
AIMMS Language Reference - Time-Based Modeling

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com or order your hard-copy at www.lulu.com/aimms.

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

| | | |
|----------------------------------|----------------------------------|----------------------------------|
| Paragon Decision Technology B.V. | Paragon Decision Technology Inc. | Paragon Decision Technology Pte. |
| Schipholweg 1 | 500 108th Avenue NE | Ltd. |
| 2034 LS Haarlem | Ste. # 1085 | 80 Raffles Place |
| The Netherlands | Bellevue, WA 98004 | UOB Plaza 1, Level 36-01 |
| Tel.: +31 23 5511512 | USA | Singapore 048624 |
| Fax: +31 23 5511517 | Tel.: +1 425 458 4024 | Tel.: +65 9640 4182 |
| | Fax: +1 425 458 4025 | |

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 31

Time-Based Modeling

In AIMMS there are three fundamental building blocks for time-based modeling namely *horizons*, *calendars* and *timetable-based aggregation and disaggregation*. These concepts coincide with your natural view of time, but there are associated details that need to be examined. Using these building blocks, you can develop time-dependent model-based applications with substantially less effort than would otherwise be required.

This chapter

31.1 Introduction

Time plays an important role in various real-life modeling applications. Typical examples are found in the areas of planning, scheduling, and control. The time scale in control models is typically seconds and minutes. Scheduling models typically refer to hours and days, while the associated time unit in planning models is usually expressed in terms of weeks, months, or even years. To facilitate time-based modeling, AIMMS provides a number of tools to relate model time and calendar time.

Time and models

Time-dependent data in a model is usually associated with time periods. Some data items associated with a period index can be interpreted as taking place *during* the period, while others take place *at a particular moment*. For instance, the stock in a tank is usually measured at, and associated with, a specific moment in a period, while the flow of material into the tank is usually associated with the entire period.

Use of time periods

Time-dependent data in a model can also represent continuous time values. For instance, consider a parameter containing the starting times of a number of processes. Even though this representation is not ideal for constructing most time-based optimization models, it allows time to be expressed to any desired accuracy.

Use of time as a continuous quantity

A large portion of the data in time-dependent models originates from the real world where quantities are specified relative to some calendar. Optimization models usually refer to abstract model periods such as p_1 , p_2 , p_3 , etc., allowing the optimization model to be formulated independent of real time. This common distinction makes it essential that quantities associated with real calendar time can be converted to quantities associated with model periods and vice versa.

Calendar periods versus model periods

In many planning and scheduling applications, time-dependent models are solved repeatedly as time passes. Future data becomes present data and eventually becomes past data. Such a moving time span is usually referred to as a “rolling horizon”. By using the various features discussed in this chapter, it is fairly straightforward to implement models with a rolling horizon.

Rolling horizon

AIMMS offers two special data types for time-based modeling applications, namely CALENDAR and HORIZON. Both are index sets with special features for dealing with time. CALENDARS allow you to create a set of time slots of fixed length in real time, while HORIZONS enable you to distinguish past, planning and beyond periods in your model.

CALENDARS and HORIZONS

In addition, AIMMS offers support for automatically creating *timetables* (represented through indexed sets) which link model periods in a HORIZON to time slots in a CALENDAR in a flexible manner. Based on a timetable, AIMMS provides functions to let you *aggregate* data defined over a CALENDAR to data defined over the corresponding HORIZON and vice versa. Figure 31.1 illustrates an example of a timetable relating a horizon and a calendar.

Timetables

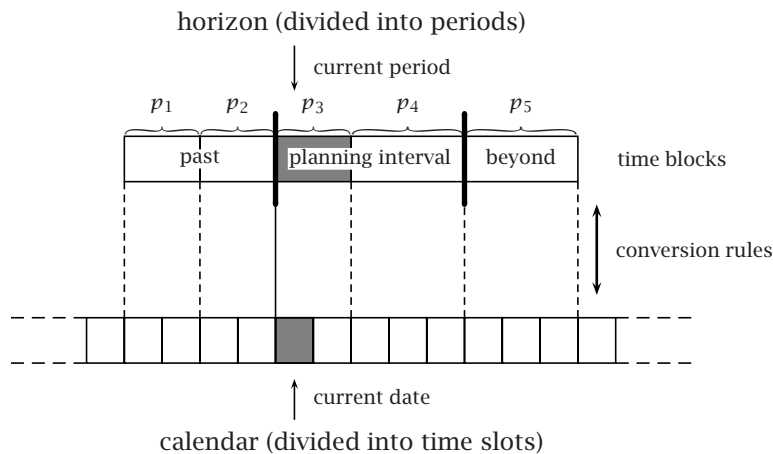


Figure 31.1: Timetable relating calendar and horizon

The horizon consists of periods divided into three time blocks, namely a past, the planning interval, and beyond. There is a current period in the horizon which can be linked to a current date in the calendar. The calendar consists of time slots and its range is defined by a begin date and an end date. When you construct your mathematical program, it will typically be in terms of periods in the planning interval of the horizon. However, the input data of the model will typically be in terms of calendar periods. The conversion of calendar data into horizon data and vice versa is done on request by AIMMS in accordance with pre-specified conversion rules.

Explanation

31.2 Calendars

A *calendar* is defined as a set of consecutive *time slots* of unit length covering the complete time frame from the calendar's begin date to its end date. You can use a calendar to index data defined in terms of calendar time.

Calendars

Calendars have several associated attributes, which are listed in Table 31.1. Some of these attributes are inherited from sets, while others are new and specific to calendars. The new ones are discussed in this section.

Calendar attributes

| Attribute | Value-type | See also page | Mandatory |
|-----------------|--|--|-----------|
| BEGIN DATE | <i>string</i> | | yes |
| END DATE | <i>string</i> | | yes |
| UNIT | <i>unit</i> | | yes |
| TIMESLOT FORMAT | <i>string</i> | | yes |
| INDEX PARAMETER | <i>identifier-list</i> <i>identifier-list</i> | 31 31 | yes |
| TEXT COMMENT | <i>string</i> <i>comment string</i> | 19 19 | |

Table 31.1: Calendar attributes

The UNIT attribute defines the length of a single time slot in the calendar. It must be specified as one of the following time units or an integer multiple thereof:

Unit

- century,
- year,
- month,
- day,
- hour,
- minute,

- second, and
- tick (i.e. sec/100).

Thus, 15*min and 3*month are valid time units, but the equivalent 0.25*hour and 0.25*year are not. Besides a constant integer number it is also allowed to use an AIMMS parameter to specify the length of the time slots in the calendar (e.g. NumberOfMinutesPerTimeslot*min).

Although you can only use the fixed unit names listed above to specify the UNIT attribute of a calendar, AIMMS does not have a predefined QUANTITY for time (see also Chapter 30). This means that the units of time you want to use in your model, do not have to coincide with the time units required in the calendar declaration. Therefore, prior to specifying the UNIT attribute of a calendar, you must first specify a quantity defining both your own time units and the conversion factors to the time units required by AIMMS. In the **Model Explorer**, AIMMS will automatically offer to add the relevant time QUANTITY to your model when the calendar unit does not yet exist in the model tree.

Not predefined

The mandatory BEGIN DATE and END DATE attributes of a calendar specify its range. AIMMS will generate all time slots of the specified length, whose *begin time* lies between the specified BEGIN and END DATE. As a consequence, the *end time* of the last time slot may be after the specified END DATE. An example of this behavior occurs, for instance, when the requested length of all time slots is 3 days and the END DATE does not lie on a 3-day boundary from the BEGIN DATE. Any period references that start outside this range will be ignored by the system. This makes it easy to select all relevant time-dependent data from a database.

Begin and end date

Any set element describing either the BEGIN DATE or the END DATE must be given in the following fixed *reference date* format which contains the specific year, month, etc. up to and including the appropriate reference to the time unit associated with the calendar.

