
AIMMS Tutorial for Professionals - Functions and Procedures

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

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 9

Functions and Procedures

In the previous chapter you were introduced to database procedures. In this chapter you will develop several AIMMS procedures to read data and to control the entire rolling horizon process. In addition, you will work with an external procedure that is called from within AIMMS.

This chapter

The procedures in this chapter have all been kept small for ease of understanding. The underlying rolling horizon algorithm, however, is not trivial, and results in a multitude of procedures. The chapter is therefore both a tutorial in the use of procedures and a tutorial in the application of a rolling horizon.

Many small procedures

9.1 Reading from a database

Reading all the data at once from a database table is quite easy in AIMMS. Consider, for instance, the database table `LocationTable` declared in the previous chapter. The following statement

*Reading a database table
...*

```
read from table LocationTable;
```

is an instruction to AIMMS to read all identifiers that have been specified in the **Mapping** attribute of the corresponding database table.

It is also possible to read a *selection* of all identifiers specified in the **Mapping** attribute of a database table. For instance, the following statement

... or a portion thereof



```
read XCoordinate, YCoordinate from table LocationTable;
```

only reads data of `XCoordinate` and `YCoordinate`.

At this point, you are asked to create a single procedure named `ReadFromDatabase` to be placed between the `Database Declarations` node and the `NumberOfProductionLinesQuery` node in the model tree in the following manner:

Creating a procedure

- ▶ select the `Database Link` section of the model tree,

- ▶ if open, close this section by clicking on the minus sign  in front of the icon,
- ▶ press the **New Procedure** button  button on the toolbar,
- ▶ enter 'ReadFromDatabase' as the name of the procedure, and
- ▶ press the *Enter* key to register this name.

Open the attribute form of the procedure ReadFromDatabase by double-clicking on its name, and complete the **Body** attribute as shown in Figure 9.1. Note that the two database procedures NumberOfProductionLinesQuery and TotalDemandQuery both result in temporary tables inside the database, and that AIMMS acts as if the name of each procedure is the same as the name of the temporary table.

*Completing the
Body attribute*

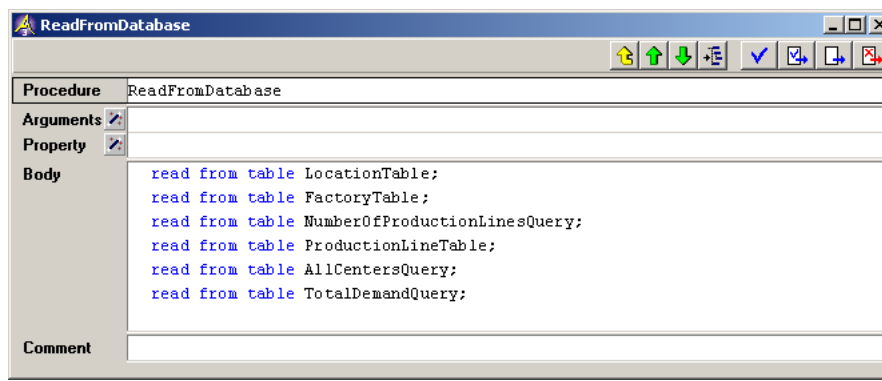



Figure 9.1: The procedure 'ReadFromDatabase'

After you have completed the **Body** attribute of the procedure ReadFromDatabase, close the attribute form using the **Check, commit and close** button . You can now run the procedure by performing the following steps:

*Running the
procedure*

- ▶ select the procedure ReadFromDatabase in the model tree, and
- ▶ select the **Run Procedure** command using the right-mouse pop-up menu (see Figure 9.2).

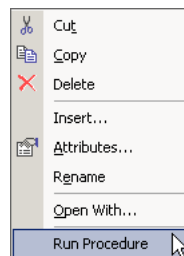



Figure 9.2: The right-mouse menu of the procedure 'ReadFromDatabase'

After you have executed the procedure `ReadFromDatabase` you may want to look at the contents of, for instance, the parameter `MaximumProductionLineLevel`. Before you are able to view its data, you need to locate this parameter in the model tree. You can find it in the following manner:

Finding an identifier ...

- ▶ press the **Find** button  on the toolbar,
- ▶ enter 'MaximumProductionLineLevel' using the name completion facility (see Figure 9.3), and
- ▶ press the **Declaration...** button.

