*

other_not_void: *other* /= *void*

ensure -- from *GENERAL*

symmetric: *Result* ***implies*** *other*.*is_equal* (*Current*);

consistent: *standard_is_equal* (*other*) ***implies*** *Result*

feature -- Status report

all_cleared: *BOOLEAN*

-- Are all items set to default values?

-- (from *ARRAY*)

changeable_comparison_criterion: *BOOLEAN*

-- May *object_comparison* be changed?

-- (Answer: yes by default.)

-- (from *CONTAINER*)

consistent (*other*: ***like*** *Current*): *BOOLEAN*

-- Is object in a consistent state so that *other*

-- may be copied onto it? (Default answer: yes).

-- (from *ARRAY*)

empty: *BOOLEAN*

-- Is structure empty?

-- (from *FINITE*)

extendible: *BOOLEAN*

-- May items be added?

-- (Answer: no, although array may be resized.)

-- (from *ARRAY*)

full: *BOOLEAN*

-- Is structure filled to capacity? (Answer: yes)

-- (from *ARRAY*)

object_comparison: *BOOLEAN*

-- Must search operations use *equal* rather than =

-- for comparing references? (Default: no, use =.)

-- (from *CONTAINER*)

prunable: *BOOLEAN*

-- May items be removed? (Answer: no.)

-- (from *ARRAY*)

resizable: *BOOLEAN*

-- May *capacity* be changed? (Answer: yes.)

-- (from *RESIZABLE*)

valid_index (*i*: *INTEGER*): *BOOLEAN*

-- Is *i* within the bounds of the array?

-- (from *ARRAY*)

feature -- Status setting

compare_objects
-- Ensure that future search operations will use *equal*
-- rather than = for comparing references.
-- (from *CONTAINER*)
require -- from *CONTAINER*
changeable_comparison_criterion
ensure -- from *CONTAINER*
object_comparison

compare_references
-- Ensure that future search operations will use =
-- rather than *equal* for comparing references.
-- (from *CONTAINER*)
require -- from *CONTAINER*
changeable_comparison_criterion
ensure -- from *CONTAINER*
reference_comparison: ***not*** *object_comparison*

feature -- Element change

enter (*v*: ***like*** *item_array*; *i*: *INTEGER*)
-- Replace *i*-th entry, if in index interval, by *v*.
-- Was declared in ARRAY as synonym of *put* and *enter.*
-- (from *ARRAY*)

fill (*other*: *CONTAINER* [*INTEGER*])
-- Fill with as many items of *other* as possible.
-- The representations of *other* and current structure
-- need not be the same.
-- (from *COLLECTION*)
require -- from *COLLECTION*
other_not_void: *other* /= *void*;
extendible

force (*v*: ***like*** *item_array*; *i*: *INTEGER*)
-- Assign item *v* to *i*-th entry.
-- Always applicable: resize the array if *i* falls out of
-- currently defined bounds; preserve existing items.
-- (from *ARRAY*)
ensure -- from *ARRAY*
inserted: *item_array* (*i*) = *v*;
higher_count: *count* >= ***old*** *count*

put (*d*: *DATE*)
-- Insert *d*;

```
                -- Index will be given by last.
        require
                d_not_void: d /= void
        ensure
                inserted: item (last).is_equal (d)

    frozen  put_array (v: like  item_array; i: INTEGER)
                -- Replace i-th entry, if in index interval, by v.
                -- Was declared in ARRAY as synonym of put and enter.
                -- (from ARRAY)
        require  -- from TABLE
                valid_key: valid_index (k)
        ensure  -- from INDEXABLE
                insertion_done: item_array (k) = v

    subcopy (other: like  Current; start_pos, end_pos, index_pos: INTEGER)
                -- Copy items of other within bounds start_pos and end_pos
                -- to current array starting at index index_pos.
                -- (from ARRAY)
        require  -- from ARRAY
                other_not_void: other /= void;
                valid_start_pos: other.valid_index (start_pos);
                valid_end_pos: other.valid_index (end_pos);
                valid_bounds: (start_pos <= end_pos) or  (start_pos = end_pos + 1);
                valid_index_pos: valid_index (index_pos);
                enough_space: (upper - index_pos) >= (end_pos - start_pos)

feature  -- Removal

    clear_all
                -- Reset all items to default values.
                -- (from ARRAY)
        ensure  -- from ARRAY
                all_cleared: all_cleared

    prune_all (v: INTEGER)
                -- Remove all occurrences of v.
                -- (Reference or object equality,
                -- based on object_comparison.)
                -- (from COLLECTION)
        require  -- from COLLECTION
                prunable
        ensure  -- from COLLECTION
                no_more_occurrences: not  has (v)

    wipe_out
                -- Make array empty.
                -- (from ARRAY)
        require  -- from COLLECTION
```

prunable

ensure -- from *COLLECTION*

wiped_out: *empty*

feature -- Resizing

automatic_grow

-- Change the capacity to accommodate at least

-- *Growth_percentage* more items.

-- (from *RESIZABLE*)

ensure -- from *RESIZABLE*

increased_capacity: *capacity* >= ***old*** *capacity* + ***old*** *capacity* * *growth_percentage* // *100*

grow (*i*: *INTEGER*)

-- Change the capacity to at least *i*.

-- (from *ARRAY*)

ensure -- from *RESIZABLE*

new_capacity: *capacity* >= *i*

resize (*minindex*, *maxindex*: *INTEGER*)

-- Rearrange array so that it can accommodate

-- indices down to *minindex* and up to *maxindex*.

-- Do not lose any previously entered item.

-- (from *ARRAY*)

require -- from *ARRAY*

good_indices: *minindex* <= *maxindex*

ensure -- from *ARRAY*

no_low_lost: *lower* = *minindex.min* (***old*** *lower*);

no_high_lost: *upper* = *maxindex.max* (***old*** *upper*)

feature -- Conversion

linear_representation: *LINEAR* [*INTEGER*]

-- Representation as a linear structure

-- (from *ARRAY*)

to_c: *ANY*

-- Address of actual sequence of values,

-- for passing to external (non-Eiffel) routines.

-- (from *ARRAY*)

feature -- Duplication

copy (*other*: ***like*** *Current*)

-- Reinitialize by copying all the items of *other*.

-- (This is also used by *clone*.)

-- (from *ARRAY*)

require -- from *GENERAL*

other_not_void: *other* /= *void*;

type_identity: *same_type* (*other*)
ensure -- from *GENERAL*
is_equal: *is_equal* (*other*)
ensure then -- from *ARRAY*
equal_areas: *area.is_equal* (*other.area*)

subarray (*start_pos*, *end_pos*: *INTEGER*): ***like*** *Current*
-- Array made of items of current array within
-- bounds *start_pos* and *end_pos*.
-- (from *ARRAY*)
require -- from *ARRAY*
valid_start_pos: *valid_index* (*start_pos*);
valid_end_pos: *valid_index* (*end_pos*);
valid_bounds: (*start_pos* <= *end_pos*) ***or*** (*start_pos* = *end_pos* + *1*)
ensure -- from *ARRAY*
lower: *Result.lower* = *start_pos*;
upper: *Result.upper* = *end_pos*

feature -- Creation

make (*n*: *INTEGER*)
-- Create structure for initial
-- estimate of *n* dates.
require
n_not_void: *n* /= *void*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
last_non_negative: *last* >= *0*;
last_small_enough: *last* <= *count*;
-- from *ARRAY*
consistent_size: *capacity* = *upper* - *lower* + *1*;
non_negative_count: *count* >= *0*;
-- from *RESIZABLE*
increase_by_at_least_one: *minimal_increase* >= *1*;
-- from *BOUNDED*
valid_count: *count* <= *capacity*;
full_definition: *full* = (*count* = *capacity*);
-- from *FINITE*
empty_definition: *empty* = (*count* = *0*);
non_negative_count: *count* >= *0*;

end -- class *DATE_SET*

6.1.6. *DATE_TIME*

indexing

description: "*Absolute temporal value composed with a date and a time*"
status: "*See notice at end of class*"
date: "*$Date: 1998/04/01 17:37:08 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
DATE_TIME

creation

make (*y, mo, d, h, mi, s*: *INTEGER*)
-- Set *year, month day* to *y, mo, d*.
-- Set *hour, minute, second* to *h, mi, s*.
require
month_large_enough: *mo >= 1*;
month_small_enough: *mo <= months_in_year*;
day_large_enough: *d >= 1*;
day_small_enough: *d <= days_in_i_th_month* (*mo, y*);
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *mi >= 0*;
m_small_enough: *mi < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
year_set: *year = y*;
month_set: *month = mo*;
day_set: *day = d*;
hour_set: *hour = h*;
minute_set: *minute = mi*;
second_set: *second = s*

make_fine (*y, mo, d, h, mi*: *INTEGER*; *s*: *DOUBLE*)
-- Set *year, month day* to *y, mo, d*.
-- Set *hour, minute, second* to *h, m, s*.
require
month_large_enough: *mo >= 1*;
month_small_enough: *mo <= months_in_year*;
day_large_enough: *d >= 1*;
day_small_enough: *d <= days_in_i_th_month* (*mo, y*);
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *mi >= 0*;
m_small_enough: *mi < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
year_set: *year = y*;

month_set: *month* = *mo*;
day_set: *day* = *d*;
hour_set: *hour* = *h*;
minute_set: *minute* = *mi*;
second_set: *fine_second* = *s*

make_by_date_time (*d*: *DATE*; *t*: *TIME*)
-- Set *date* to *d* and *time* to *t*
require
d_exists: *d* /= *void*;
t_exists: *t* /= *void*
ensure
date_set: *date* = *d*;
time_set: *time* = *t*

make_by_date (*d*: *DATE*)
-- Set *date* to *d* and *time* to origin of time.
require
d_exists: *d* /= *void*
ensure
date_set: *date* = *d*;
time_set: *time*.*is_equal* (*time*.*origin*)

make_now
-- Get the date and the time from the system.

make_from_string (*s*: *STRING*; *code*: *STRING*)
-- Initialise from a "standard" string of form
-- *code*
require
s_exists: *s* /= *void*;
c_exists: *code* /= *void*;
date_time_valid: *date_time_valid* (*s*, *code*)

feature -- Initialization

make (*y, mo, d, h, mi, s*: *INTEGER*)
-- Set *year*, *month day* to *y*, *mo*, *d*.
-- Set *hour*, *minute*, *second* to *h*, *mi*, *s*.
require
month_large_enough: *mo* >= *1*;
month_small_enough: *mo* <= *months_in_year*;
day_large_enough: *d* >= *1*;
day_small_enough: *d* <= *days_in_i_th_month* (*mo*, *y*);
h_large_enough: *h* >= *0*;
h_small_enough: *h* < *hours_in_day*;
m_large_enough: *mi* >= *0*;
m_small_enough: *mi* < *minutes_in_hour*;
s_large_enough: *s* >= *0*;

s_small_enough: *s* < *seconds_in_minute*

ensure

year_set: *year* = *y*;
month_set: *month* = *mo*;
day_set: *day* = *d*;
hour_set: *hour* = *h*;
minute_set: *minute* = *mi*;
second_set: *second* = *s*

make_by_date (*d*: *DATE*)

-- Set *date* to *d* and *time* to origin of time.

require

d_exists: *d* /= *void*

ensure

date_set: *date* = *d*;
time_set: *time*.*is_equal* (*time*.*origin*)

make_by_date_time (*d*: *DATE*; *t*: *TIME*)

-- Set *date* to *d* and *time* to *t*

require

d_exists: *d* /= *void*;
t_exists: *t* /= *void*

ensure

date_set: *date* = *d*;
time_set: *time* = *t*

make_fine (*y, mo, d, h, mi*: *INTEGER*; *s*: *DOUBLE*)

-- Set *year, month day* to *y, mo, d*.
-- Set *hour, minute, second* to *h, m, s*.