Reference date format

YYYY-MM-DD hh:mm:ss

All entries must be numbers *with leading zeros present*. The hours are expressed using the 24-hour clock. You do not need to specify all entries. Only those fields that refer to time units that are longer or equal to the predefined AIMMS time unit in your calendar are required. All time/date fields beyond the requested granularity are ignored. For instance, a calendar expressed in hours may have a BEGIN DATE such as

- "1996-01-20 09:00:00", or
- "1996-01-20 09:00", or
- "1996-01-20 09",

which all refer to exactly the same time, 9:00 AM on January 20th, 1996.

AIMMS always assumes that reference dates are specified according to the local time zone without daylight saving time. However, for calendars with granularity day AIMMS will ignore any timezone and daylight saving time offsets, and just take the day as specified. In the example above, a daily calendar with the above BEGIN DATE will always start with period "1996-01-20", while an hourly calendar may start with a period "1996-01-19 23:00" if the difference between the local time zone, and the time zone specification in the timeslot format is 10 hours.

Time zone and DST offsets

Set elements and string-valued parameters capturing time-related information must deal with a variety of formatting possibilities in order to meet end-user requirements around the globe (there are no true international standards for formatting time slots and time periods). The flexible construction of dates and date formats using the TIMESLOT FORMAT is presented in Section 31.7.

Format of time-related attributes

The following example is a declaration of a daily calendar and a monthly calendar

Example

```

CALENDAR:
  identifier      : DailyCalendar
  index          : d
  parameter      : CurrentDay
  text           : A work-week calendar for production planning
  begin date     : "1996-01-01"
  end date       : "1997-06-30"
  unit           : day
  timeslot format : "%d/%m/%y" ;      ! format explained later

CALENDAR:
  identifier      : MonthlyCalendar
  index          : m
  begin date     : CalendarBeginMonth
  end date       : CalendarEndMonth
  unit           : month
  timeslot format : "%m/%y" ;        ! format explained later

```

The calendar DailyCalendar thus declared will be a set containing the elements '01/01/96', ..., '06/30/97' for every day in the period from January 1, 1996 through June 30, 1997. When the BEGIN and END DATE attributes are specified as string parameters containing the respective begin and end dates (as in MonthlyCalendar), the number of generated time slots can be changed dynamically. In order to generate zero time slots, leave one of these string parameters empty.

Varying number of time slots

By default, AIMMS assumes that a calendar uses the local time zone without daylight saving time, in accordance with the specification of the BEGIN and END DATE attributes. However, if this is not the case, you can modify the TIMESLOT FORMAT attribute in such a manner, that AIMMS

Time zones and daylight saving time

- will take daylight saving time into account during the construction of the calendar slots, or,
- will generate the calendar slots according to a specified time zone.

In both cases, AIMMS still requires that the BEGIN and END DATE attributes be specified as reference dates in the local time zone without daylight saving time, as already indicated. Support for time zones and daylight saving time is explained in full detail in Section [31.7.4](#).

31.3 Horizons

A horizon in AIMMS is basically a set of planning periods. The elements in a horizon are divided into three groups, also referred to as time blocks. The main group of elements comprise the *planning interval*. Periods prior to the planning interval form the *past*, while periods following the planning interval form the *beyond*. When variables and constraints are indexed over a horizon, AIMMS automatically restricts the generation of these constraints and variables to periods within the planning interval.

Horizons

Whenever you use a horizon to construct a time-dependent model, AIMMS has the following features:

Effect on constraints and assignments

- constraints are excluded from the past and beyond periods,
- variables are assumed to be fixed for these periods, and
- assignments and definitions to variables and parameters are, by default, only executed for the periods in the planning interval.

Horizons, like calendars, have a number of associated attributes, some of which are inherited from sets. They are summarized in Table [31.2](#).

Horizon attributes

| Attribute | Value-type | See also page | Mandatory |
|---|---|--|-----------|
| SUBSET OF INDEX PARAMETER TEXT COMMENT DEFINITION | <i>subset-domain identifier-list identifier-list string comment string set-expression</i> | 31 31 31 19 19 33 | yes |
| CURRENT PERIOD INTERVAL LENGTH | <i>element integer-reference</i> | | yes |

Table 31.2: Horizon attributes

The CURRENT PERIOD attribute denotes the first period of the planning interval. The periods prior to the current period belong to the past. The integer value associated with the attribute INTERVAL LENGTH determines the number of periods in the planning interval (including the current period). Without an input value, all periods from the current period onwards are part of the planning interval.

*Horizon
attributes
explained*

AIMMS requires you to specify the contents of a HORIZON uniquely through its DEFINITION attribute. The ordering of the periods in the horizon is fully determined by the set expression in its definition. You still have the freedom, however, to specify the HORIZON as a subset of another set.

*Definition is
mandatory*

Given a scalar parameter MaxPeriods, the following example illustrates the declaration of a horizon.

Example

```
HORIZON:
  identifier      : ModelPeriods
  index          : h
  parameter      : IntervalStart
  current period : IntervalStart
  definition      : ElementRange( 1, MaxPeriods, prefix: "p-" ) ;
```

If, for instance, the scalar MaxPeriods equals 10, this will result in the creation of a period set containing the elements 'p-01', ..., 'p-10'. The start of the planning interval is fully determined by the value of the element parameter IntervalStart, which for instance could be equal to 'p-02'. This will result in a planning interval consisting of the periods 'p-02', ..., 'p-10'.

Consider the parameter Demand(h) together with the variables Production(h) and Stock(h). Then the definition of the variable Stock can be declared as follows.

Example of use

```
VARIABLE:
  identifier      : Stock
  index domain    : h
  range          : NonNegative
  definition      : Stock(h-1) + Production(h) - Demand(h) ;
```

When the variable Stock is included in a mathematical program, AIMMS will only generate individual variables with their associated definition for those values of h that correspond to the current period and onwards. The reference Stock(h-1) refers to a fixed value of Stock from the past whenever the index h points to the current period. The values associated with periods from the past (and from the beyond if they were there) are assumed to be fixed.

To provide easy access to periods in the past and the beyond, AIMMS offers three horizon-specific suffices. They are:

Accessing past and beyond

- the Past suffix,
- the Planning suffix, and
- the Beyond suffix.

These suffices provide access to the subsets of the horizon representing the past, the planning interval and the beyond.

When you use a horizon index in an index binding operation (see Chapter 9), AIMMS will, by default, perform that operation only for the periods in the planning interval. You can override this default behavior by a local binding using the suffices discussed above.

Horizon binding rules

Consider the horizon `ModelPeriods` of the previous example. The following assignments illustrate the binding behavior of horizons.

Example

```
Demand(h) := 10; ! only periods in planning interval (default)
Demand(h in ModelPeriods.Planning) := 10; ! only periods in planning interval

Demand(h in ModelPeriods.Past) := 10; ! only periods in the past
Demand(h in ModelPeriods.Beyond) := 10; ! only periods in the beyond

Demand(h in ModelPeriods) := 10; ! all periods in the horizon
```

When you use one of the lag and lead operators `+`, `++`, `-` or `--` (see also Section 5.2.4) in conjunction with a horizon index, AIMMS will interpret such references with respect to the *entire* horizon, and not just with respect to the planning period. If the horizon index is locally re-bound to one of the subsets of periods in the Past or Beyond, as illustrated above, the lag or lead operation will be interpreted with respect to the specified subset.

Use of lag and lead operators

Consider the horizon `ModelPeriods` of the previous example. The following assignments illustrate the use of lag and lead operators in conjunction with horizons.

Example

```
Stock(h) := Stock(h-1) + Supply(h) - Demand(h);
Stock(h | h in ModelPeriods.Planning) := Stock(h-1) + Supply(h) - Demand(h);

Stock(h in ModelPeriods.Planning) := Stock(h-1) + Supply(h) - Demand(h);
Stock(h in ModelPeriods.Planning) := Stock(h--1) + Supply(h) - Demand(h);
```

The first two assignments are completely equivalent (in fact, the second assignment is precisely the way in which AIMMS interprets the default binding behavior of a horizon index). For the first element in the planning interval, the reference `h-1` refers to the last element of the past interval. In the third assignment, `h-1` refers to a non-existing element for the first element in the

planning interval, completely in accordance with the default semantics of lag and lead operators. In the fourth assignment, $h--1$ refers to the last element of the planning interval.