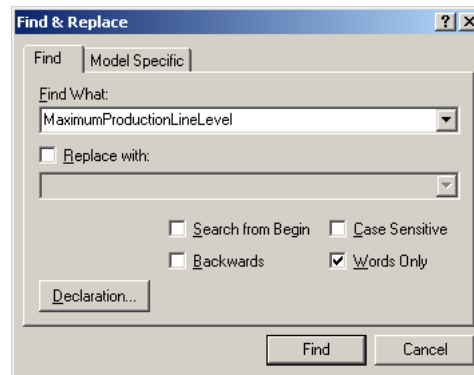


Figure 9.3: The **Find & Replace** dialog box

Next, open the data page for the parameter `MaximumProductionLineLevel` by performing the following two steps:

... and inspecting its data

- ▶ press the right-mouse button to activate the pop-up menu, and
- ▶ select the **Data...** command.

The data page on your computer should now look like the one shown in Figure 9.4.

| | line-01 | line-02 | line-03 | line-04 |
|-----------|---------|---------|---------|---------|
| Eindhoven | 450 | 550 | | |
| Haarlem | 450 | 500 | 550 | 600 |
| Zwolle | 450 | 550 | | |

Figure 9.4: The data page for `MaximumProductionLineLevel`

9.2 External DLL functions

In this section, you will link an external *Dynamic Link Library* (DLL) named ‘External Routines.dll’ to your AIMMS model. Inside this DLL, there is a function named `DLLUnitTransportCost`, that determines the unit transport cost on the basis of the distance between a particular factory and a particular distribution center. Writing your own DLLs is beyond the scope of this tutorial. Chapters 11 and 32 of *The Language Reference*, however, elaborate further on the use of DLLs and the related AIMMS Programming Interface. The source code of ‘External Routines.dll’ has already been copied to the ‘DLL’ subdirectory of your project.

External DLL

The DLL ‘External Routines.dll’ exports the following function.

DLL function ...

```
double DLLUnitTransportCost( char *from_name, char *to_name )
```



The two input arguments of the function are strings representing the names of the two locations for which the unit transport cost is calculated.

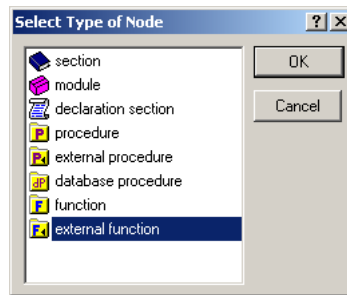
For each external DLL function used in an AIMMS application, you must declare a corresponding *external function* in AIMMS. In this tutorial, the external function is named `ExternalUnitTransportCost`, and has the same number of arguments as its external counterpart.

... and its counterpart in AIMMS

To declare the external function you should perform the following tasks:

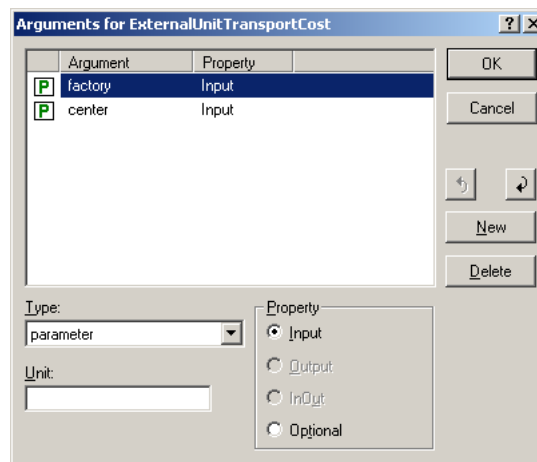
Declaring an external function

- ▶ open the DLL Link model section,
- ▶ press the **Other...**  button on the toolbar,
- ▶ select the external function  from the **Select Type of Node** dialog box (see Figure 9.5),
- ▶ specify ‘ExternalUnitTransportCost(factory,center)’ as the name of the function, and
- ▶ press the *Enter* key to register its name.

Figure 9.5: The **Select Node Type** dialog box

Next, AIMMS will automatically open the **Arguments** wizard as shown in Figure 9.6.

The Arguments wizard

Figure 9.6: The **Arguments** wizard

To complete the **Arguments** wizard, execute the following steps:

- ▶ change the type of the currently selected argument factory to ‘element parameter’,
- ▶ select Factories as its **Range** attribute,
- ▶ then click on the second argument center,
- ▶ change its type to ‘element parameter’,
- ▶ select Centers as its **Range** attribute, and
- ▶ press the **OK** button.