require

month_large_enough: *mo* >= *1*;
month_small_enough: *mo* <= *months_in_year*;
day_large_enough: *d* >= *1*;
day_small_enough: *d* <= *days_in_i_th_month* (*mo, y*);
h_large_enough: *h* >= *0*;
h_small_enough: *h* < *hours_in_day*;
m_large_enough: *mi* >= *0*;
m_small_enough: *mi* < *minutes_in_hour*;
s_large_enough: *s* >= *0*;
s_small_enough: *s* < *seconds_in_minute*

ensure

year_set: *year* = *y*;
month_set: *month* = *mo*;
day_set: *day* = *d*;
hour_set: *hour* = *h*;
minute_set: *minute* = *mi*;
second_set: *fine_second* = *s*

make_from_string (*s*: *STRING*; *code*: *STRING*)

```
                -- Initialise from a "standard" string of form
                -- code
        require
                s_exists: s /= void;
                c_exists: code /= void;
                date_time_valid: date_time_valid (s, code)

    make_from_string_default (s: STRING)
                -- Initialise from a "standard" string of form
                -- default_format_string
        require
                s_exists: s /= void;
                date_time_valid: date_time_valid (s, default_format_string)

    make_now
                -- Get the date and the time from the system.

feature -- Access

    date: DATE
                -- Date of the current object

    date_default_format_string: STRING
                -- (from DATE_CONSTANTS)

    date_duration: DATE_DURATION
                -- Definite duration between origin of date and current date

    date_time_tools: DATE_TIME_TOOLS
                -- (from TIME_UTILITY)

    day: INTEGER
                -- Day of the current object
                -- (from DATE_TIME_VALUE)
        ensure  -- from DATE_TIME_VALUE
                same_day: Result = date.day

    days: INTEGER
                -- Number of days elapsed since origin

    days_in_i_th_month (i, y: INTEGER): INTEGER
                -- Number of days in the i th month at year y
                -- (from DATE_CONSTANTS)
        require -- from DATE_CONSTANTS
                i_large_enough: i >= 1;
                i_small_enough: i <= months_in_year

    Days_in_leap_year: INTEGER is  366
                -- Number of days in a leap year
```

-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

fine_second: *DOUBLE*
-- Representation of second with decimals
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_fine_second: *Result = time.fine_second*

fractionnal_second: *DOUBLE*
-- Decimal part of second
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_fractionnal: *Result = time.fractionnal_second*

hour: *INTEGER*
-- Hour of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_hour: *Result = time.hour*

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*

-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

minute: *INTEGER*
-- Minute of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_minute: *Result = time.minute*

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

month: *INTEGER*
-- Month of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_month: *Result = date.month*

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

origin: *DATE_TIME*
-- Origin date with origin time
ensure -- from *ABSOLUTE*
result_exists: *Result /= void*

second: *INTEGER*
-- Second of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_second: *Result = time.second*

seconds: *INTEGER*
-- Number of seconds elapsed from midnight of the current date

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*

```
                -- Number of seconds in a minute
                -- (from TIME_CONSTANTS)

        time: TIME
                -- Time of the current object

        time_default_format_string: STRING
                -- (from TIME_CONSTANTS)

        time_duration: TIME_DURATION
                -- Duration elapsed from midnight of the current date

        year: INTEGER
                -- Year of the current object
                -- (from DATE_TIME_VALUE)
            ensure  -- from DATE_TIME_VALUE
                same_year: Result = date.year

feature  -- Measurement

        duration: DATE_TIME_DURATION
                -- Definite duration elapsed from origin
            ensure
                definite_result: Result.definite

feature  -- Comparison

        is_equal (other: like Current): BOOLEAN
                -- Is the current object equal to other?
            require  -- COMPARABLE
                precursor: True
            require else  -- from GENERAL
                other_not_void: other /= void
            ensure  -- from COMPARABLE
                trichotomy: Result = (not (Current < other) and not (other < Current))
            ensure then  -- from GENERAL
                symmetric: Result implies other.is_equal (Current);
                consistent: standard_is_equal (other) implies Result

        max (other: like Current): like Current
                -- The greater of current object and other
                -- (from COMPARABLE)
            require  -- from COMPARABLE
                other_exists: other /= void
            ensure  -- from COMPARABLE
                current_if_not_smaller: Current >= other implies Result = Current;
                other_if_smaller: Current < other implies Result = other

        min (other: like Current): like Current
```

```
                -- The smaller of current object and other
                -- (from COMPARABLE)
        require  -- from COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                current_if_not_greater: Current <= other implies  Result = Current;
                other_if_greater: Current > other implies  Result = other

three_way_comparison (other: like  Current): INTEGER
                -- If current object equal to other, 0;
                -- if smaller, -1; if greater, 1
                -- (from COMPARABLE)
        require  -- from COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                equal_zero: (Result = 0) = is_equal (other);
                smaller_negative: (Result = -1) = (Current < other);
                greater_positive: (Result = 1) = (Current > other)

infix  "<" (other: like  Current): BOOLEAN
                -- Is the current object before other?
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                asymmetric: Result implies  not  (other < Current)

infix  "<=" (other: like  Current): BOOLEAN
                -- Is current object less than or equal to other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = ((Current < other) or  is_equal (other))

infix  ">" (other: like  Current): BOOLEAN
                -- Is current object greater than other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = (other < Current)

infix  ">=" (other: like  Current): BOOLEAN
                -- Is current object greater than or equal to other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = (other <= Current)
```

feature -- Basic operations

add (*dtd*: *DATE_TIME_DURATION*)
-- Adds *dtd* to the current object.

day_add (*d*: *INTEGER*)
-- Add *d* days to the current date.
ensure
days_set: *days* = ***old*** *days* + *d*

definite_duration (*other*: ***like*** *Current*): *DATE_TIME_DURATION*
-- Duration from *other* to the current date, expressed in year, month, day and time
require
other_exists: *other* /= *void*
ensure
definite_result: *Result**.**definite*

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i* = *j* * *Result* + *mod* (*i*, *j*)

fine_second_add (*s*: *DOUBLE*)
-- Add *s* seconds to the current object.
-- if *s* has decimals, *fractionnal_second* from *time* is modified.

hour_add (*h*: *INTEGER*)
-- Add *h* hours to the current time.

minute_add (*m*: *INTEGER*)
-- Add *m* minutes to the current time.

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result* >= *0*;
result_definition: *i* = *j* * *div* (*i*, *j*) + *Result*

relative_duration (*other*: ***like*** *Current*): *DATE_TIME_DURATION*
-- Duration from *other* to the current date, expressed in year, month, day and time
require -- from *ABSOLUTE*
other_exists: *other* /= *void*
ensure -- from *ABSOLUTE*
result_exists: *Result* /= *void*

second_add (*s*: *INTEGER*)
-- Add *s* seconds to the current time.

infix "+" (*d*: *DATE_TIME_DURATION*): ***like*** *Current*
-- Sum the current object with *d*
ensure
result_exists: *Result* /= *void*

infix "-" (*other*: ***like*** *Current*): *INTERVAL* [***like*** *Current*]
-- Interval between current object and *other*
-- (from *ABSOLUTE*)
require -- from *ABSOLUTE*
other_exists: *other* /= *void*;
other_smaller_than_current: *other* <= *Current*
ensure -- from *ABSOLUTE*
result_exists: *Result* /= *void*;
result_set: *Result***.***start_bound***.***is_equal* (*other*) ***and then*** *Result***.***end_bound***.***is_equal* (*Current*)

feature -- Element Change

copy (*other*: ***like*** *Current*)
-- set *date* and *time* with the *other* attributes.
require -- from *GENERAL*
other_not_void: *other* /= *void*;
type_identity: *same_type* (*other*)
ensure -- from *GENERAL*
is_equal: *is_equal* (*other*)

set_date (*d*: *DATE*)
-- Set *date* to *d*.
require
d_exists: *d* /= *void*
ensure
date_set: *date* = *d*

set_time (*t*: *TIME*)
-- Set *time* to *t*.
require
t_exists: *t* /= *void*
ensure
time_set: *time* = *t*

feature -- Output

formatted_out (*s*: *STRING*): *STRING*
-- Printable representation of the current object
-- With "standard" form: *s*

require

s_exists: *s* /= *void*

out: *STRING*

-- Printable representation of the current object
-- With "standard" form: *default_format_string*

feature -- Preconditions

date_time_valid (*s*: *STRING*; *code_string*: *STRING*): *BOOLEAN*

-- Is the code_string enough precise
-- To create an instance of type DATE_TIME
-- And does the string *s* correspond to *code_string*?

require

s_exists: *s* /= *void*;
code_exists: *code_string* /= *void*

invariant

-- from *GENERAL*

reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

-- from *COMPARABLE*

irreflexive_comparison: ***not*** (*Current* < *Current*);

-- from *DATE_TIME_VALUE*

date_exists: *date* /= *void*;
time_exists: *time* /= *void*;

end -- class *DATE_TIME*

6.1.7. *DATE_TIME_DURATION*

indexing

description: "*duration expressed in date and time*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:56 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface

DATE_TIME_DURATION

creation

make (*y, mo, d, h, mi, s*: *INTEGER*)

-- Set *year, month, day* to *y, mo, d* .
-- Set *hour, minute, second* to *h, mi, s*.

ensure

year_set: *year = y*;
month_set: *month = mo*;

```
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = mi;
                second_set: second = s

        make_definite (d, h, m, s: INTEGER)
                -- Set day to d.
                -- Set hour, minute, second to h, m, s.
            ensure
                definite_result: definite;
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = m;
                second_set: second = s

        make_fine (y, mo, d, h, mi: INTEGER; s: DOUBLE)
                -- set year, month, day to y, mo, d.
                -- set hour, minute, second to h, mi, s.
            ensure
                year_set: year = y;
                month_set: month = mo;
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = mi;
                fine_second_set: fine_second = s

        make_by_date_time (d: DATE_DURATION; t: TIME_DURATION)
                -- Set date to d and time to t.
            require
                d_exists: d /= void;
                t_exists: t /= void
            ensure
                date_set: date = d;
                time_set: time = t

        make_by_date (d: DATE_DURATION)
                -- Set date to d and time to zero.
            require
                d_exists: d /= void
            ensure
                date_set: date = d;
                time_set: time.is_equal (time.zero)

    feature -- Initialization

        make (y, mo, d, h, mi, s: INTEGER)
                -- Set year, month, day to y, mo, d .
                -- Set hour, minute, second to h, mi, s.
            ensure
```

```
                year_set: year = y;
                month_set: month = mo;
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = mi;
                second_set: second = s

        make_by_date (d: DATE_DURATION)
                -- Set date to d and time to zero.
            require
                d_exists: d /= void
            ensure
                date_set: date = d;
                time_set: time.is_equal (time.zero)

        make_by_date_time (d: DATE_DURATION; t: TIME_DURATION)
                -- Set date to d and time to t.
            require
                d_exists: d /= void;
                t_exists: t /= void
            ensure
                date_set: date = d;
                time_set: time = t

        make_definite (d, h, m, s: INTEGER)
                -- Set day to d.
                -- Set hour, minute, second to h, m, s.
            ensure
                definite_result: definite;
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = m;
                second_set: second = s

        make_fine (y, mo, d, h, mi: INTEGER; s: DOUBLE)
                -- set year, month, day to y, mo, d.
                -- set hour, minute, second to h, mi, s.
            ensure
                year_set: year = y;
                month_set: month = mo;
                day_set: day = d;
                hour_set: hour = h;
                minute_set: minute = mi;
                fine_second_set: fine_second = s

feature -- Access

        date: DATE_DURATION
                -- date part of the current duration
```

date_default_format_string: *STRING*
-- (from *DATE_CONSTANTS*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

day: *INTEGER*
-- Day of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_day: *Result = date.day*

days_in_i_th_month (*i*, *y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
-- (from *DATE_CONSTANTS*)
require -- from *DATE_CONSTANTS*
i_large_enough: *i >= 1*;
i_small_enough: *i <= months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year
-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

fine_second: *DOUBLE*
-- Representation of second with decimals
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_fine_second: *Result = time.fine_second*

fine_seconds_count: *DOUBLE*
-- Number of seconds and fractionnals of seconds of the current duration

fractionnal_second: *DOUBLE*
-- Decimal part of second

-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_fractionnal: *Result = time.time_value_frac_sec*

hour: *INTEGER*
-- Hour of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_hour: *Result = time.hour*