Operations which can be applied to identifiers without references to their indices (such as the READ, WRITE or DISPLAY statements), operate on the entire horizon domain. Thus, for example, during data transfer with a database, AIMMS will retrieve or store the data for *all* periods in the horizon, and not just for the periods in the planning interval.

*Data transfer
on entire
domain*

31.4 Creating timetables

A *timetable* in AIMMS is an indexed set, which, for every period in a HORIZON, lists the corresponding time slots in the associated CALENDAR. Timetables play a central role during the conversion from calendar data to horizon data and vice versa.

Timetables

Through the predefined procedure CreateTimeTable, you can request AIMMS to flexibly construct a *timetable* on the basis of

*The procedure
CreateTimeTable*

- a *time slot* in the calendar and a *period* in the horizon that should be aligned at the beginning of the planning interval,
- the desired *length* of each period in the horizon expressed as a number of time slots in the calendar,
- an indication, for every period in the horizon, whether the *length dominates* over any specified delimiter slots,
- a set of *inactive time slots*, which should be excluded from the timetable and, consequently, from the period length computation, and
- a set of *delimiter time slots*, at which new horizon periods should begin.

The syntax of the procedure CreateTimeTable is as follows:

Syntax

- CreateTimeTable(*timetable*, *current-timeslot*, *current-period*,
period-length, *length-dominates*,
inactive-slots, *delimiter-slots*)

The (output) *timetable* argument of the procedure CreateTimeTable must, in general, be an indexed set in a calendar and defined over the horizon to be linked to the calendar. Its contents is completely determined by AIMMS on the basis of the other arguments. The *current-timeslot* and *current-period* arguments must be elements of the appropriate calendar and horizon, respectively.

In the special case that you know a priori that each period in the timetable is associated with exactly one time slot in the calendar, AIMMS also allows the *timetable* argument of the `CreateTimeTable` procedure to be an element parameter (instead of an indexed set). When you specify an element parameter, however, a runtime error will result if the input arguments of the call to `CreateTimeTable` give rise to periods consisting of multiple time slots.

*Element
parameter as
timetable*

You have several possibilities of specifying your input data which influence the way in which the timetable is created. You can:

*Several
possibilities*

- only specify the length of each period to be created,
- only specify delimiter slots at which a new period must begin, or
- flexibly combine both of the above two methods.

The *period-length* argument must be a positive integer-valued one-dimensional parameter defined over the horizon. It specifies the desired length of each period in the horizon in terms of the number of time slots to be contained in it. If you do not provide delimiter slots (explained below), AIMMS will create a timetable solely on the basis of the indicated period lengths.

Period length

The *inactive-slots* argument must be a subset of the calendar that is specified as the range of the *timetable* argument. Through this argument you can specify a set of time slots that are always to be excluded from the timetable. You can use this argument, for instance, to indicate that weekend days or holidays are not to be part of a planning period. Inactive time slots are excluded from the timetable, and are not accounted for in the computation of the desired period length.

Inactive slots

The *delimiter-slots* argument must be a subset of the calendar that is specified as the range of the *timetable* argument. AIMMS will begin a new period in the horizon whenever it encounters a delimiter slot in the calendar provided no (offending) period length has been specified for the period that is terminated at the delimiter slot.

Delimiter slots

In addition to using either of the above methods to create a timetable, you can also combine them to create timetables in an even more flexible manner by specifying the *length-dominates* argument, which must be a one-dimensional parameter defined over the horizon. The following rules apply.

*Combining
period length
and delimiters*

- If the *length-dominates* argument is nonzero for a particular period, meeting the specified period length prevails over any delimiter slots that are possibly contained in that period.
- If the *length-dominates* argument is zero for a particular period and the specified period length is 0, AIMMS will not restrict that period on the basis of length, but only on the basis of delimiter slots.

- If the *length-dominates* argument is zero for a particular period and the specified period length is positive, AIMMS will try to construct a period of the indicated length, but will terminate the period earlier if it encounters a delimiter slot first.

In creating a timetable, AIMMS will always start by aligning the *current-timeslot* argument with the beginning of the *current-period*. Periods beyond *current-period* are determined sequentially by moving forward time slot by time slot, until a new period must be started due to hitting the period length criterion of the current period (taking into account the inactive slots), or by hitting a delimiter slot. Periods prior to *current-period* are determined sequentially by moving backwards in time starting at *current-timeslot*.

*Timetable
creation*

As a timetable is nothing more than an indexed set, you still have the opportunity to make manual changes to a timetable after its contents have been computed by the AIMMS procedure `CreateTimeTable`. This allows you to make any change to the timetable that you cannot, or do not want to, implement directly using the procedure `CreateTimeTable`.

*Adapting
timetables*

Consider a timetable which links the daily calendar declared in Section 31.2 and the horizon of Section 31.3, which consists of 10 periods named p-01 ... p-10. The following conditions should be met:

Example

- the planning interval starts at period p-02, i.e. period p-01 is in the past,
- periods p-01...p-05 have a fixed length of 1 day,
- periods p-06...p-10 should have a length of at most a week, with new periods starting on every Monday.

To create the corresponding timetable using the procedure `CreateTimeTable`, the following additional identifiers need to be added to the model:

- an indexed subset `TimeTable(h)` of `DailyCalendar`,
- a subset `DelimiterDays` of `DailyCalendar` containing all Mondays in the calendar (i.e. '01-01-96', '08-01-96', etc.),
- a subset `InactiveDays` of `DailyCalendar` containing all days that you want to exclude from the timetable (e.g. all weekend days),
- a parameter `PeriodLength(h)` assuming the value 1 for the periods p-01 ... p-05, and zero otherwise,
- a parameter `LengthDominates(h)` assuming the value 1 for the periods p-01 ... p-05, and zero otherwise.

To compute the contents of the timetable, aligning the time slot pointed at by `CurrentDay` and period `IntervalStart`, one should call

```
CreateTimeTable( TimeTable, CurrentDay, IntervalStart,
                 PeriodLength, LengthDominates,
                 InactiveDays, DelimiterDays );
```

If all weekend days are inactive, and CurrentDay equals '24/01/96' (a Wednesday), then TimeTable describes the following mapping.

| Period | Calendar slots | Period | Calendar slots |
|--------|----------------|--------|-------------------------------|
| p-01 | 23/01/96 (Tue) | p-06 | 30/01/96 - 02/02/96 (Tue-Fri) |
| p-02 | 24/01/96 (Wed) | p-07 | 05/01/96 - 09/02/96 (Mon-Fri) |
| p-03 | 25/01/96 (Thu) | p-08 | 12/01/96 - 16/02/96 (Mon-Fri) |
| p-04 | 26/01/96 (Fri) | p-09 | 19/01/96 - 23/02/96 (Mon-Fri) |
| p-05 | 29/01/96 (Mon) | p-10 | 26/01/96 - 01/03/96 (Mon-Fri) |

The process of initializing the sets used in the *delimiter-slots* and *inactive-slots* arguments can be quite cumbersome when your model covers a large time span. For that reason AIMMS offers the convenient function TimeslotCharacteristic. With it, you can obtain a numeric value which characterizes the time slot, in terms of its day of the week, its day in the year, etc. The syntax of the function is straightforward:

The function
Timeslot-
Characteristic

- TimeslotCharacteristic(timeslot, characteristic[, timezone[, ignoredst]])

The *characteristic* argument must be an element of the predefined set TimeslotCharacteristics. The elements of this set, as well as the associated function values are listed in Table 31.3.

| Characteristic | Function value range | First |
|----------------|----------------------|---------|
| century | 0, ..., 99 | |
| year | 0, ..., 99 | |
| quarter | 1, ..., 4 | |
| month | 1, ..., 12 | January |
| weekday | 1, ..., 7 | Monday |
| yearday | 1, ..., 366 | |
| monthday | 1, ..., 31 | |
| week | 1, ..., 53 | |
| weekyear | 0, ..., 99 | |
| weekcentury | 0, ..., 99 | |
| hour | 0, ..., 23 | |
| minute | 0, ..., 59 | |
| second | 0, ..., 59 | |
| tick | 0, ..., 99 | |
| dst | 0, 1 | |

Table 31.3: Elements of the set TimeslotCharacteristics

Internally, AIMMS takes Monday as the first day in a week, and considers week 1 as the first week that contains at least four days of the new year. This is equivalent to stating that week 1 contains the first Thursday of the new year. Through the 'week', 'weekyear' and 'weekcentury' characteristics you obtain the week number corresponding to a particular date and its corresponding year and century. For instance, Friday January 1, 1999 is day 5 of week 53 of year 1998.