After completing the **Arguments** wizard, AIMMS will have declared the two input arguments as local element parameters. You may verify that AIMMS has indeed placed these local parameters in a new declaration section underneath the ExternalUnitTransportCost node in the model tree (see Figure 9.7).

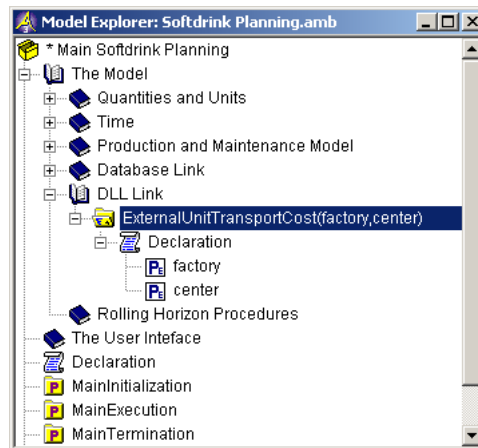


Figure 9.7: The completed DLL section of the model tree

Using wizards it is now straightforward to complete the **Dll name** and **Return type** attributes of the external function as shown in Figure 9.8.

Completing the attributes

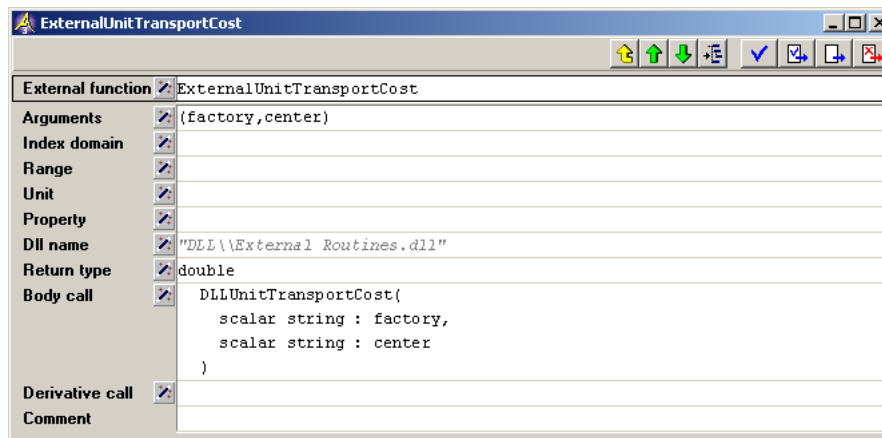




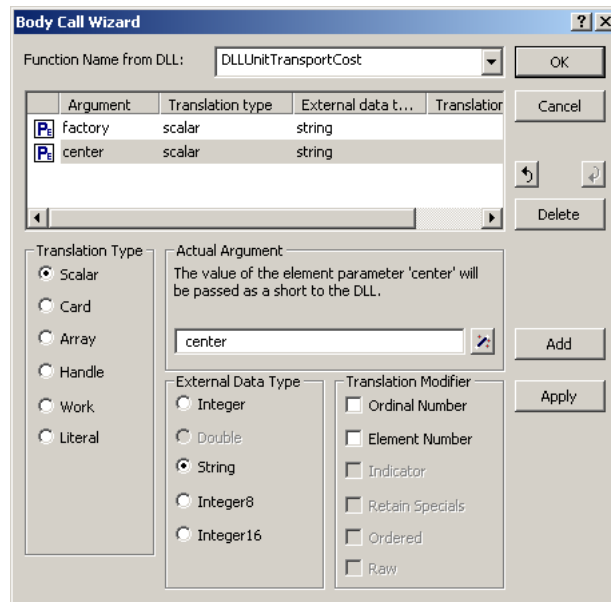
Figure 9.8: The attribute form of the external function ExternalUnitTransport-Cost

The **Body call** attribute specifies the actual link between the arguments of the function in AIMMS and in the DLL. There is an extensive **Body call** wizard, as shown in Figure 9.9, which supports several choices in establishing the link. In the **Body call** wizard (see Figure 9.9) you should perform the following actions:

The Body call attribute

- ▶ select 'Scalar' translation type
- ▶ press the wizard button  to select the element parameter factory as the actual argument,

- ▶ set the external datatype to 'String',
- ▶ press the **Add** button,
- ▶ select 'Scalar' translation type
- ▶ press the wizard button  to select the element parameter center as the actual argument,
- ▶ set the external datatype to 'String',
- ▶ press the **Add** button, and
- ▶ press the **OK** button.