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

minute: *INTEGER*
-- Minute of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_minute: *Result = time.minute*

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

month: *INTEGER*
-- Month of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_month: *Result = date.month*

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]

-- (from *DATE_CONSTANTS*)

second: *INTEGER*
-- Second of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_second: *Result = time.second*

seconds_count: *INTEGER*
-- Total number of seconds of the current duration

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute
-- (from *TIME_CONSTANTS*)

time: *TIME_DURATION*
-- time part of current duration

time_default_format_string: *STRING*
-- (from *TIME_CONSTANTS*)

year: *INTEGER*
-- Year of the current object
-- (from *DATE_TIME_VALUE*)
ensure -- from *DATE_TIME_VALUE*
same_year: *Result = date.year*

zero: *DATE_TIME_DURATION*
-- Neutral element
ensure -- from *GROUP_ELEMENT*
result_exists: *Result /= void*

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Are the current duration an *other* equal?
require -- from *GENERAL*
other_not_void: *other /= void*
ensure -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*

```
        infix "<" (other: like Current): BOOLEAN
                        -- Is the current duration smaller than other?
                        -- False if either is not definite
                require -- from PART_COMPARABLE
                        other_exists: other /= void
                ensure
                        non_definite_result: not (definite and other.definite) implies Result = false

        infix "<=" (other: like Current): BOOLEAN
                        -- Is current object less than or equal to other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

        infix ">" (other: like Current): BOOLEAN
                        -- Is current object greater than other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

        infix ">=" (other: like Current): BOOLEAN
                        -- Is current object greater than or equal to other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

feature -- Status report

        canonical (start_date: DATE_TIME): BOOLEAN
                        -- Are the time and date parts of the same sign,
                        -- and both canonical?

        definite: BOOLEAN
                        -- Is this duration date-independent?
                        -- (True if it only uses day, not year and month)
                ensure
                        result_definition: Result = (year = 0 and then month = 0)

feature -- Conversion

        time_to_canonical: like Current
                        -- A new duration, equivalent to current one
                        -- but time is canonical and has the same sign as date
                require
                        definite_duration: definite
                ensure
                        time_canonical: Result.time.canonical;
                        same_sign: ((Result.date > date.zero) implies (Result.time >= time.zero)) and then
```

((*Result.date* < *date.zero*) ***implies*** (*Result.time* <= *time.zero*))

to_canonical (*start_date*: *DATE_TIME*): ***like*** *Current*
-- A new duration, equivalent to current one
-- and canonical for *start_date*
ensure
canonical_set: *Result.canonical* (*start_date*);
duration_not_changed: (*start_date* + *Current*)*.is_equal* (*start_date* + *Result*)

feature -- Basic operations

day_add (*d*: *INTEGER*)
-- Add *d* days to the current duration.
ensure
result_definition: *day* = ***old*** *day* + *d*

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i* = *j* * *Result* + *mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result* >= *0*;
result_definition: *i* = *j* * *div* (*i*, *j*) + *Result*

infix "+" (*other*: ***like*** *Current*): ***like*** *Current*
-- Sum with *other* (commutative)
require -- from *GROUP_ELEMENT*
other_exists: *other* /= *void*
ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*;
commutative: *Result.is_equal* (*other* + *Current*)

prefix "+ ": ***like*** *Current*
-- Unary plus
ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*;
result_definition: *Result.is_equal* (*Current*)

prefix "- ": ***like*** *Current*
-- Unary minus
ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*;

```
                result_definition: (Result + Current).is_equal (zero)

        infix "-" (other: like Current): like Current
                -- Difference with other
            require -- from GROUP_ELEMENT
                other_exists: other /= void
            ensure -- from GROUP_ELEMENT
                result_exists: Result /= void

feature -- Element Change

        set_date (d: DATE_DURATION)
                -- Set date to d.
            require
                d_exists: d /= void
            ensure
                date_set: date = d

        set_time (t: TIME_DURATION)
                -- Set time to t.
            require
                t_exists: time /= void
            ensure
                time_set: time = t

invariant

                -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);
                -- from DATE_TIME_VALUE
        date_exists: date /= void;
        time_exists: time /= void;
                -- from GROUP_ELEMENT
        neutral_addition: Current.is_equal (Current + zero);
        self_subtraction: zero.is_equal (Current - Current);

end -- class DATE_TIME_DURATION
```

6.1.8. *DATE_TIME_SET*

```
indexing
        description: "Sets of compactly coded date-time pairs"
        date: "$Date: 1998/6/5 10:46 AM $"
        revision: "$Revision: 4.3$"

class interface
        DATE_TIME_SET

creation
```

make (*n*: *INTEGER*)
-- Create structure for initial
-- estimate of *n* date-time pairs.
require
n_not_void: *n* /= *void*

feature -- Access

item (*i*: *INTEGER*): *DATE_TIME*
-- Element at index *i*
require
i_not_void: *i* /= *void*

last: *INTEGER*
-- Index of the last element inserted

feature -- Element change

put (*dt*: *DATE_TIME*)
-- Insert *dt*;
-- Index will be given by *last.*
require
dt_not_void: *dt* /= *void*

feature -- Creation

make (*n*: *INTEGER*)
-- Create structure for initial
-- estimate of *n* date-time pairs.
require
n_not_void: *n* /= *void*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
last_non_negative: *last >= 0*;

end -- class *DATE_TIME_SET*

6.1.9. *DATE_TIME_VALUE*

indexing
status: "*See notice at end of class*"
date: "*$Date: 1998/04/01 17:37:09 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
DATE_TIME_VALUE

creation
feature -- Access

date: *DATE_VALUE*
-- Date of the current object

date_default_format_string: *STRING*
-- (from *DATE_CONSTANTS*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

day: *INTEGER*
-- Day of the current object
ensure
same_day: *Result = date.day*

days_in_i_th_month (*i*, *y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
-- (from *DATE_CONSTANTS*)
require -- from *DATE_CONSTANTS*
i_large_enough: *i >= 1*;
i_small_enough: *i <= months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year
-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*
-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

fine_second: *DOUBLE*
-- Representation of second with decimals
ensure
same_fine_second: *Result = time.fine_second*

fractionnal_second: *DOUBLE*
-- Decimal part of second
ensure
same_fractionnal: *Result = time.fractionnal_second*

hour: *INTEGER*
-- Hour of the current object
ensure
same_hour: *Result = time.hour*

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

minute: *INTEGER*
-- Minute of the current object
ensure
same_minute: *Result = time.minute*

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

month: *INTEGER*
-- Month of the current object
ensure
same_month: *Result = date.month*

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

second: *INTEGER*
-- Second of the current object
ensure
same_second: *Result = time.second*

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute
-- (from *TIME_CONSTANTS*)

time: *TIME_VALUE*
-- Time of the current object

time_default_format_string: *STRING*
-- (from *TIME_CONSTANTS*)

year: *INTEGER*
-- Year of the current object
ensure
same_year: *Result = date.year*

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i = j * Result + mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i*, *j*) *+ Result*

invariant

-- from *GENERAL*

reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
date_exists: *date* /= *void*;
time_exists: *time* /= *void*;

end -- class *DATE_TIME_VALUE*

6.1.10. *DATE_VALUE*

indexing
description: "*value dealing with year, month and day*"
status: "*See notice at end of class*"
date: "*$Date: 1998/04/01 17:37:09 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
DATE_VALUE

creation
feature -- Access

compact_date: *INTEGER*
-- Day, month and year coded.

date_default_format_string: *STRING*
-- (from *DATE_CONSTANTS*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

day: *INTEGER*
-- Day of the current object

days_in_i_th_month (*i, y*: *INTEGER*): *INTEGER*
-- Number of days in the *i* th month at year *y*
-- (from *DATE_CONSTANTS*)
require -- from *DATE_CONSTANTS*
i_large_enough: *i* >= *1*;
i_small_enough: *i* <= *months_in_year*

Days_in_leap_year: *INTEGER* ***is*** *366*
-- Number of days in a leap year
-- (from *DATE_CONSTANTS*)

Days_in_non_leap_year: *INTEGER* ***is*** *365*
-- Number of days in a non-leap year
-- (from *DATE_CONSTANTS*)

Days_in_week: *INTEGER* ***is*** *7*

-- Number of days in a week
-- (from *DATE_CONSTANTS*)

days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

i_th_leap_year (*i*: *INTEGER*): *BOOLEAN*
-- Is the i-th year a leap year?
-- (from *DATE_CONSTANTS*)

long_days_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

long_months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

Max_weeks_in_year: *INTEGER* ***is*** *53*
-- Maximun number of weeks in a year
-- (from *DATE_CONSTANTS*)

month: *INTEGER*
-- Month of the current object

Months_in_year: *INTEGER* ***is*** *12*
-- Number of months in year
-- (from *DATE_CONSTANTS*)

months_text: *ARRAY* [*STRING*]
-- (from *DATE_CONSTANTS*)

year: *INTEGER*
-- Year of the current object

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i = j * Result + mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)

ensure -- from *TIME_UTILITY*
positive_result: *Result* >= *0*;
result_definition: *i* = *j* * *div* (*i*, *j*) + *Result*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_VALUE*

6.1.11. *DURATION*

indexing
description: "*duration of an interval of time*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:58 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

deferred class interface
DURATION

feature -- Access

zero: **like** *Current*
-- Neutral element for "+" and "-"
-- (from *GROUP_ELEMENT*)
ensure -- from *GROUP_ELEMENT*
result_exists: *Result* /= *void*

feature -- Comparison

infix "<" (*other*: **like** *Current*): *BOOLEAN*
-- Is current object less than *other*?
-- (from *PART_COMPARABLE*)
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*

infix "<=" (*other*: **like** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (from *PART_COMPARABLE*)
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*

infix ">" (*other*: **like** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (from *PART_COMPARABLE*)
require -- from *PART_COMPARABLE*

```
                    other_exists: other /= void

        infix  ">=" (other: like  Current): BOOLEAN
                    -- Is current object greater than or equal to other?
                    -- (from PART_COMPARABLE)
                require  -- from PART_COMPARABLE
                    other_exists: other /= void

feature  -- Basic operations

        infix  "+" (other: like  Current): like  Current
                    -- Sum with other (commutative)
                    -- (from GROUP_ELEMENT)
                require  -- from GROUP_ELEMENT
                    other_exists: other /= void
                ensure  -- from GROUP_ELEMENT
                    result_exists: Result /= void;
                    commutative: Result.is_equal (other + Current)

        prefix  "+ ": like  Current
                    -- Unary plus
                    -- (from GROUP_ELEMENT)
                ensure  -- from GROUP_ELEMENT
                    result_exists: Result /= void;
                    result_definition: Result.is_equal (Current)

        infix  "-" (other: like  Current): like  Current
                    -- Result of subtracting other
                    -- (from GROUP_ELEMENT)
                require  -- from GROUP_ELEMENT
                    other_exists: other /= void
                ensure  -- from GROUP_ELEMENT
                    result_exists: Result /= void

        prefix  "- ": like  Current
                    -- Unary minus
                    -- (from GROUP_ELEMENT)
                ensure  -- from GROUP_ELEMENT
                    result_exists: Result /= void;
                    result_definition: (Result + Current).is_equal (zero)

invariant

                -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);
                -- from GROUP_ELEMENT
        neutral_addition: Current.is_equal (Current + zero);
        self_subtraction: zero.is_equal (Current - Current);
```

end -- class *DURATION*

6.1.12. *GROUP_ELEMENT*

indexing

description: "*Invertible object with an internal + operation*"
note: "*The model is that of a commutative group.*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:58 $*"
revision: "*$Revision: 4.2 $*"
access: *algebra*

deferred class interface
GROUP_ELEMENT

feature -- Access

zero: ***like*** *Current*
-- Neutral element for "+" and "-"
ensure
result_exists: *Result* /= *void*

feature -- Basic operations

infix "+" (*other*: ***like*** *Current*): ***like*** *Current*
-- Sum with *other* (commutative)
require
other_exists: *other* /= *void*
ensure
result_exists: *Result* /= *void*;
commutative: *Result.is_equal* (*other* + *Current*)

prefix "+ ": ***like*** *Current*
-- Unary plus
ensure
result_exists: *Result* /= *void*;
result_definition: *Result.is_equal* (*Current*)

infix "-" (*other*: ***like*** *Current*): ***like*** *Current*
-- Result of subtracting *other*
require
other_exists: *other* /= *void*
ensure
result_exists: *Result* /= *void*

prefix "- ": ***like*** *Current*
-- Unary minus
ensure
result_exists: *Result* /= *void*;

result_definition: (*Result + Current*)**.***is_equal* (*zero*)