Day and week numbering

Consider a daily calendar `DailyCalendar` with index `d`. The following assignment to a subset `WorkingDays` of a `DailyCalendar` will select all non-weekend days in the calendar.

Example

```
WorkingDays := { d | TimeslotCharacteristic(d,'weekday') <= 5 } ;
```

You can also use the function `TimeslotCharacteristic` to create a timetable linking two calendars (e.g. to create monthly overviews of daily data). As an example, consider the calendars `DailyCalendar` and `MonthlyCalendar` declared in Section 31.2, as well as an indexed set `MonthDays(m)` of `DailyCalendar`, which can serve as a timetable. `MonthDays` can be computed as follows.

Calendar-calendar linkage

```
MonthDays(m) := { d | TimeslotCharacteristic(d,'year') =  
                    TimeslotCharacteristic(m,'year') and  
                    TimeslotCharacteristic(d,'month') =  
                    TimeslotCharacteristic(m,'month')    } ;
```

A check on the 'year' characteristic is not necessary if both calendars are contained within a single calendar year.

Through the optional *timezone* argument of the function `TimeslotCharacteristic`, you can specify with respect to which time zone you want to obtain the specified characteristic. The *timezone* argument must be an element of the predefined set `AllTimeZones` (see also Section 31.7.4). By default, AIMMS assumes the local time zone without daylight saving time.

Time zone support

When you specify a time zone with daylight saving time, you can retrieve whether daylight saving time is active through the 'dst' characteristic. With the optional argument *ignoredst* (default 0) of the function `TimeslotCharacteristic`, you can specify whether you want daylight saving time to be ignored. With *ignoredst* set to 1, or in a time zone without daylight saving time, the outcome for the 'dst' characteristic will always be 0.

Daylight saving time

31.5 Data conversion of time-dependent identifiers

When you are working with time-dependent data, it is usually not sufficient to provide and work with a single fixed-time scale. The following examples serve as an illustration.

Time-dependent data

- Demand data is available in a database on a day-by-day basis, but is needed in a mathematical program for each horizon period.
- Production quantities are computed per horizon period, but are needed on a day-by-day basis.
- For all of the above data weekly or monthly overviews are also required.

With the procedures `Aggregate` and `Disaggregate` you can instruct AIMMS to perform an aggregation or disaggregation step from one time scale to another. Both procedures perform the aggregation or disaggregation of a single identifier in one time scale to another identifier in a second time scale, given a timetable linking both time scales and a predefined aggregation type. The syntax is as follows.

The procedures Aggregate and Disaggregate

- `Aggregate(timeslot-data, period-data, timetable, type[, locus])`
- `Disaggregate(period-data, timeslot-data, timetable, type[, locus])`

The identifiers (or identifier slices) passed to the `Aggregate` and `Disaggregate` procedures holding the time-dependent data must be of equal dimension. All domain sets in the index domains must coincide, except for the time domains. These must be consistent with the domain and range of the specified timetable.

Time slot and period data

As was mentioned in Section 31.1, time-dependent data can be interpreted as taking place *during* a period or *at a given moment* in the period. Calendar data, which takes place *during* a period, needs to be converted into a period-based representation by allocating the data values in proportion to the overlap between time slots and horizon periods. On the other hand, calendar data which takes place *at a given moment*, needs to be converted to a period-based representation by linearly interpolating the original data values.

Different conversions

The possible values for the *type* argument of the `Aggregate` and `Disaggregate` procedures are the elements of the predefined set `AggregationTypes` given by:

Aggregation types

- summation,
- average,
- maximum,
- minimum, and
- interpolation.

All of the above predefined conversion rules are characterized by the following property.

Reverse conversion

The disaggregation of period data into time slot data, followed by immediate aggregation, will reproduce identical values of the period data.

Aggregation followed by disaggregation does not have this property. Fortunately, as the horizon rolls along, disaggregation followed by aggregation is the essential conversion.

The conversion rule summation is the most commonly used aggregation/disaggregation rule for quantities that take place *during* a period. It is appropriate for such typical quantities as production and arrivals. Data values from a number of consecutive time slots in the calendar are summed together to form a single value for a multi-unit period in the horizon. The reverse conversion takes place by dividing the single value equally between the consecutive time slots.

The summation rule

The conversion rules average, maximum, and minimum are less frequently used aggregation/disaggregation rules for quantities that take place *during* a period. These rules are appropriate for such typical quantities as temperature or capacity. Aggregation of data from a number of consecutive time slots to a single period in the horizon takes place by considering the average or the maximum or minimum value over all time slots contained in the period. The reverse conversion consists of assigning the single value to each time slot contained in the period.

The average, maximum, and minimum rules

Table 31.4 demonstrates the aggregation and disaggregation taking place for each conversion rule. The conversion operates on a single period consisting of 3 time slots in the calendar.

Illustration of aggregation

| Conversion rule | Calendar to horizon | | | Horizon to calendar | | |
|-----------------|---------------------|---|---|---------------------|---|---|
| | 3 | 1 | 2 | 3 | | |
| summation | 6 | | | 1 | 1 | 1 |
| average | 2 | | | 3 | 3 | 3 |
| maximum | 3 | | | 3 | 3 | 3 |
| minimum | 1 | | | 3 | 3 | 3 |

Table 31.4: Conversion rules for “during” quantities

The interpolation rule should be used for all quantities that take place *at a given moment* in a period. For the interpolation rule you have to specify one additional argument in the Aggregate and Disaggregate procedures, the *locus*. The *locus* of the interpolation defines at which moment in a period—as a value between 0 and 1—the quantity at hand is to be measured. Thus, a *locus* of 0 means that the quantity is measured at the beginning of every period, a *locus* of 1 means that the quantity is measured at the end of every period, while a *locus* of 0.5 means that the quantity is measured midway through the period.

Interpolation

When disaggregating data from periods to time slots, AIMMS interpolates linearly between the respective loci of two subsequent periods. For the outermost periods, AIMMS assigns the last available interpolated value.

Interpolation for disaggregation

AIMMS applies a simple rule for the seemingly awkward interpolation of data from unit-length time slots to variable-length horizon periods. It will simply take the value associated with the time slot in which the locus is contained, and assign it to the period. This simple rule works well for loci of 0 and 1, which are the most common values.

Interpolation for aggregation

Table 31.5 demonstrates aggregation and disaggregation of a horizon of 3 periods, each consisting of 3 time slots, for loci of 0, 1, and 0.5. The underlined values are the values determined by the reverse conversion.

Illustration of interpolation

| Locus | Horizon data | | | | | | | | |
|-------|--------------|----------|----------|----------|----------|----------|----------|----------|----------|
| | 0 | | | 3 | | | 9 | | |
| 0 | <u>0</u> | 1 | 2 | <u>3</u> | 5 | 7 | <u>9</u> | 9 | 9 |
| 1 | 0 | 0 | <u>0</u> | 1 | 2 | <u>3</u> | 5 | 7 | <u>9</u> |
| 0.5 | 0 | <u>0</u> | 1 | 2 | <u>3</u> | 5 | 7 | <u>9</u> | 9 |

Table 31.5: Conversion rules for interpolated data

Consider the calendar `DailyCalendar`, the horizon `ModelPeriods` and the timetable `TimeTable` declared in Sections 31.2, 31.3 and 31.4, along with the identifiers

Example

- `DailyDemand(d)`,
- `Demand(h)`,
- `DailyStock(d)`, and
- `Stock(h)`.