Figure 9.9: The **Body call** wizard

9.3 Specifying the rolling horizon

In this section, you will specify all the procedures that are necessary to describe the rolling horizon process. Once you have implemented the single step contained in this process, it becomes straightforward to describe the overall process. After proper data initialization you are then ready to run the completed set of rolling horizon procedures.

This section

This section is divided into four subsections, as shown in Figure 9.10. You should add these subsections to your own model tree.

Structuring the tree

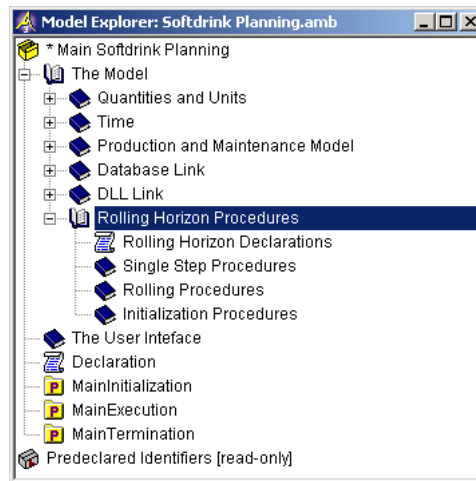


Figure 9.10: The structure of the Rolling Horizon Procedures section

9.3.1 Rolling horizon declarations

There are several identifiers that play a role in the rolling horizon process. Their names are mostly self-explanatory, and their contents are specified below. As you will see in the next subsection, these identifiers are used in the formation of timetables, which link the abstract periods in the rolling horizon model to the specific days and weeks in the two calendars.

*Horizon
identifiers ...*

At this stage, you should enter the following declarations in Rolling Horizon Declarations.

*... and their
declarations*

```

ELEMENT PARAMETER:
  identifier : FirstDayInPlanningInterval
  range     : Days

SET:
  identifier : WeeksInPlanningInterval
  subset of : Weeks
  definition : union[ t, WeekInPeriod(t) ]

ELEMENT PARAMETER:
  identifier : FirstWeekInPlanningInterval
  range     : Weeks
  definition : DayToWeek(FirstDayInPlanningInterval)

ELEMENT PARAMETER:
  identifier : LastWeekInPlanningInterval
  range     : Weeks
  definition : last(WeeksInPlanningInterval)

```

```

PARAMETER:
  identifier : LengthDominatesNotActive
  index domain : t

```

The identifier named `LengthDominatesNotActive` is a required input for the procedure `CreateTimeTable` discussed in the next subsection. Whenever this identifier assumes its default value of zero, then the desired length of any period may not be achieved due to a delimiter slot being encountered in that period. In the example in this tutorial, this parameter is indeed zero. As a result, the timetable `DaysInPeriod` will make sure that each period starts on a Monday (the delimiter slot). Even though the desired length of each period has been set to seven days, its actual length is shortened due to weekends and the official holidays (the so-called inactive days).

Additional explanation

In addition to the five horizon identifiers, you need to enter two registration identifiers. These two identifiers are used to store the overall maintenance and line usage planning. Add the following two parameters at the end of the Rolling Horizon Declarations section:

Registration identifiers

```

PARAMETER:
  identifier : OverallMaintenancePlanning
  index domain : (f,p,w) | p in FactoryProductionLines(f)

PARAMETER:
  identifier : OverallLineUsagePlanning
  index domain : (f,p,w) | p in FactoryProductionLines(f)