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
neutral_addition: *Current***.***is_equal* (*Current + zero*);
self_subtraction: *zero***.***is_equal* (*Current - Current*);

end -- class *GROUP_ELEMENT*

6.1.13. *INTERVAL [G -> ABSOLUTE]*

indexing
description: "*Interval of G*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 16:59:59 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
INTERVAL [*G -> ABSOLUTE*]

creation

make (*s*, *e*: *G*)
-- Sets *start_bound* and *end_bound* to *s* and *e* respectively.
require
s_exists: *s /= void*;
e_exists: *e /= void*;
s_before_e: *s <= e*
ensure
start_bound_set: *start_bound /= void* ***and then*** *deep_equal* (*start_bound*, *s*);
end_bound_set: *end_bound /= void* ***and then*** *deep_equal* (*end_bound*, *e*)

feature -- Initialization

make (*s*, *e*: *G*)
-- Sets *start_bound* and *end_bound* to *s* and *e* respectively.
require
s_exists: *s /= void*;
e_exists: *e /= void*;
s_before_e: *s <= e*
ensure
start_bound_set: *start_bound /= void* ***and then*** *deep_equal* (*start_bound*, *s*);
end_bound_set: *end_bound /= void* ***and then*** *deep_equal* (*end_bound*, *e*)

feature -- Access

end_bound: *G*
-- End bound of the current interval

start_bound: *G*
-- Start bound of the current interval

feature -- Measurement

duration: *DURATION*
-- lenght of the interval

feature -- Comparison

includes (*other*: ***like*** *Current*): *BOOLEAN*
-- Does the current interval include *other*?
-- This feature will be removed when the repeated inheritance bug will be fixed
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*

intersects (*other*: ***like*** *Current*): *BOOLEAN*
-- Does the current interval intersect *other*?
require
other_exists: *other* /= *void*

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Are the current interval an *other* the same interval?
require -- from *GENERAL*
other_not_void: *other* /= *void*
ensure -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*

is_included_by (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current interval includded by *other*?
require -- from *PART_COMPARABLE*
other_exists: *other* /= *void*

is_met_by (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current interval met by *other*?
require
other_exists: *other* /= *void*
ensure
symetry: *Result* = *other.meets* (*Current*)

is_overlapped_by (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current interval overlapped by *other*?
require
other_exists: *other* /= *void*
ensure

```
                symetry: Result = other.overlaps (Current)

is_strict_included_by (other: like Current): BOOLEAN
                -- Is the current interval strictly includded by other?
        require -- from PART_COMPARABLE
                other_exists: other /= void

meets (other: like Current): BOOLEAN
                -- Does the current interval meet other?
        require
                other_exists: other /= void
        ensure
                symetry: Result = other.is_met_by (Current);
                result_definition: Result = (Current <= other and intersects (other))

overlaps (other: like Current): BOOLEAN
                -- Does the current interval overlaps other?
        require
                other_exists: other /= void
        ensure
                result_defition: Result = (strict_before (other.end_bound) and has (other.start_bound));
                symetry: Result = other.is_overlapped_by (Current)

strict_includes (other: like Current): BOOLEAN
                -- Does the current interval strictly include other?
                -- This feature will be removed when the repeated inheritance bug will be fixed.
        require -- from PART_COMPARABLE
                other_exists: other /= void

infix "<" (other: like Current): BOOLEAN
                -- Is the current interval strictly before other?
        require -- from PART_COMPARABLE
                other_exists: other /= void

infix "<=" (other: like Current): BOOLEAN
                -- Is the current interval before other?
        require -- from PART_COMPARABLE
                other_exists: other /= void

infix ">" (other: like Current): BOOLEAN
                -- Is the current interval after other?
        require -- from PART_COMPARABLE
                other_exists: other /= void

infix ">=" (other: like Current): BOOLEAN
                -- Is the current interval after other?
        require -- from PART_COMPARABLE
                other_exists: other /= void
```

feature -- Status report

after (*g*: *G*): *BOOLEAN*
-- Is the current interval after *g*?
require
g_exist: *g /= void*
ensure
result_definition: *Result* = (*start_bound >= g*)

before (*g*: *G*): *BOOLEAN*
-- Is the current interval before *g*?
require
g_exist: *g /= void*
ensure
result_definition: *Result* = (*end_bound <= g*)

empty: *BOOLEAN*
-- Is the current interval empty?
ensure
result_definition: *Result = duration.is_equal* (*duration.zero*)

has (*g*: *G*): *BOOLEAN*
-- Does the current interval have *g* between its bounds?
require
g_exist: *g /= void*
ensure
result_definition: *Result* ***xor not*** ((*start_bound <= g*) ***and then*** (*end_bound >= g*))

strict_after (*g*: *G*): *BOOLEAN*
-- Is the current interval strictly after *g*?
require
g_exist: *g /= void*
ensure
result_definition: *Result* = (*start_bound > g*)

strict_before (*g*: *G*): *BOOLEAN*
-- Is the current interval strictly before *g*?
require
g_exist: *g /= void*
ensure
result_definition: *Result* ***xor*** (***not*** (*end_bound < g*))

feature -- Element change

set_end_bound (*e*: *G*)
-- Set *end_bound* to *e*.
require
e_after_end_bound: *e /= void* ***and then*** *e >= start_bound*
ensure

```
                end_bound_set: end_bound /= void and then  end_bound.is_equal (e)

        set_start_bound (s: G)
                -- Set start_bound to s.
            require
                s_before_end_bound: s /= void and then  s <= end_bound
            ensure
                start_bound_set: start_bound /= void and then  start_bound.is_equal (s)

feature  -- Conversion

        out: STRING
                -- Printable representation of the current interval

feature  -- Basic operations

        gather (other: like  Current): like  Current
                -- Union of other and current interval if other meets current interval
            require
                other_exist: other /= void;
                meeting_interval: meets (other)
            ensure
                result_exist: Result /= void;
                result_same_as_union: Result.is_equal (union (other))

        intersection (other: like  Current): like  Current
                -- Intersection with other
            require
                other_exists: other /= void
            ensure
                intersects_validity: intersects (other) implies  Result /= void;
                result_is_included_by_current: intersects (other) implies  includes (Result);
                result_is_included_by_other: intersects (other) implies  other.includes (Result)

        union (other: like  Current): like  Current
                -- Union with other
            require
                other_exists: other /= void;
                intersects: intersects (other)
            ensure
                result_exists: Result /= void;
                result_includes_current: Result.includes (Current);
                result_includes_other: Result.includes (other)

invariant

                -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);
```

start_bound_exists: *start_bound* /= *void*;
end_bound_exists: *end_bound* /= *void*;
start_bound_before_end_bound: *start_bound* <= *end_bound*;
current_intersection: *intersection* (*Current*)**.***is_equal* (*Current*);
current_union: *union* (*Current*)**.***is_equal* (*Current*);
has_bounds: *has* (*start_bound*) ***and*** *has* (*end_bound*);
between_bound: *after* (*start_bound*) ***and*** *before* (*end_bound*);

end -- class *INTERVAL*

6.1.14. *TIME*

indexing

description: "*absolute time*"
status: "*See notice at end of class*"
date: "*$Date: *"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface

TIME

creation

make (*h, m, s*: *INTEGER*)
-- Set *hour, ‘minute* and *second* to *h, m, s* respectively.
require
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *m >= 0*;
m_small_enough: *m < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
hour_set: *hour = h*;
minute_set: *minute = m*;
second_set: *second = s*

make_fine (*h, m*: *INTEGER*; *s*: *DOUBLE*)
-- Set *hour, ‘minute* and *second* to *h, m* and truncated to integer part of *s* respectively.
-- Set *fractionnal_second* to the fractionnal part of *s*.
require
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *m >= 0*;
m_small_enough: *m < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
hour_set: *hour = h*;

minute_set: *minute = m*;
fine_second_set: *fine_second = s*

make_now
-- Set current time according to timezone.

make_by_seconds (*sec*: *INTEGER*)
-- Set the object by the number of seconds *sec* from midnight.
require
s_large_enough: *sec >= 0*;
s_small_enough: *sec < seconds_in_day*
ensure
seconds_set: *seconds = sec*

make_by_fine_seconds (*sec*: *DOUBLE*)
-- Set the object by the number of seconds *sec*.
require
s_large_enough: *sec >= 0*;
s_small_enough: *sec < seconds_in_day*

make_from_string (*s*: *STRING*; *code*: *STRING*)
-- Initialise from a "standard" string of form
-- *code*
require
s_exists: *s /= void*;
c_exists: *code /= void*;
time_valid: *time_valid* (*s, code*)

make_by_compact_time (*c_t*: *INTEGER*)
-- Initialize from *compact_time*.
require
c_t_not_void: *c_t /= void*;
c_t_valid: *compact_time_valid* (*c_t*)
ensure
compact_time_set: *compact_time = c_t*

feature -- Initialization

make (*h*, *m*, *s*: *INTEGER*)
-- Set *hour, ‘minute* and *second* to *h*, *m*, *s* respectively.
require
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *m >= 0*;
m_small_enough: *m < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
hour_set: *hour = h*;

minute_set: *minute = m*;
second_set: *second = s*

make_by_compact_time (*c_t*: *INTEGER*)
-- Initialize from *compact_time*.
require
c_t_not_void: *c_t /= void*;
c_t_valid: *compact_time_valid* (*c_t*)
ensure
compact_time_set: *compact_time = c_t*

make_by_fine_seconds (*sec*: *DOUBLE*)
-- Set the object by the number of seconds *sec*.
require
s_large_enough: *sec >= 0*;
s_small_enough: *sec < seconds_in_day*

make_by_seconds (*sec*: *INTEGER*)
-- Set the object by the number of seconds *sec* from midnight.
require
s_large_enough: *sec >= 0*;
s_small_enough: *sec < seconds_in_day*
ensure
seconds_set: *seconds = sec*

make_fine (*h*, *m*: *INTEGER*; *s*: *DOUBLE*)
-- Set *hour, 'minute* and *second* to *h*, *m* and truncated to integer part of *s* respectively.
-- Set *fractionnal_second* to the fractionnal part of *s*.
require
h_large_enough: *h >= 0*;
h_small_enough: *h < hours_in_day*;
m_large_enough: *m >= 0*;
m_small_enough: *m < minutes_in_hour*;
s_large_enough: *s >= 0*;
s_small_enough: *s < seconds_in_minute*
ensure
hour_set: *hour = h*;
minute_set: *minute = m*;
fine_second_set: *fine_second = s*

make_from_string (*s*: *STRING*; *code*: *STRING*)
-- Initialise from a "standard" string of form
-- *code*
require
s_exists: *s /= void*;
c_exists: *code /= void*;
time_valid: *time_valid* (*s*, *code*)

make_from_string_default (*s*: *STRING*)