The aggregation of `DailyDemand` to `Demand` can then be accomplished by the statement

```
Aggregate( DailyDemand, Demand, TimeTable, 'summation' );
```

Assuming that the Stock is computed at the end of each period, the disaggregation (by interpolation) to daily values is accomplished by the statement

```
Disaggregate( Stock, DailyStock, TimeTable, 'interpolation', locus: 1 );
```

If your particular aggregation/disaggregation scheme is not covered by the predefined aggregation types available in AIMMS, it is usually not too difficult to implement a custom aggregation scheme yourself in AIMMS. For instance, the aggregation by summation from DailyDemand to Demand can be implemented as

User-defined conversions

```
Demand(h) := sum( d in TimeTable(h), DailyDemand(d) );
```

while the associated disaggregation rule becomes the statement

```
DailyDemand(d) := sum( h | d in TimeTable(h), Demand(h)/Card(TimeTable(per)) );
```

31.6 Implementing a model with a rolling horizon

The term *rolling horizon* is used to indicate that a time-dependent model is solved repeatedly, and in which the planning interval is moved forward in time during each solution step. With the facilities introduced in the previous sections setting up such a model is relatively easy. This section outlines the steps that are required to implement a model with a rolling horizon, without going into detail regarding the contents of the underlying model.

Rolling horizons

In this section you will find two strategies for implementing a rolling horizon. One is a simple strategy that will only work with certain restrictions. It requires just a single aggregation step and a single disaggregation step. The other is a generic strategy that will work in all cases. This strategy, however, requires that aggregation and disaggregation steps be performed between every two subsequent SOLVE statements.

Two strategies

The simple strategy will work provided that

Simple strategy

- all periods in the horizon are of equal length, and
- the horizon rolls from period boundary to period boundary.

It is then sufficient to make the horizon sufficiently large so as to cover the whole time range of interest.

The algorithm to implement the rolling horizon can be outlined as follows.

*Algorithm
outline*

1. Select the current time slot and period, and create the global timetable.
2. Aggregate all calendar-based data into horizon-based data.
3. Solve the optimization model for a planning interval that is a subset of the complete horizon.
4. Move the current period to the next period boundary of interest, and repeat from steps until the time range of interest has passed.
5. Disaggregate the horizon-based solution into a calendar-based solution.

The examples below that illustrate both the simple and generic strategy make the following assumptions.

Assumptions

- The model contains the daily calendar `DailyCalendar`, the horizon `ModelPeriods` and the timetable `TimeTable` declared in Sections 31.2, 31.3 and 31.4, respectively.
- The model contains a time-dependent mathematical program `TimeDependentModel`, which produces a plan over the planning interval associated with `ModelPeriods`.
- The planning horizon, for which the model is to be solved, rolls along from `FirstWeekBegin` to `LastWeekBegin` in steps of one week. Both identifiers are element parameters in `DailyCalendar`.

The outline of the simple strategy can be implemented as follows.

Code outline

```

CurrentDay := FirstWeekBegin;
CreateTimeTable( TimeTable , CurrentDay , IntervalStart,
                PeriodLength, LengthDominates,
                InactiveDays, DelimiterDays );

Aggregate( DailyDemand, Demand, TimeTable, 'summation' );
! ... along with any other aggregation required

repeat
    solve TimeDependentModel;

    CurrentDay += 7;
    IntervalStart += 1;

    break when (not CurrentDay) or (CurrentDay > LastWeekBegin);
endrepeat;

Disaggregate( Stock , DailyStock , TimeTable, 'interpolation', locus: 1 );
Disaggregate( Production, DailyProduction, TimeTable, 'summation' );
! ... along with any other disaggregation required

```

The simple strategy will not work

Generic strategy

- whenever the lengths of periods in the horizon (expressed in time slots of the calendar) vary, or
- when the start of a new planning interval does not align with a future model period.

In both cases, the horizon-based solution obtained from a previous solve will not be accurate when you move the planning interval. Thus, you should follow a generic strategy which adds an additional disaggregation and aggregation step to every iteration.

The generic strategy for implementing a rolling horizon is outlined as follows.

Algorithm outline

1. Select the initial current time slot and period, and create the initial timetable.
2. Aggregate all calendar-based data into horizon-based data.
3. Solve the mathematical program.
4. Disaggregate all horizon-based variables to calendar-based identifiers.
5. Move the current time slot forward in time, and recreate the timetable.
6. Aggregate all identifiers disaggregated in step 4 back to the horizon using the updated timetable.
7. Repeat from step 2 until the time range of interest has passed.

The outline of the generic strategy can be implemented as follows.

Code outline

```

CurrentDay := FirstWeekBegin;
CreateTimeTable( TimeTable , CurrentDay , IntervalStart,
                 PeriodLength, LengthDominates,
                 InactiveDays, DelimiterDays );

repeat
  Aggregate( DailyDemand, Demand, TimeTable, 'summation' );
  ! ... along with any other aggregation required

  solve TimeDependentModel;

  Disaggregate( Stock , DailyStock , TimeTable, 'interpolation', locus: 1 );
  Disaggregate( Production, DailyProduction, TimeTable, 'summation' );
  ! ... along with any other disaggregation required

  CurrentDay += 7;
  break when (not CurrentDay) or (CurrentDay > LastWeekBegin);
  CreateTimeTable( TimeTable , CurrentDay , IntervalStart,
                 PeriodLength, LengthDominates,
                 InactiveDays, DelimiterDays );

  Aggregate( DailyStock , Stock , TimeTable, 'interpolation', locus: 1 );
  Aggregate( DailyProduction, Production, TimeTable, 'summation' );
  ! ... along with any other aggregation required
endrepeat;

```

31.7 Format of time slots and periods

While the BEGIN and END DATE attributes have to be specified using the fixed *reference date* format (see Section 31.2), AIMMS provides much more flexible formatting capabilities to describe

Flexible time slot and period formats

- time slots in a CALENDAR consisting of a single basic time unit (e.g. 1-day time slots),
- time slots in a CALENDAR consisting of multiple basic time units (e.g. 3-day time slots), and
- periods in a timetable consisting of multiple time slots.

The formatting capabilities described in this section are quite extensive, and allow for maximum flexibility.

In the Model Explorer, AIMMS provides a wizard to support you in constructing the appropriate formats. Through this wizard, you can not only select from a number of predefined formats (including some that use the regional settings of your computer), you also have the possibility of constructing a custom format, observing the result as you proceed.

Wizard support

AIMMS offers both a *basic* and an *extended format* for the description of time slots and periods. The basic format only refers to the beginning of a time slot or period. The extended format allows you to refer to both the first and last basic time unit contained in a time slot or period. Both the basic and extended formats are constructed according to the same rules.

Basic and extended format

The TIMESLOT FORMAT used in a CALENDAR must contain a reference to either its beginning, its end, or both. As the specified format is used to identify calendar elements when reading data from external data sources such as files and databases, you have to ensure that the specified format contains sufficient date and time references to uniquely identify each time slot in a calendar.

Care is needed

For instance, the description “January 1” is sufficient to uniquely identify a time slot in a calendar with a range of one year. However, in a two-year calendar, corresponding days in the first and second year are identified using exactly the same element description. In such a case, you must make sure that the specified format contains a reference to a year.

Example

A format description is a sequence of four types of components. These are

Building blocks

- predefined date components,
- predefined time components,
- predefined period references (extended format), and
- ordinary characters.

Predefined components begin with the % sign. Components that begin otherwise are interpreted as ordinary characters. To use a percent sign as an ordinary character, escape it with another percent sign, as in %%.

*Ordinary
characters*

31.7.1 Date-specific components

The date-specific components act as conversion specifiers to denote portions of a date description. They may seem rather cryptic at first, but you will find them useful and constructive when creating customized references to time. They are summarized in Table 31.6.