-- Initialise from a "standard" string of form
-- *default_format_string*
require
s_exists: *s* /= *void*;
time_valid: *time_valid* (*s*, *default_format_string*)

make_now
-- Set current time according to timezone.

feature -- Access

compact_time: *INTEGER*
-- Hour, minute, second coded.
-- (from *TIME_VALUE*)

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

fine_second: *DOUBLE*
-- Representation of second with decimals
-- (from *TIME_VALUE*)

fractionnal_second: *DOUBLE*
-- Fractionnal part of *fine_second*
-- (from *TIME_VALUE*)

hour: *INTEGER*
-- Hour of the current time
-- (from *TIME_VALUE*)

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

micro_second: *INTEGER*
-- Microsecond of the current time
-- (from *TIME_VALUE*)

milli_second: *INTEGER*
-- Millisecond of the current time
-- (from *TIME_VALUE*)

minute: *INTEGER*
-- Minute of the current time
-- (from *TIME_VALUE*)

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

nano_second: *INTEGER*
-- Nanosecond of the current time
-- (from *TIME_VALUE*)

origin: *TIME*
-- Origin time
ensure -- from *ABSOLUTE*
result_exists: *Result* /= *void*

second: *INTEGER*
-- Second of the current time
-- (from *TIME_VALUE*)

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute
-- (from *TIME_CONSTANTS*)

time_default_format_string: *STRING*
-- (from *TIME_CONSTANTS*)

feature -- Measurement

duration: *TIME_DURATION*
-- Duration elapsed from midnight
ensure
seconds_large_enough: *duration.seconds_count* >= *0*;
seconds_small_enough: *duration.seconds_count* < *seconds_in_day*

fine_seconds: *DOUBLE*
-- Number of seconds and fractions of seconds elapsed from midnight

seconds: *INTEGER*
-- Number of seconds elapsed from midnight
ensure
result_definition: *Result* = *duration.seconds_count*

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* attached to an object of the same type
-- as current object and identical to it?
-- (from *COMPARABLE*)
require -- *COMPARABLE*
precursor: True
require else -- from *GENERAL*
other_not_void: *other /= void*
ensure -- from *COMPARABLE*
trichotomy: *Result* = (***not*** (*Current* < *other*) ***and not*** (*other* < *Current*))
ensure then -- from *GENERAL*
symmetric: *Result* ***implies*** *other.is_equal* (*Current*);
consistent: *standard_is_equal* (*other*) ***implies*** *Result*
ensure then -- from *COMPARABLE*
trichotomy: *Result* = (***not*** (*Current* < *other*) ***and not*** (*other* < *Current*))

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
current_if_not_smaller: *Current >= other* ***implies*** *Result = Current*;
other_if_smaller: *Current < other* ***implies*** *Result = other*

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
current_if_not_greater: *Current <= other* ***implies*** *Result = Current*;
other_if_greater: *Current > other* ***implies*** *Result = other*

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0;
-- if smaller, -1; if greater, 1
-- (from *COMPARABLE*)
require -- from *COMPARABLE*
other_exists: *other /= void*
ensure -- from *COMPARABLE*
equal_zero: (*Result = 0*) = *is_equal* (*other*);
smaller_negative: (*Result = -1*) = (*Current* < *other*);
greater_positive: (*Result = 1*) = (*Current* > *other*)

infix "<" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is the current time before *other*?
require -- from *PART_COMPARABLE*

```
                other_exists: other /= void
        ensure  -- from COMPARABLE
                asymmetric: Result implies not (other < Current)

    infix "<=" (other: like Current): BOOLEAN
                -- Is current object less than or equal to other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = ((Current < other) or is_equal (other))

    infix ">" (other: like Current): BOOLEAN
                -- Is current object greater than other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = (other < Current)

    infix ">=" (other: like Current): BOOLEAN
                -- Is current object greater than or equal to other?
                -- (from COMPARABLE)
        require  -- from PART_COMPARABLE
                other_exists: other /= void
        ensure  -- from COMPARABLE
                definition: Result = (other <= Current)

feature  -- Element change

    set_fine_second (s: DOUBLE)
                -- Set fine_second to s
        require
                s_large_enough: s >= 0;
                s_small_enough: s < seconds_in_minute
        ensure
                fine_second_set: fine_second = s

    set_fractionnals (f: DOUBLE)
                -- Set fractionnal_second to f.
        require
                f_large_enough: f >= 0;
                f_small_enough: f < 1
        ensure
                second_same: second = old second

    set_hour (h: INTEGER)
                -- Set hour to h.
        require
```

```
                h_large_enough: h >= 0;
                h_small_enough: h < hours_in_day
        ensure
                hour_set: hour = h

    set_minute (m: INTEGER)
                -- Set minute to m.
        require
                m_large_enough: m >= 0;
                m_small_enough: m < minutes_in_hour
        ensure
                minute_set: minute = m

    set_second (s: INTEGER)
                -- Set second to s.
        require
                s_large_enough: s >= 0;
                s_small_enough: s < seconds_in_minute
        ensure
                second_set: second = s

feature -- Basic operations

    div (i, j: INTEGER): INTEGER
                -- (i \\ j) if i positive
                -- (i \\ j + 1) if i negative
                -- (from TIME_UTILITY)
        ensure -- from TIME_UTILITY
                result_definition: i = j * Result + mod (i, j)

    fine_second_add (f: DOUBLE)
                -- Add f seconds to the current time.
                -- if f has decimals, fractionnal_second is modified.

    hour_add (h: INTEGER)
                -- Add h hours to the current object.

    hour_back
                -- Move to evious hour.

    hour_forth
                -- Move to next hour.

    minute_add (m: INTEGER)
                -- Add m minutes to the current object.

    minute_back
                -- Move to evious minute.
```

minute_forth
-- Move to next minute.

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i*, *j*) + *Result*

relative_duration (*other*: ***like*** *Current*): *TIME_DURATION*
-- Duration elapsed from *other* to *Current*
require -- from *ABSOLUTE*
other_exists: *other /= void*
ensure -- from *ABSOLUTE*
result_exists: *Result /= void*

second_add (*s*: *INTEGER*)
-- Add *s* seconds to the current time.

second_back
-- Move to previous second.

second_forth
-- Move to next second.

infix "+" (*t*: *TIME_DURATION*): *TIME*
-- Sum of the current time and duration *t*
require
t_exists: *t /= void*
ensure
result_exists: *Result /= void*

infix "-" (*other*: ***like*** *Current*): *INTERVAL* [***like*** *Current*]
-- Interval between current object and *other*
-- (from *ABSOLUTE*)
require -- from *ABSOLUTE*
other_exists: *other /= void*;
other_smaller_than_current: *other <= Current*
ensure -- from *ABSOLUTE*
result_exists: *Result /= void*;
result_set: *Result.start_bound.is_equal* (*other*) ***and then*** *Result.end_bound.is_equal* (*Current*)

feature -- Output

formatted_out (*s*: *STRING*): *STRING*
-- Printable representation of time with "standard"

-- Form: *s*

require

s_exists: *s /= void*

out: *STRING*

-- Printable representation of time with "standard"
-- Form: *time_default_format_string*

feature -- Preconditions

compact_time_valid (*c_t*: *INTEGER*): *BOOLEAN*

require

c_t_not_void: *c_t /= void*

time_valid (*s*: *STRING*; *code_string*: *STRING*): *BOOLEAN*

-- Is the code_string enough precise
-- To create an instance of type TIME
-- And does the string *s* correspond to *code_string*?

require

s_exists: *s /= void*;
code_exists: *code_string /= void*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
second_large_enough: *second >= 0*;
second_small_enough: *second < seconds_in_minute*;
fractionnals_large_enough: *fractionnal_second >= 0*;
fractionnals_small_enough: *fractionnal_second < 1*;
minute_large_enough: *minute >= 0*;
minute_small_enough: *minute < minutes_in_hour*;
hour_large_enough: *hour >= 0*;
hour_small_enough: *hour < hours_in_day*;
-- from *COMPARABLE*
irreflexive_comparison: ***not*** (*Current < Current*);

end -- class *TIME*

6.1.15. *TIME_CONSTANTS*

indexing

description: "*Universal constants of time in a day*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 17:00:00 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface

TIME_CONSTANTS

feature -- Access

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute

time_default_format_string: *STRING*

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i = j * Result + mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i*, *j*) + *Result*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *TIME_CONSTANTS*

6.1.16 *TIME_DURATION*

indexing

description: "*duration expressed in time*"
status: "*See notice at end of class*"
date: "*$Date: *"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface

TIME_DURATION

creation

make (*h*, *m*, *s*: *INTEGER*)
-- Set *hour*, *minute* and *second* to *h*, *m*, *s* respectively.
ensure
hour_set: *hour = h*;
minute_set: *minute = m*;
second_set: *second = s*

make_fine (*h*, *m*: *INTEGER*; *s*: *DOUBLE*)
-- Set *hour, ‘minute* and *second* to *h*, *m* and truncated to integer part of *s* respectively.
-- Set *fractionnal_second* to the fractionnal part of *s*.
ensure
hour_set: *hour = h*;
minute_set: *minute = m*;
fine_second_set: *fine_second = s*

make_by_seconds (*s*: *INTEGER*)
-- Set the object by the number of seconds *s*.
ensure
seconds_count_set: *seconds_count = s*

make_by_fine_seconds (*s*: *DOUBLE*)
-- Set the object by the number of seconds *s*.
ensure
minute_large_enough: *minute >= 0*;
minute_small_enough: *minute < minutes_in_hour*;
second_large_enough: *second >= 0*;
second_small_enough: *second < seconds_in_minute*;
fine_seconds_set: *fine_seconds_count = s*

feature -- Initialization

make (*h*, *m*, *s*: *INTEGER*)
-- Set *hour*, *minute* and *second* to *h*, *m*, *s* respectively.

ensure

hour_set: *hour = h*;
minute_set: *minute = m*;
second_set: *second = s*

make_by_fine_seconds (*s*: *DOUBLE*)

-- Set the object by the number of seconds *s*.

ensure

minute_large_enough: *minute >= 0*;
minute_small_enough: *minute < minutes_in_hour*;
second_large_enough: *second >= 0*;
second_small_enough: *second < seconds_in_minute*;
fine_seconds_set: *fine_seconds_count = s*

make_by_seconds (*s*: *INTEGER*)

-- Set the object by the number of seconds *s*.

ensure

seconds_count_set: *seconds_count = s*

make_fine (*h*, *m*: *INTEGER*; *s*: *DOUBLE*)

-- Set *hour, 'minute* and *second* to *h*, *m* and truncated to integer part of *s* respectively.
-- Set *fractionnal_second* to the fractionnal part of *s*.

ensure

hour_set: *hour = h*;
minute_set: *minute = m*;
fine_second_set: *fine_second = s*

feature -- Access

compact_time: *INTEGER*

-- Hour, minute, second coded.
-- (from *TIME_VALUE*)

date_time_tools: *DATE_TIME_TOOLS*

-- (from *TIME_UTILITY*)

default_format_string: *STRING*

-- (from *TIME_UTILITY*)

fine_seconds_count: *DOUBLE*

-- Number of seconds and fractionnals of seconds of the current duration

time_value_frac_sec: *DOUBLE*

-- Fractionnal part of *fine_second*
-- (from *TIME_VALUE*)