*Date-specific
components*

| Conversion specifier | Meaning | Possible entries |
|-----------------------------|--------------|------------------|
| %d | day | 01,...,31 |
| %m | month | 01,...,12 |
| %Am <i>set-identifier</i> | month | <i>element</i> |
| %y | year | 00,...,99 |
| %q | quarter | 01,...,04 |
| %Y | weekyear | 00,...,99 |
| %c | century | 00,...,99 |
| %C | weekcentury | 00,...,99 |
| %w | day of week | 1,...,7 |
| %Aw <i>set-identifier</i> | day of week | <i>element</i> |
| %W | week of year | 01,...,53 |
| %j | day of year | 001,...,366 |

Table 31.6: Conversion specifiers for date components

All date conversion specifiers allow only predefined numerical values, except for the specifiers %Am and %Aw. These allow you to specify references to sets. You can use %Am and %Aw to denote months and days by the elements in a specified set. These are typically the names of the months or days in your native language. AIMMS will interpret the elements by their ordinal number. The predefined identifiers AllMonths, AllAbbrMonths, AllWeekdays and AllAbbrWeekdays hold the full and abbreviated English names of both months and days.

*Custom
date-specific
references*

The %Y and %C specifiers refer to the weekyear and weekcentury values of a specific date, as explained on page 481. You can use these if you want to refer to weekly calendar periods by their week number and year.

*Week year and
century*

AIMMS can interpret numerical date-specific references with or without leading zeros when reading your input data. When writing data, AIMMS will insert all leading zeros to ensure a uniform length for date elements. If you do not want leading zeros for a specific component, you can insert the 's' modifier directly after the % sign. For instance, the string "%sd" will direct AIMMS to produce single-digit numbers for the first nine days.

*Omitting
leading zeros*

When using the %Am and %Aw specifiers, AIMMS will generate uniform length elements by adding sufficient trailing blanks to the shorter elements. As with leading zeros, you can use the s modifier to override the generation of these trailing blanks.

*Omitting
trailing blanks*

The format "%Am|AllMonths| %sd, %c%y" will result in the generation of time slots such as 'January 1, 1996'. The date portion of the fixed reference date format used to specify the BEGIN and END DATE attributes of a calendar can be reproduced using the format "%c%y-%m-%d".

Example

31.7.2 Time-specific components

The conversion specifiers for time components are listed in Table 31.7. There are no custom time-specific references in this table, because the predefined numerical values are standard throughout the world.

*Time-specific
components*

| Conversion specifier | Meaning | Possible entries |
|----------------------|----------------------|------------------|
| %h | hour | 01, ..., 12 |
| %H | hour | 00, ..., 23 |
| %M | minute | 00, ..., 59 |
| %S | second | 00, ..., 59 |
| %t | tick | 00, ..., 99 |
| %p | before or after noon | AM, PM |

Table 31.7: Conversion specifiers for time components

AIMMS can interpret numerical time-specific references with or without leading zeros when reading your input data. When writing data, AIMMS will insert leading zeros to ensure a uniform length for time elements. If you do not want leading zeros for a specific component, you can insert the 's' modifier directly after the % sign. For instance, the string "%sh" will direct AIMMS to produce single-digit numbers for the first nine hours.

*Omitting
leading zeros*

The time slot format “%sAw|WeekDays| %sh:%M %p” will result in the generation of time slots such as 'Friday 11:00 PM', 'Friday 12:00 PM' and 'Saturday 1:00 AM'. The full reference date format is given by “%c%-m-%d %H:%M:%S”.

Example

31.7.3 Period-specific components

With period-specific conversion specifiers in either a time slot format or a period format you can indicate that you want AIMMS to display both the begin and end date/time of a time slot or period. You only need to use period-specific references in the following cases.

Use of period references

- The UNIT attribute of your calendar consists of a multiple of one of the basic time units known to AIMMS (e.g. each time slot in your calendar consists of 3 days), and you want to refer to the begin and end day of every time slot.
- You want to provide a description for a period in a timetable consisting of multiple time slots in the associated calendar using the function `PeriodToString` (see also Section 31.8), referring to both the first and last time slot in the period.

By including a period-specific component in a time slot or period format, you indicate to AIMMS that any date, or time, specific component following it refers to either the beginning or the end of a time slot or period. The list of available period-specific conversion specifiers is given in Table 31.8.

Period-specific components

| Conversion specifier | Meaning |
|----------------------|--|
| %B | begin of unit period |
| %b | begin of time slot |
| %I | end of period (inclusive) |
| %i | end of period (inclusive), but omitted when equal to begin of period |
| %E | end of period (exclusive) |
| %e | end of time slot |

Table 31.8: Period-specific conversion specifiers.

Through the “%I” and “%E” specifiers you can indicate whether you want any date/ time components used in the description of the end of a period (or time slot) to be included in that period or excluded from it. Inclusive behavior is common for date references, e.g. the description “Monday - Wednesday” usually means the period consisting of Monday, Tuesday *and* Wednesday. For time references exclusive behavior is used most commonly, i.e. “1:00 - 3:00 PM” usually means the period from 1:00 PM *until* 3:00 PM.

Inclusive or exclusive

After a conversion specifier that refers to the end of a period or time slot (i.e. “%E”, “%I” or “%i”) you should take care when using other date, or time, specific specifiers. AIMMS will only be able to discern time units that are larger than the basic time unit specified in the UNIT attribute of the calendar at hand (or, when you use the function `PeriodToString`, of the calendar associated with the timetable at hand). For instance, when the time slots of a calendar consists of periods of 2 months, AIMMS will be able to distinguish the specific months at the beginning and end of each time slot, but will not know the specific week number, week day or month day at the end of each time slot. Thus, in this case you should avoid the use of the “%W”, the “%w” and the “%d” specifiers after a “%E”, “%I” or “%i” specifier.

Limited resolution

With the “%i” specifier you indicate inclusive behavior, and additionally you indicate that AIMMS must omit the remaining text when the basic time units (w.r.t. the underlying calendar) of begin and end slot of the period to which the specifier is applied, coincide. In practice, the “%i” specifier only makes sense when used in the function `PeriodToString` (see also Section 31.8), as time slots in a calendar always have a fixed length.

The %i specifier

The period description “Monday 12:00-15:00” contains three logical references, namely to a day, to the begin time in hours, and to the end time in hours. The day reference is intended to be shared by the begin and end times.

First example

- The day reference is based on the elements of the (predefined) set `AllWeekdays`. The corresponding conversion specifier is “%Aw|AllWeekdays|”
- The descriptions of the begin and end times both use the conversion specifier “%H:%M”. To denote the begin time of the period you must use the “%B” period reference. For the end time of the period, which is not included in the period, you must use “%E”.

By combining these building blocks with a few ordinary characters you get the complete format string “%Aw|AllWeekdays| %B%H:%M-%E%H:%M”. With this string AIMMS can correctly interpret the element “Monday 12:00-15:00” within a calendar covering no more than one week.

Consider the format “%B%Aw|AllWeekdays|%I - %Aw|AllWeekdays|” within a calendar with day as its basic time unit, and covering at most a week. Using this format string AIMMS will interpret the element “Monday - Wednesday” as the three-day period consisting of Monday, Tuesday, and Wednesday.

Second example

31.7.4 Support for time zones and daylight saving time

When your time zone has daylight saving time, and you are working with time slots or periods on an hourly basis, you may want to include an indicator into the time slot or period format to indicate whether daylight saving time is

Support for daylight saving time

active during a particular time slot or period. Such an indicator enables you, for instance, to distinguish between the duplicate hour when the clock is set back at the end of daylight saving time.

In addition, when your application has users located in different time zones, you may wish to present each user with calendar elements corresponding to their particular time zone. Or, when time-dependent data is stored in a database using UTC time (Universal Time Coordinate, or Greenwich Mean Time), a translation may be required to your own local time representation.

Support for time zones

To support you in scenarios as described above, AIMMS provides

AIMMS support

- a special time zone conversion specifier, which can modify the representation of calendar elements based on specified time zone and daylight saving time, and
- a `TIMESLOT FORMAT` attribute in unit `CONVENTIONS` (see also Section 30.8), which you can use to override the time slot format of every calendar when the `CONVENTION` is active.

With the conversion specifier `%TZ`, described in Table 31.9, you can accomplish the following `CALENDAR`-related tasks:

Time zone specifier

- create the `CALENDAR` elements between the given `BEGIN DATE` and `END DATE` relative to a specified time zone, and
- specify the indicators that must be added to the `CALENDAR` elements when standard or daylight saving time is active.

| Conversion specifier | Meaning |
|--|---|
| <code>%TZ(TimeZone)</code> | translation of calendar element to specified <i>TimeZone</i> , ignoring daylight saving time |
| <code>%TZ(TimeZone) Std Dst </code> | translation of calendar element to specified <i>TimeZone</i> , plus string indicator for standard time (<i>Std</i>) and daylight saving time (<i>Dst</i>) |

Table 31.9: Time zone conversion specifier

The *TimeZone* component of the `%TZ` conversion specifier that you must specify, is a time zone corresponding to the elements in your `CALENDAR`. You must specify the time zone as an explicit and quoted element of the predefined set `AllTimeZones` (explained below), or through a reference to an element parameter into that set. If you do not specify the *Std* and *Dst* indicators, AIMMS will ignore daylight saving time when generating the time slots, regardless whether

Specifying the time zone

daylight saving time is defined for that time zone. If you do not use the %TZ specifier to specify a time zone, AIMMS assumes that you intend to use the local time zone without daylight saving time.

AIMMS provides you access to all time zones defined by your operating system through the predefined set AllTimeZones. The set AllTimeZones contains

*The set
AllTimeZones*

- the fixed element 'Local', representing the local time zone without daylight saving time,
- the fixed element 'LocalDST', representing the local time zone with daylight saving time (if applicable),
- the fixed element 'UTC', representing the Universal Time Coordinate (or Greenwich Mean Time) time zone, and
- all time zones defined by your operating system.

The remaining components of the %TZ specifier are two string indicators *Std* and *Dst*, which are displayed in all generated CALENDAR slots or period strings when standard time (i.e. no daylight saving time) or daylight saving time is active, respectively. Both indicators must be either quoted strings, or references to scalar string parameters. In addition, a run time error will occur when both indicators evaluate to the same string.

*Daylight saving
time indicators*

When you use the %TZ specifier, the date and time components of the generated time slots of a calendar may differ when you specify different time zones, but do not modify the reference dates specified in the BEGIN and END DATE attributes of the calendar. AIMMS always assumes that reference dates are specified in local time without daylight saving time (i.e. in the 'Local' time zone). Hence, all time slots will be shifted by the time differences between the specified time zone and the 'Local' time zone, plus any additional difference caused by daylight saving time.

*Effect on time
slots*

Consider the following four CALENDAR declarations.

Examples

```
CALENDAR:
  identifier   : HourCalendarLocal
  index       : h1
  unit        : hour
  begin date  : "2001-03-25 00"
  end date    : "2001-03-25 06"
  timeslot format : "%c%-m-%d %H:00" ;
CALENDAR:
  identifier   : HourCalendarLocalIgnore
  index       : hi
  unit        : hour
  begin date  : "2001-03-25 00"
  end date    : "2001-03-25 06"
  timeslot format : "%c%-m-%d %H:00%TZ('LocalDST')" ;
CALENDAR:
  identifier   : HourCalendarLocalDST
```

```

index      : hd
unit       : hour
begin date : "2001-03-25 00"
end date   : "2001-03-25 06"
timeslot format : "%c%y-%m-%d %H:00%TZ('LocalDST')|\\"|\\" DST\"|" ;
CALENDAR:
identifier : HourCalendarUTC
index      : hc
unit       : hour
begin date : "2001-03-25 00"
end date   : "2001-03-25 06"
timeslot format : "%c%y-%m-%d %H:00%TZ('UTC')|\\"|\\" DST\"|" ;

```

Assuming that the 'Local' time zone has an offset of +1 hours compared to the 'UTC' time zone, this will result in the generation of the following time slots for each of the calendars

| ! HourCalendarLocal | HourCalendarIgnore | HourCalendarLocalDST | HourCalendarUTC |
|---------------------|--------------------|------------------------|--------------------|
| '2001-03-25 00:00' | '2001-03-25 00:00' | '2001-03-25 00:00' | '2001-03-24 23:00' |
| '2001-03-25 01:00' | '2001-03-25 01:00' | '2001-03-25 01:00' | '2001-03-25 00:00' |
| '2001-03-25 02:00' | '2001-03-25 02:00' | '2001-03-25 03:00 DST' | '2001-03-25 01:00' |
| '2001-03-25 03:00' | '2001-03-25 03:00' | '2001-03-25 04:00 DST' | '2001-03-25 02:00' |
| '2001-03-25 04:00' | '2001-03-25 04:00' | '2001-03-25 05:00 DST' | '2001-03-25 03:00' |
| '2001-03-25 05:00' | '2001-03-25 05:00' | '2001-03-25 06:00 DST' | '2001-03-25 04:00' |
| '2001-03-25 06:00' | '2001-03-25 06:00' | '2001-03-25 07:00 DST' | '2001-03-25 05:00' |

Note that the time slots generated for HourCalendarLocal and HourCalendarIgnore are identical (although 'LocalDST' supports daylight saving time). This is because daylight saving time is ignored when the %TZ specifier has no *Std* and *Dst* indicators. The time slots generated for HourCalendarUTC do not contain the specified daylight saving time indicator, because the 'UTC' time zone has no daylight saving time.

31.8 Converting time slots and periods to strings

The following functions enable conversion between calendar slots and free format strings using the conversion specifiers discussed in the previous section. Their syntax is

Converting time slots to strings

- TimeSlotToString(*format-string*, *calendar*, *time-slot*)
- StringToTimeSlot(*format-string*, *calendar*, *moment-string*).

The result of the function TimeSlotToString is a description of the specified *time-slot* according to *format-string*. The result of StringToTimeSlot is the time slot in *calendar* in which the string *moment-string*, specified according to *format-string*, is contained.

With the function `PeriodToString` you can obtain a description of a period in a timetable that consists of multiple calendar slots.

- `PeriodToString(format-string, timetable, period)`

The result of the function is a description of the time span covered by a *period* in a horizon according to the specified *timetable* and *format-string*. The *format-string* argument can use period-specific conversion specifiers to generate a description referring to both the beginning and end of the period.

Converting timetable periods to strings

The functions `CurrentToString` and `CurrentToTimeSlot` can be used to obtain the current time. Their syntax is

- `CurrentToString(format-string)`
- `CurrentToTimeSlot(calendar)`

The function `CurrentToString` will return the current time according to the specified format string. If you do not use the `%TZ` specifier in the format string of the `CurrentToString` function, AIMMS assumes daylight saving time by default. You can change this default behavior by setting the global option `current_time_in_LocalDST` to 0. The function `CurrentToTimeSlot` returns the time slot in *calendar* containing the current moment.

Obtaining the current time

31.9 Working with elapsed time

Sometimes you may find it easier to formulate your model in terms of (continuous) elapsed time with respect to some reference date rather than in terms of discrete time periods. For example, for a task in a schedule it is often more natural to store just the start and end time rather than to specify all of the time slots in a calendar during which the task will be executed. In addition, working with elapsed time allows you to store time references to any desired accuracy.

Use of elapsed time

For data entry or for the generation of reports, however, elapsed time may not be your preferred format. In this event AIMMS offers a number of functions for the conversion of elapsed time to calendar strings (or set elements) and vice versa, using the conversion specifiers described in section 31.7.

Input-output conversion

The following functions allow conversion between elapsed time and time slots in an existing calendar. Their syntax is

- `MomentToTimeSlot(calendar, reference-date, elapsed-time)`
- `TimeSlotToMoment(calendar, reference-date, time-slot)`

Conversion to calendar elements

The *reference-date* argument must be a time slot in the specified calendar. The *elapsed-time* argument is the elapsed time from the *reference-date* measured

in terms of the calendar's unit. The result of the function `MomentToTimeSlot` is the time slot containing the moment represented by the reference date plus the elapsed time. The result of the function `TimeSlotToMoment` is the elapsed time from the reference date to the value of the *time-slot* argument (measured in the calendar's unit).