Hours_in_day: *INTEGER* ***is*** *24*

-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

micro_second: *INTEGER*
-- Microsecond of the current time
-- (from *TIME_VALUE*)

milli_second: *INTEGER*
-- Millisecond of the current time
-- (from *TIME_VALUE*)

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

nano_second: *INTEGER*
-- Nanosecond of the current time
-- (from *TIME_VALUE*)

seconds_count: *INTEGER*
-- Total number of seconds of the current duration
ensure
same_count: *Result = fine_seconds_count.truncated_to_integer*

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute
-- (from *TIME_CONSTANTS*)

time_default_format_string: *STRING*
-- (from *TIME_CONSTANTS*)

zero: *TIME_DURATION*
-- Neutral element for "+" and "-"
ensure -- from *GROUP_ELEMENT*
result_exists: *Result /= void*

feature -- Comparison

infix "<=" (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (from *PART_COMPARABLE*)
require -- from *PART_COMPARABLE*
other_exists: *other /= void*

```
        infix ">" (other: like Current): BOOLEAN
                        -- Is current object greater than other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

        infix ">=" (other: like Current): BOOLEAN
                        -- Is current object greater than or equal to other?
                        -- (from PART_COMPARABLE)
                require -- from PART_COMPARABLE
                        other_exists: other /= void

feature -- Status report

        canonical: BOOLEAN
                        -- Is duration expressed minimally, i.e.
                        --      If duration is positive then
                        --              hour positive,
                        --              minute and second between 0 and 59,
                        --              fractionnal_second between 0 and 999?
                        --      If duration is negative then
                        --              hour negative,
                        --              minute and second between -59 and 0,
                        --              fractionnal_second between -999 and 0?

feature -- Conversion

        time_modulo_day: like Current
                        -- Duration modulo duration of a day
                ensure
                        result_smaller_than_day: Result.seconds_count < seconds_in_day;
                        result_positive: Result >= zero

        to_canonical: like Current
                        -- A new duration
                ensure
                        result_canonical: Result.canonical

        to_days: INTEGER
                        -- Total number of days equivalent to the current duration

feature -- Basic operations

        div (i, j: INTEGER): INTEGER
                        -- (i \\ j) if i positive
                        -- (i \\ j + 1) if i negative
                        -- (from TIME_UTILITY)
                ensure -- from TIME_UTILITY
```

result_definition: *i = j * Result + mod* (*i*, *j*)

fine_second_add (*s*: *DOUBLE*)
-- Add *s* seconds to the current time.
-- if *s* has decimals, *fractionnal_second* is modifed.

hour_add (*h*: *INTEGER*)
-- Add *h* hours to the current duration.
ensure
hour_set: *hour* = ***old*** *hour + h*

minute_add (*m*: *INTEGER*)
-- Add *m* minutes to the current duration.
ensure
minute_set: *minute* = ***old*** *minute + m*

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i*, *j*) + *Result*

second_add (*s*: *INTEGER*)
-- Add *s* seconds to the current duration.
ensure
second_set: *second* = ***old*** *second + s*

infix "+" (*other*: ***like*** *Current*): ***like*** *Current*
-- Sum with *other*
require -- from *GROUP_ELEMENT*
other_exists: *other /= void*
ensure -- from *GROUP_ELEMENT*
result_exists: *Result /= void*;
commutative: *Result.is_equal* (*other + Current*)

prefix "+ ": ***like*** *Current*
-- Unary plus
ensure -- from *GROUP_ELEMENT*
result_exists: *Result /= void*;
result_definition: *Result.is_equal* (*Current*)

infix "-" (*other*: ***like*** *Current*): ***like*** *Current*
-- Difference with *other*
require -- from *GROUP_ELEMENT*
other_exists: *other /= void*
ensure -- from *GROUP_ELEMENT*
result_exists: *Result /= void*

```
        prefix  "- ": like  Current
                        -- Unary minus
                ensure  -- from GROUP_ELEMENT
                        result_exists: Result /= void;
                        result_definition: (Result + Current).is_equal (zero)

feature  -- Attributes

        fine_second: DOUBLE

        fractionnal_second: DOUBLE

        hour: INTEGER

        minute: INTEGER

        second: INTEGER

feature  -- Comparaison

        is_equal (other: like  Current): BOOLEAN
                        -- Is the current duration equal to other?
                require  -- from GENERAL
                        other_not_void: other /= void
                ensure  -- from GENERAL
                        symmetric: Result implies  other.is_equal (Current);
                        consistent: standard_is_equal (other) implies  Result

        infix  "<" (other: like  Current): BOOLEAN
                        -- Is the current duration smaller than other?
                require  -- from PART_COMPARABLE
                        other_exists: other /= void

feature  -- Element Change

        set_fine_second (s: DOUBLE)
                        -- Set fine_second to s.
                ensure
                        second_set: fine_second = s

        set_fractionnals (f: DOUBLE)
                        -- Set fractionnal_second to f.
                        -- f must have the same sign as second.
                require
                        same_sign: (f.sign = second.sign) or else  f.sign = 0 or else  second.sign = 0;
                        f_large_enough: f > -1;
                        f_small_enough: f < 1
```

set_hour (*h*: *INTEGER*)
-- Set *hour* to *h*.
ensure
hour_set: *hour = h*

set_minute (*m*: *INTEGER*)
-- Set *minute* to *m*.
ensure
minute_set: *minute = m*

set_second (*s*: *INTEGER*)
-- Set *second* to *s*.
-- *fractionnal_second* is cut down to 0.
ensure
second_set: *second = s*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);
fractionnals_large_enough: *fractionnal_second > -1*;
fractionnals_small_enough: *fractionnal_second < 1*;
fractionnal_and_second_same_sign: *second * fractionnal_second >= 0*;
-- from *GROUP_ELEMENT*
neutral_addition: *Current.is_equal* (*Current + zero*);
self_subtraction: *zero.is_equal* (*Current - Current*);

end -- class *TIME_DURATION*

6.1.17. *TIME_SET*

indexing
description: "*Sets of compactly coded times*"
date: "*$Date: 1998/6/5 10:46 AM $*"
revision: "*$Revision: 4.3$*"

class interface
TIME_SET

creation

make (*n*: *INTEGER*)
-- Create structure for initial
-- estimate of *n* times.
require
n_not_void: *n /= void*

feature -- Initialization

make_array (*minindex*, *maxindex*: *INTEGER*)
-- Allocate array; set index interval to
-- *minindex* .. *maxindex*; set all values to default.
-- (Make array empty if *minindex* = *maxindex* + 1).
-- (from *ARRAY*)
require -- from *ARRAY*
valid_indices: *minindex* <= *maxindex* ***or*** (*minindex* = *maxindex* + *1*)
ensure -- from *ARRAY*
lower = *minindex*;
upper = *maxindex*

make_from_array (*a*: *ARRAY* [*NUMERIC*])
-- Initialize from the items of *a*.
-- (Useful in proper descendants of class *ARRAY*,
-- to initialize an array-like object from a manifest array.)
-- (from *ARRAY*)
require -- from *ARRAY*
array_exists: *a* /= *void*

setup (*other*: ***like*** *Current*)
-- Perform actions on a freshly created object so that
-- the contents of *other* can be safely copied onto it.
-- (from *ARRAY*)
ensure -- from *GENERAL*
consistent (*other*)

feature -- Access

area: *SPECIAL* [*NUMERIC*]
-- Special data zone
-- (from *TO_SPECIAL*)

entry (*i*: *INTEGER*): *NUMERIC*
-- Entry at index *i*, if in index interval
-- Was declared in ARRAY as synonym of *item*, @ and *entry*.
-- (from *ARRAY*)

has (*v*: *NUMERIC*): *BOOLEAN*
-- Does *v* appear in array?
-- (Reference or object equality,
-- based on *object_comparison*.)
-- (from *ARRAY*)
ensure -- from *CONTAINER*
not_found_in_empty: *Result* ***implies*** ***not*** *empty*

item (*i*: *INTEGER*): *TIME*
-- Element at index *i*
require
i_not_void: *i* /= *void*

frozen *item_array* (*i*: *INTEGER*): *NUMERIC*
-- Entry at index *i*, if in index interval
-- Was declared in ARRAY as synonym of *item*, @ and *entry.*
-- (from *ARRAY*)
require -- from *TABLE*
valid_key: *valid_index* (*k*)

last: *INTEGER*
-- Index of the last element inserted

frozen ***infix*** "@" (*i*: *INTEGER*): *NUMERIC*
-- Entry at index *i*, if in index interval
-- Was declared in ARRAY as synonym of *item*, @ and *entry.*
-- (from *ARRAY*)
require -- from *TABLE*
valid_key: *valid_index* (*k*)

feature -- Measurement

additional_space: *INTEGER*
-- Proposed number of additional items
-- (from *RESIZABLE*)
ensure -- from *RESIZABLE*
at_least_one: *Result >= 1*

capacity: *INTEGER*
-- Number of available indices
-- Was declared in ARRAY as synonym of *count* and *capacity.*
-- (from *ARRAY*)

count: *INTEGER*
-- Number of available indices
-- Was declared in ARRAY as synonym of *count* and *capacity.*
-- (from *ARRAY*)

Growth_percentage: *INTEGER* ***is*** *50*
-- Percentage by which structure will grow automatically
-- (from *RESIZABLE*)

lower: *INTEGER*
-- Minimum index
-- (from *ARRAY*)

Minimal_increase: *INTEGER* ***is*** *5*
-- Minimal number of additional items
-- (from *RESIZABLE*)

occurrences (*v*: *NUMERIC*): *INTEGER*

```
                -- Number of times v appears in structure
                -- (from ARRAY)
            ensure  -- from BAG
                non_negative_occurrences: Result >= 0

        upper: INTEGER
                -- Maximum index
                -- (from ARRAY)

feature  -- Comparison

        is_equal (other: like Current): BOOLEAN
                -- Is array made of the same items as other?
                -- (from ARRAY)
            require  -- from GENERAL
                other_not_void: other /= void
            ensure  -- from GENERAL
                symmetric: Result implies other.is_equal (Current);
                consistent: standard_is_equal (other) implies Result

feature  -- Status report

        all_cleared: BOOLEAN
                -- Are all items set to default values?
                -- (from ARRAY)

        changeable_comparison_criterion: BOOLEAN
                -- May object_comparison be changed?
                -- (Answer: yes by default.)
                -- (from CONTAINER)

        consistent (other: like Current): BOOLEAN
                -- Is object in a consistent state so that other
                -- may be copied onto it? (Default answer: yes).
                -- (from ARRAY)

        empty: BOOLEAN
                -- Is structure empty?
                -- (from FINITE)

        extendible: BOOLEAN
                -- May items be added?
                -- (Answer: no, although array may be resized.)
                -- (from ARRAY)

        full: BOOLEAN
                -- Is structure filled to capacity? (Answer: yes)
                -- (from ARRAY)
```

object_comparison: *BOOLEAN*
-- Must search operations use *equal* rather than =
-- for comparing references? (Default: no, use =.)
-- (from *CONTAINER*)

prunable: *BOOLEAN*
-- May items be removed? (Answer: no.)
-- (from *ARRAY*)

resizable: *BOOLEAN*
-- May *capacity* be changed? (Answer: yes.)
-- (from *RESIZABLE*)

valid_index (*i*: *INTEGER*): *BOOLEAN*
-- Is *i* within the bounds of the array?
-- (from *ARRAY*)

feature -- Status setting

compare_objects
-- Ensure that future search operations will use *equal*
-- rather than = for comparing references.
-- (from *CONTAINER*)
require -- from *CONTAINER*
changeable_comparison_criterion
ensure -- from *CONTAINER*
object_comparison

compare_references
-- Ensure that future search operations will use =
-- rather than *equal* for comparing references.
-- (from *CONTAINER*)
require -- from *CONTAINER*
changeable_comparison_criterion
ensure -- from *CONTAINER*
reference_comparison: ***not*** *object_comparison*