The following functions enable conversion between elapsed time and free format strings. Their syntax is

*Conversion to
calendar strings*

- `MomentToString(format-string, unit, reference-date, elapsed-time)`
- `StringToMoment(format-string, unit, reference-date, moment-string)`.

The *reference-date* argument must be provided in the fixed format for reference dates, as described in Section 31.2. The *moment-string* argument must be a period in the format given by *format-string*. The *elapsed-time* argument is the elapsed time in *unit* with respect to the *reference-date* argument. The result of the function `MomentToString` is a description of the corresponding moment according to *format-string*. Strictly spoken, the *unit* argument in `MomentToString` is only required when the option `elapsed_time_is_unitless` (see below) is set to on, and, consequently, *elapsed-time* is unitless. In the case that `elapsed_time_is_unitless` is set to off (the default), you are advised to set the *unit* argument equal to the associated unit of the *elapsed-time* argument. The result of the function `StringToMoment` is the elapsed time in *unit* between *reference-date* and *moment-string*.

```
moment := MomentToString("%c%-Am|A|AbbrMonths|-%d (%sAw|A|Weekdays|) %H:%M",
    [hour], "1996-01-01 14:00", 2.2 [hour] );
! result : "1996-Jan-01 (Monday) 16:12"

elapsed := StringToMoment("%c%-Am|A|AbbrMonths|-%d (%sAw|A|Weekdays|) %H:%M",
    [hour], "1996-01-01 14:00", "1996-Jan-01 (Monday) 16:12" );
! result : 2.2 [hour]
```

Example

The function `CurrentToMoment` can be used to obtain the elapsed time since *reference-date* in the specified *unit* of the current time. Its syntax is

*Obtaining the
current time*

- `CurrentToMoment(unit, reference-date)`.

By default, the result of the functions `TimeSlotToMoment`, `StringToMoment` and `CurrentToMoment` will have an associated unit, namely the unit specified in the *unit* argument. In addition, AIMMS expects the *elapsed-time* argument in the function `MomentToTimeSlot` and `MomentToString` to be of the same unit as its associated *unit* argument. If you want the result or arguments of these functions to be unitless, you can accomplish this by setting the compile time option `elapsed_time_is_unitless` to on. Note, however, that any change to this option affects all calls to these function throughout your model.

Unitless result

31.10 Working in multiple time zones

If your application uses external time-dependent data supplied by users worldwide, you are faced with the problem of converting all dates and times to the calendar in which your application is set up to run. The facilities described in this section help you with that task.

Multiple time zones

With the functions `TimeZoneOffset`, `DaylightSavingStartDate` and `DayLightSavingEndDate` you can obtain various time zone specific characteristics.

Obtaining time zone info

- `TimeZoneOffset(FromTimeZone, ToTimeZone)`
- `DaylightSavingStartDate(year, TimeZone)`
- `DaylightSavingEndDate(year, TimeZone)`

The function `TimeZoneOffset` returns the offset in minutes between the time zones *FromTimeZone* and *ToTimeZone*. You can use the function `DaylightSavingStartDate` and `DaylightSavingEndDate` to retrieve the reference dates, within a particular time zone, corresponding to the begin and end date of daylight saving time.

```
offset := TimeZoneOffset('UTC', 'Eastern Standard Time');
! result: -300 [min]
startdate := DaylightSavingStartDate( '2001', 'Eastern Standard Time');
! result: "2001-04-01 02"
```

Example

With the function `ConvertReferenceDate` you can convert reference dates to different time zones. Its syntax is:

Converting reference dates

- `ConvertReferenceDate(ReferenceDate, FromTimeZone, ToTimeZone[, IgnoreDST])`

With the function you can convert a reference date *ReferenceDate*, within the time zone *FromTimeZone*, to a reference date within the time zone *ToTimeZone*. If the optional argument *IgnoreDST* is set to 1, AIMMS will ignore daylight saving time for both input and output reference dates (see also Section 31.7.4). By default, AIMMS will not ignore daylight saving time.

```
UTCdate := ConvertReferenceDate("2001-05-01 12", 'Local', 'UTC');
! result: "2001-05-01 11"
```

Example

When your application needs to read data from or write data to a database or file with dates not specified in local time, some kind of conversion of all dates appearing in the data source is required during the data transfer. To support you with such date conversions, AIMMS allows you to override the

CONVENTIONS and time zones

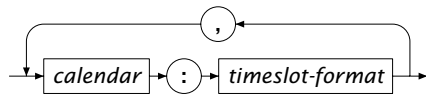
default timeslot format of one or more calendars in your model through a unit CONVENTION (see also Section 30.8).

In the TIMESLOT FORMAT attribute of a CONVENTION you can specify the time slot format (see also Section 31.7) to be used for each CALENDAR while communicating with an external source. The syntax of the TIMESLOT FORMAT attribute is:

*The TIMESLOT
FORMAT attribute*

timeslot-format-list :

Syntax



When an active CONVENTION overrides the time slot format for a particular calendar, all calendar elements will be displayed according to the time slot format specified in the CONVENTION. The time slot format specified in the calendar itself is then ignored.

*Use in graphical
user interface*

If you use a CONVENTION to override the time slot format while writing data to a data file, report file or listing file, AIMMS will convert all calendar elements to the format specified in the TIMESLOT FORMAT for that calendar in the CONVENTION prior to writing the data. Similarly, when reading data from a data file, AIMMS will interpret the calendar slots present in the file according to the specified time slot format, and convert them to the corresponding calendar elements in the model.

*Reading and
writing to file*

When communicating calendar data to a database, there is one additional issue that you have to take into account, namely the data type of the column in which the calendar data is stored in the database table. If calendar data is stored in a string column, AIMMS will transfer data according to the exact date-time format specified in the TIMESLOT FORMAT attribute of the CONVENTION, including any indicators for specifying the time zone and/or daylight saving time.

*Database com-
munication...*

However, if the data type of the column is a special date-time data type, AIMMS will always communicate calendar slots as reference dates, which is the standard date-time string representation used by ODBC. In translating calendar slots to reference dates, AIMMS will adhere to the format specified in the TIMESLOT FORMAT attribute of either the CALENDAR itself, or as overridden in the currently active CONVENTION.

*... for date-
time columns*

The reference dates are generated according to the time zone specified in the %TZ specifier of the currently active TIMESLOT SPECIFIER. These dates always ignore daylight saving time (i.e. shift back by one hour if necessary), as daylight saving time cannot be represented in the fixed reference date format. Specifiers in the TIMESLOT FORMAT other than the %TZ specifier are not used when mapping to date-time values in a database. If you do not specify a %TZ specifier in the TIMESLOT FORMAT, AIMMS will assume that all date-time columns in a database are represented in the 'Local' time zone (the default).

*Generated
reference dates*

Consider the calendar HourCalendarLocalDST defined below.

Example

```
CALENDAR:
  identifier      : HourCalendarLocalDST
  index          : hd
  unit           : hour
  begin date     : "2001-03-25 00"
  end date       : "2001-03-25 06"
  timeslot format : "%c%y-%m-%d %H:00TZ('LocalDST')|\\"|\\" DST\"|";
```

If you want to transfer data defined over this calendar with a database table in which all dates are represented in UTC time, you should define a convention defined as follows.

```
CONVENTION:
  identifier      : UTCConvention
  timeslot format :
    HourCalendarLocalDST : "%c%y-%m-%d %H:00TZ('UTC')";
```

When this convention is active, AIMMS will represent all calendar slots of the calendar HourCalendarLocalDST as reference dates according to the UTC time zone, by shifting as many hours as dictated by the local time zone and/or daylight saving time if applicable. Hence, when you use the convention UTCConvention during data transfer with the database, all calendar data slots will be in the expected format.