feature -- Element change

enter (*v*: ***like*** *item_array*; *i*: *INTEGER*)
-- Replace *i*-th entry, if in index interval, by *v*.
-- Was declared in ARRAY as synonym of *put* and *enter.*
-- (from *ARRAY*)

fill (*other*: *CONTAINER* [*NUMERIC*])
-- Fill with as many items of *other* as possible.
-- The representations of *other* and current structure
-- need not be the same.
-- (from *COLLECTION*)

```
            require  -- from COLLECTION
                     other_not_void: other /= void;
                     extendible

        force (v: like item_array; i: INTEGER)
                     -- Assign item v to i-th entry.
                     -- Always applicable: resize the array if i falls out of
                     -- currently defined bounds; preserve existing items.
                     -- (from ARRAY)
            ensure   -- from ARRAY
                     inserted: item_array (i) = v;
                     higher_count: count >= old count

        put (d: TIME)
                     -- Insert d;
                     -- Index will be given by last.
            require
                     d_not_void: d /= void
            ensure
                     inserted: item (last).is_equal (d)

        frozen put_array (v: like item_array; i: INTEGER)
                     -- Replace i-th entry, if in index interval, by v.
                     -- Was declared in ARRAY as synonym of put and enter.
                     -- (from ARRAY)
            require  -- from TABLE
                     valid_key: valid_index (k)
            ensure   -- from INDEXABLE
                     insertion_done: item_array (k) = v

        subcopy (other: like Current; start_pos, end_pos, index_pos: INTEGER)
                     -- Copy items of other within bounds start_pos and end_pos
                     -- to current array starting at index index_pos.
                     -- (from ARRAY)
            require  -- from ARRAY
                     other_not_void: other /= void;
                     valid_start_pos: other.valid_index (start_pos);
                     valid_end_pos: other.valid_index (end_pos);
                     valid_bounds: (start_pos <= end_pos) or (start_pos = end_pos + 1);
                     valid_index_pos: valid_index (index_pos);
                     enough_space: (upper - index_pos) >= (end_pos - start_pos)

feature  -- Removal

        clear_all
                     -- Reset all items to default values.
                     -- (from ARRAY)
            ensure   -- from ARRAY
                     all_cleared: all_cleared
```

```
        prune_all (v: NUMERIC)
                        -- Remove all occurrences of v.
                        -- (Reference or object equality,
                        -- based on object_comparison.)
                        -- (from COLLECTION)
                require -- from COLLECTION
                        prunable
                ensure  -- from COLLECTION
                        no_more_occurrences: not has (v)

        wipe_out
                        -- Make array empty.
                        -- (from ARRAY)
                require -- from COLLECTION
                        prunable
                ensure  -- from COLLECTION
                        wiped_out: empty

feature -- Resizing

        automatic_grow
                        -- Change the capacity to accommodate at least
                        -- Growth_percentage more items.
                        -- (from RESIZABLE)
                ensure  -- from RESIZABLE
                        increased_capacity: capacity >= old capacity + old capacity * growth_percentage // 100

        grow (i: INTEGER)
                        -- Change the capacity to at least i.
                        -- (from ARRAY)
                ensure  -- from RESIZABLE
                        new_capacity: capacity >= i

        resize (minindex, maxindex: INTEGER)
                        -- Rearrange array so that it can accommodate
                        -- indices down to minindex and up to maxindex.
                        -- Do not lose any previously entered item.
                        -- (from ARRAY)
                require -- from ARRAY
                        good_indices: minindex <= maxindex
                ensure  -- from ARRAY
                        no_low_lost: lower = minindex.min (old lower);
                        no_high_lost: upper = maxindex.max (old upper)

feature -- Conversion

        linear_representation: LINEAR [NUMERIC]
                        -- Representation as a linear structure
```

```
                -- (from ARRAY)

        to_c: ANY
                -- Address of actual sequence of values,
                -- for passing to external (non-Eiffel) routines.
                -- (from ARRAY)

feature -- Duplication

        copy (other: like Current)
                -- Reinitialize by copying all the items of other.
                -- (This is also used by clone.)
                -- (from ARRAY)
            require -- from GENERAL
                other_not_void: other /= void;
                type_identity: same_type (other)
            ensure -- from GENERAL
                is_equal: is_equal (other)
            ensure then -- from ARRAY
                equal_areas: area.is_equal (other.area)

        subarray (start_pos, end_pos: INTEGER): like Current
                -- Array made of items of current array within
                -- bounds start_pos and end_pos.
                -- (from ARRAY)
            require -- from ARRAY
                valid_start_pos: valid_index (start_pos);
                valid_end_pos: valid_index (end_pos);
                valid_bounds: (start_pos <= end_pos) or (start_pos = end_pos + 1)
            ensure -- from ARRAY
                lower: Result.lower = start_pos;
                upper: Result.upper = end_pos

feature -- Creation

        make (n: INTEGER)
                -- Create structure for initial
                -- estimate of n times.
            require
                n_not_void: n /= void

invariant

                -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);
        last_non_negative: last >= 0;
        last_small_enough: last <= count;
                -- from ARRAY
```

consistent_size: *capacity = upper - lower + 1*;
non_negative_count: *count >= 0*;
-- from *RESIZABLE*
increase_by_at_least_one: *minimal_increase >= 1*;
-- from *BOUNDED*
valid_count: *count <= capacity*;
full_definition: *full* = (*count = capacity*);
-- from *FINITE*
empty_definition: *empty* = (*count = 0*);
non_negative_count: *count >= 0*;

end -- class *TIME_SET*

6.1.18. *TIME_UTILITY*

indexing
description: "*functions usefull in time calculations*"
status: "*See notice at end of class*"
date: "*$Date: 1998/03/10 17:00:01 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface
TIME_UTILITY

feature -- Access

date_time_tools: *DATE_TIME_TOOLS*

default_format_string: *STRING*

feature -- Basic operations

div (*i, j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
ensure
result_definition: *i = j * Result + mod* (*i, j*)

mod (*i, j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
ensure
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i, j*) + *Result*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);

reflexive_conformance: *conforms_to* (*Current*);

end -- class *TIME_UTILITY*

6.1.19. *TIME_VALUE*

indexing

description: "*time value in a day*"
status: "*See notice at end of class*"
date: "*$Date: 1998/04/01 17:37:11 $*"
revision: "*$Revision: 4.2 $*"
access: *date, time*

class interface

TIME_VALUE

creation

feature -- Access

compact_time: *INTEGER*
-- Hour, minute, second coded.

date_time_tools: *DATE_TIME_TOOLS*
-- (from *TIME_UTILITY*)

default_format_string: *STRING*
-- (from *TIME_UTILITY*)

fine_second: *DOUBLE*
-- Representation of second with decimals

fractionnal_second: *DOUBLE*
-- Fractionnal part of *fine_second*

hour: *INTEGER*
-- Hour of the current time

Hours_in_day: *INTEGER* ***is*** *24*
-- Number of hours in a day
-- (from *TIME_CONSTANTS*)

micro_second: *INTEGER*
-- Microsecond of the current time

milli_second: *INTEGER*
-- Millisecond of the current time

minute: *INTEGER*
-- Minute of the current time

Minutes_in_hour: *INTEGER* ***is*** *60*
-- Number of minutes in an hour
-- (from *TIME_CONSTANTS*)

nano_second: *INTEGER*
-- Nanosecond of the current time

second: *INTEGER*
-- Second of the current time

Seconds_in_day: *INTEGER* ***is*** *86400*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_hour: *INTEGER* ***is*** *3600*
-- Number of seconds in an hour
-- (from *TIME_CONSTANTS*)

Seconds_in_minute: *INTEGER* ***is*** *60*
-- Number of seconds in a minute
-- (from *TIME_CONSTANTS*)

time_default_format_string: *STRING*
-- (from *TIME_CONSTANTS*)

feature -- Basic operations

div (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + 1) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
result_definition: *i = j * Result + mod* (*i*, *j*)

mod (*i*, *j*: *INTEGER*): *INTEGER*
-- (i \\ j) if *i* positive
-- (i \\ j + j) if *i* negative
-- (from *TIME_UTILITY*)
ensure -- from *TIME_UTILITY*
positive_result: *Result >= 0*;
result_definition: *i = j * div* (*i*, *j*) *+ Result*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *TIME_VALUE*

6.2. Cluster: time_format

6.2.1. *DATE_TIME_CODE*

indexing

description: "*Code used by the DATE/TIME to STRING conversion*"
date: "*$Date: 1998/04/14 13:00:00 $*"
revision: "*$Revision: 4.2 $*"

class interface

DATE_TIME_CODE

creation

make (*v*: *STRING*)
-- Create code.
require
v_exists: *v* /= *void*;
v_is_code: *is_code* (*v*)
ensure
value_set: *value.is_equal* (*v*)

feature -- Attributes

count_max: *INTEGER*
-- Count max of the real value

count_min: *INTEGER*
-- Count min of the real value

name: *STRING*
-- Name of the code

type: *INTEGER*
-- Type number.

value: *STRING*
-- String code

value_max: *INTEGER*
-- Max of the real value

value_min: *INTEGER*
-- Min of the real value

feature -- Change

set_value (*v*: *STRING*)
-- Set all the attributes such as
-- Value, count_max, etc.

```
            require
                v_exists: v /= void;
                v_is_code: is_code (v)
            ensure
                value_set: value.is_equal (v)

feature -- Checking

    is_code (s: STRING): BOOLEAN
                -- Is the string a code?
            require
                s_exists: s /= void

    is_colon (s: STRING): BOOLEAN
                -- Is the code a separator-colomn?
            require
                s_exists: s /= void

    is_comma (s: STRING): BOOLEAN
                -- Is the code a separator-coma?
            require
                s_exists: s /= void

    is_day (s: STRING): BOOLEAN
                -- Is the code a day-numeric?
            require
                s_exists: s /= void

    is_day0 (s: STRING): BOOLEAN
                -- Is the code a day-numeric
                -- Padded with zero?
            require
                s_exists: s /= void

    is_day_text (s: STRING): BOOLEAN
                -- Is the code a day-text?
            require
                s_exists: s /= void

    is_dot (s: STRING): BOOLEAN
                -- Is the code a separator-dot?
            require
                s_exists: s /= void

    is_fractional_second (s: STRING): BOOLEAN
                -- Is the code a fractional-second
                -- With precision to n figures?
            require
                s_exists: s /= void
```

is_hour (*s*: *STRING*): *BOOLEAN*
-- Is the code a 24-hour-clock-scale?
require
s_exists: *s* /= *void*

is_hour0 (*s*: *STRING*): *BOOLEAN*
-- Is the code a 24-hour-clock-scale
-- Padded with zero?
require
s_exists: *s* /= *void*

is_hour12 (*s*: *STRING*): *BOOLEAN*
-- Is the code a 12-hour-clock-scale?
require
s_exists: *s* /= *void*

is_minus (*s*: *STRING*): *BOOLEAN*
-- Is the code a separator-minus?
require
s_exists: *s* /= *void*

is_minute (*s*: *STRING*): *BOOLEAN*
-- Is the code a minute-numeric?
require
s_exists: *s* /= *void*

is_minute0 (*s*: *STRING*): *BOOLEAN*
-- Is the code a minute-numeric
-- Padded with zero?
require
s_exists: *s* /= *void*

is_month (*s*: *STRING*): *BOOLEAN*
-- Is the code a month-numeric?
require
s_exists: *s* /= *void*

is_month0 (*s*: *STRING*): *BOOLEAN*
-- Is the code a month-numeric
-- Padded with zero?
require
s_exists: *s* /= *void*

is_month_text (*s*: *STRING*): *BOOLEAN*
-- Is the code a month-text?
require
s_exists: *s* /= *void*

is_numeric: *BOOLEAN*
-- Has the code a numeric value?

is_second (*s*: *STRING*): *BOOLEAN*
-- Is the code a second-numeric?
require
s_exists: *s* /= *void*

is_second0 (*s*: *STRING*): *BOOLEAN*
-- Is the code a second-numeric
-- Padded with zero?
require
s_exists: *s* /= *void*

is_slash (*s*: *STRING*): *BOOLEAN*
-- Is the code a separator-slash?
require
s_exists: *s* /= *void*

is_space (*s*: *STRING*): *BOOLEAN*
-- Is the code a separator-space?
require
s_exists: *s* /= *void*

is_text: *BOOLEAN*
-- Has the code a string value?

is_year2 (*s*: *STRING*): *BOOLEAN*
-- Is the code a year-numeric
-- On 2 figures?
require
s_exists: *s* /= *void*

is_year4 (*s*: *STRING*): *BOOLEAN*
-- Is the code a year-numeric
-- On 4 figures?
require
s_exists: *s* /= *void*

feature -- Creation

make (*v*: *STRING*)
-- Create code.
require
v_exists: *v* /= *void*;
v_is_code: *is_code* (*v*)
ensure
value_set: *value*.*is_equal* (*v*)

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_CODE*

6.2.2. *DATE_TIME_CODE_STRING*

indexing
description: "*DATE/TIME to STRING conversion*"
date: "*$Date: 1998/04/14 13:00:00 $*"
revision: "*$Revision: 4.2 $*"

class interface
DATE_TIME_CODE_STRING

creation

make (*s*: *STRING*)
-- Create code descriptors and hash-table.
require
s_exists: *s* /= *void*
ensure
value_set: *value* /= *void*

feature -- Attributes

name: *STRING*
-- Name of the code string.

value: *HASH_TABLE* [*DATE_TIME_CODE*, *INTEGER*]
-- Hash-table representing the code string.

feature -- Creation

make (*s*: *STRING*)
-- Create code descriptors and hash-table.
require
s_exists: *s* /= *void*
ensure
value_set: *value* /= *void*

feature -- Interface

correspond (*s*: *STRING*): *BOOLEAN*
-- Does the user string correspond to the code string?
require
s_exists: *s* /= *void*

```
        create_date (s: STRING): DATE
                        -- Create a DATE according to the string s format
                require
                        s_exists: s /= void;
                        is_precise: precise_date;
                        s_correspond: correspond (s)
                ensure
                        date_exists: Result /= void;
;
                        date_correspond: create_date_string (Result).is_equal (s)

        create_date_string (date: DATE): STRING
                        -- Create the output of date according to the code string.
                require
                        date_exists: date /= void
                ensure
                        string_exists: Result /= void;
                        string_correspond: correspond (Result)

        create_date_time (s: STRING): DATE_TIME
                        -- Create the DATE_TIME according to s.
                require
                        s_exist: s /= void;
                        is_precise: precise;
                        s_correspond: correspond (s)
                ensure
                        date_time_exists: Result /= void;
;
                        date_time_correspond: create_string (Result).is_equal (s)

        create_string (date_time: DATE_TIME): STRING
                        -- Create the output of date_time according to the code string.
                require
                        date_time /= void
                ensure
                        string_exists: Result /= void;
                        string_correspond: correspond (Result)

        create_time (s: STRING): TIME
                        -- Create a TIME according to the string s format.
                require
                        s_exists: s /= void;
                        is_precise: precise_time;
                        s_correspond: correspond (s)
                ensure
                        time_exists: Result /= void;
                        time_correspond: create_time_string (Result).is_equal (s)
```

create_time_string (*time*: *TIME*): *STRING*
-- Create the output of time according to the code string.
require
time_exists: *time* /= *void*
ensure
string_exists: *Result* /= *void*;
string_correspond: *correspond* (*Result*)

precise: *BOOLEAN*
-- Is the code string enough precise to create
-- An instance of type DATE_TIME?
require
not_void: *value* /= *void*

precise_date: *BOOLEAN*
-- Is the code string enough precise to create
-- An instance of type DATE?
require
not_void: *value* /= *void*

precise_time: *BOOLEAN*
-- Is the code string enough precise to create
-- An instance of type TIME?
require
not_void: *value* /= *void*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_CODE_STRING*

6.2.3. *DATE_TIME_LANGUAGE_CONSTANTS*

indexing
description: "*Language settings*"
date: "*$Date: 1998/04/14 13:00:00 $*"
revision: "*$Revision: 4.2 $*"

deferred class interface
DATE_TIME_LANGUAGE_CONSTANTS

feature

date_default_format_string: *STRING*
-- Standard output of the date.

days_text: *ARRAY* [*STRING*]

-- Array of days in short format.

default_format_string: *STRING*
-- Standard output of the date and time.

long_days_text: *ARRAY* [*STRING*]
-- Array of days in long format.

long_months_text: *ARRAY* [*STRING*]
-- Array of monthes in long format.

months_text: *ARRAY* [*STRING*]
-- Array of monthes in short format.

name: *STRING*
-- Language

time_default_format_string: *STRING*
-- Standard output of the time.

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_LANGUAGE_CONSTANTS*

6.3. Cluster: time_english

6.3.1. *DATE_TIME_TOOLS*

indexing
description: "*English settings*"
date: "*$Date: 1998/04/14 13:00:00 $*"
revision: "*$Revision: 4.2 $*"

class interface
DATE_TIME_TOOLS

feature

Date_default_format_string: *STRING* ***is*** "*[0]mm/[0]dd/yyyy*"

days_text: *ARRAY* [*STRING*]

Default_format_string: *STRING* ***is*** "*[0]mm/[0]dd/yyyy hh12:[0]mi:[0]ss.ff3*"

long_days_text: *ARRAY* [*STRING*]

long_months_text: *ARRAY* [*STRING*]

months_text: *ARRAY* [*STRING*]

Name: *STRING* ***is*** *"English"*

Time_default_format_string: *STRING* ***is*** *"hh12:[0]mi:[0]ss.ff3"*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_TOOLS*

6.4. Cluster: time_german

6.4.1. *DATE_TIME_TOOLS*

indexing

description: "*German settings*"
date: "*$Date: $*"
revision: "*$Revision: $*"

class interface

DATE_TIME_TOOLS

feature

Date_default_format_string: *STRING* ***is*** *"[0]dd/[0]mm/yyyy"*

days_text: *ARRAY* [*STRING*]

Default_format_string: *STRING* ***is*** *"[0]dd/[0]mm/yyyy [0]hh:[0]mi:[0]ss.ff3"*

long_days_text: *ARRAY* [*STRING*]

long_months_text: *ARRAY* [*STRING*]

months_text: *ARRAY* [*STRING*]

Name: *STRING* ***is*** *"German"*

Time_default_format_string: *STRING* ***is*** *"[0]hh:[0]mi:[0]ss.ff3"*

invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);

reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_TOOLS*

6.5. Cluster: time_french

6.5.1. *DATE_TIME_TOOLS*

indexing

description: "*French settings*"
date: "*$Date: $*"
revision: "*$Revision: $*"

class interface

DATE_TIME_TOOLS

feature

Date_default_format_string: *STRING* ***is*** "*[0]dd/[0]mm/yyyy*"

days_text: *ARRAY* [*STRING*]

Default_format_string: *STRING* ***is*** "*[0]dd/[0]mm/yyyy [0]hh:[0]mi:[0]ss.ff3*"

long_days_text: *ARRAY* [*STRING*]

long_months_text: *ARRAY* [*STRING*]

months_text: *ARRAY* [*STRING*]

Name: *STRING* ***is*** "*French*"

Time_default_format_string: *STRING* ***is*** "*[0]hh:[0]mi:[0]ss.ff3*"

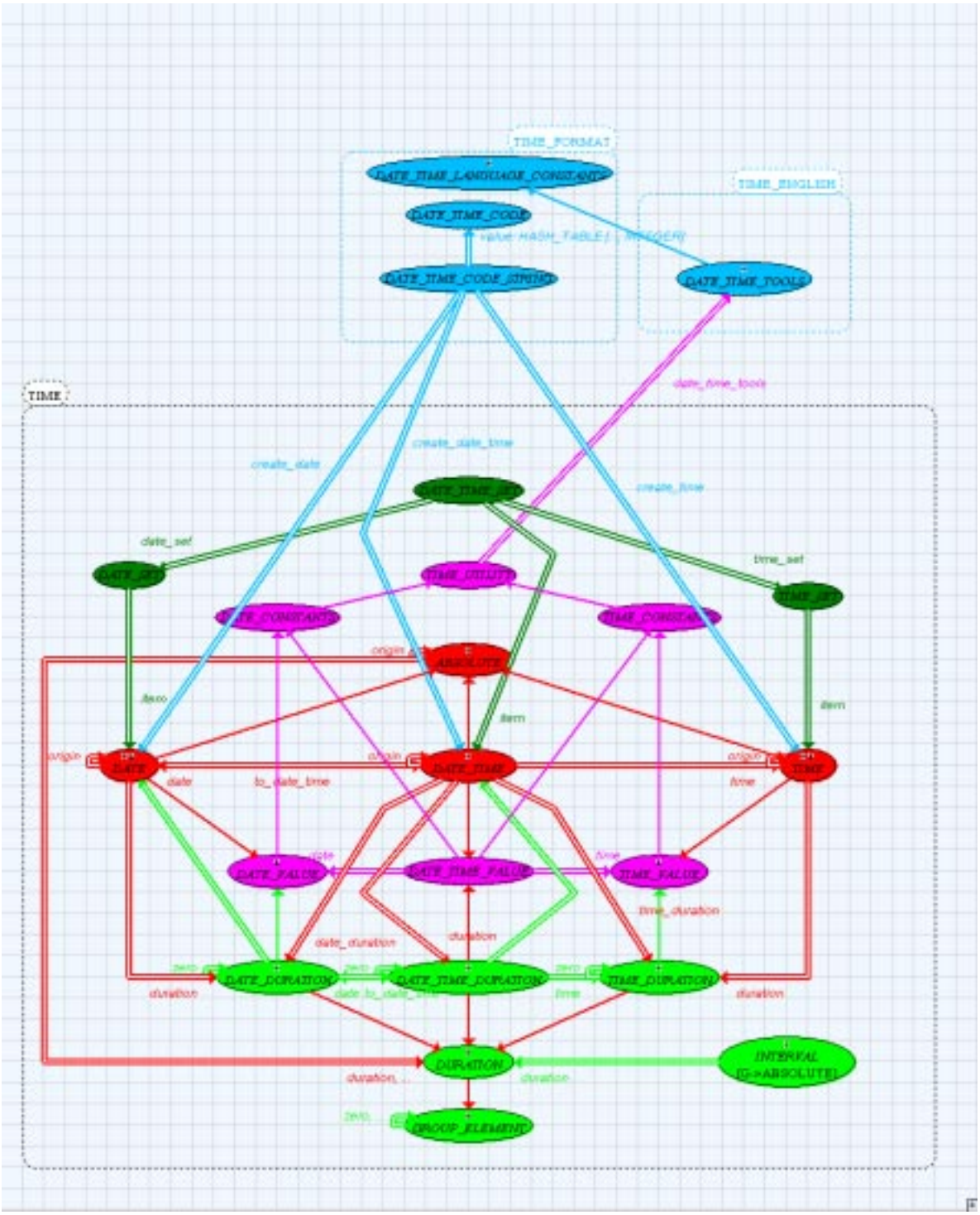
invariant

-- from *GENERAL*
reflexive_equality: *standard_is_equal* (*Current*);
reflexive_conformance: *conforms_to* (*Current*);

end -- class *DATE_TIME_TOOLS*

APPENDIX

A.The inheritance tree.



B. Compressed format.