```

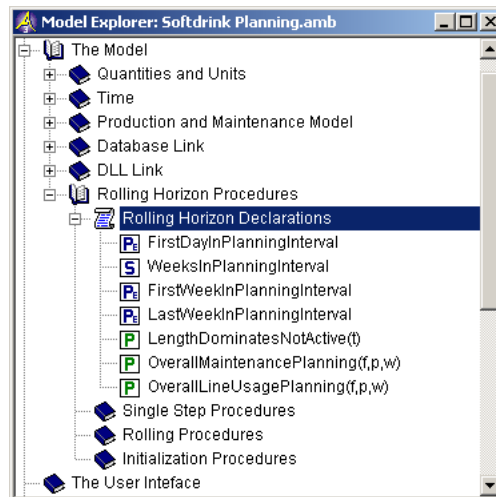


Figure 9.11: The Rolling Horizon Declarations section of the model tree

9.3.2 Single step procedures

A single step in the rolling horizon decision process can be divided into several procedures, as shown in Figure 9.12. The implementation of each procedure will be discussed later on in this subsection. Complete your model tree accordingly, but please follow the instructions in the next paragraph when entering the procedure `RegisterInOverallPlanning` with its two arguments named `iw` and `ip`.

Tree structure

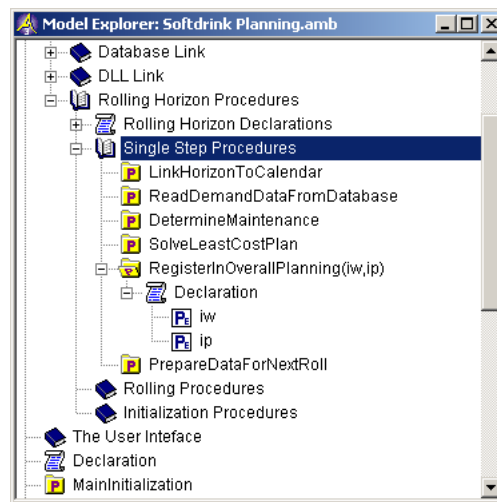


Figure 9.12: The procedures needed to specify a single step

Once you enter the procedure `RegisterInOverallPlanning(iw,ip)` with its two arguments, AIMMS will automatically open a wizard. To complete this **Argument** wizard for both `iw` (referring to a week) and `ip` (referring to a period), you should execute the following actions:

Argument wizard

- ▶ change the type of the currently selected argument `iw` to 'element parameter',
- ▶ select `Weeks` as its **Range** attribute,
- ▶ select 'Input' as its **Property** attribute,
- ▶ then click on the second argument `ip` to change the target,
- ▶ change its type to 'element parameter',
- ▶ select `Periods` as its **Range** attribute, and
- ▶ select 'Input' as its **Property** attribute.

At this point, the **Argument** wizard should be the same as the one shown in Figure Figure 9.13.

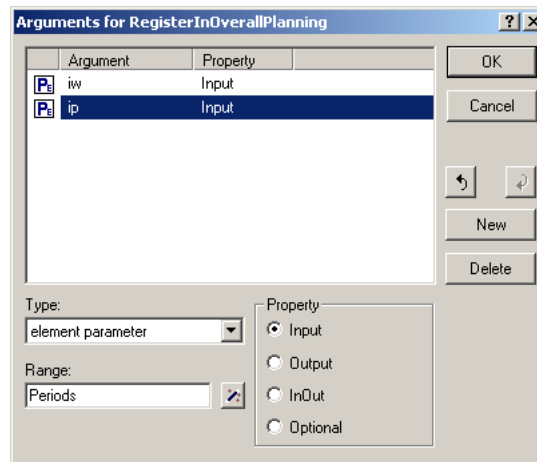


Figure 9.13: Argument wizard

A *timetable* is either an indexed set or an indexed element parameter, representing the mapping between the periods in the horizon and the timeslots in the calendar. It is an indexed set when the period can contain several time slots as for instance in the timetable `DaysInPeriod`. It can be an indexed element parameter when there is a one-to-one mapping between each period and each time slot as for instance in the timetable `WeekInPeriod`.

Describing a timetable

The quick info tip window of the predefined procedure `CreateTimeTable` are shown in Figure 9.14.

Creating a timetable...

```
CreateTimeTable(
  [Output] TimeTable AS Set,
  [Input] CurrentTimeSlot AS Element Parameter,
  [Input] CurrentPeriod AS Element Parameter,
  [Input] PeriodLength AS Parameter,
  [Input] LengthDominates AS Parameter,
  [Input] InactiveTimeSlots AS Set,
  [Input] DelimiterSlots AS Set)
```

Figure 9.14: Quick info tip window of the function `CreateTimeTable`

Through the arguments you have considerable control over the contents of the timetable. For detailed information see Section 29.4 of *The Language Reference* manual.

Go to the **Body** attribute of the procedure `LinkHorizonToCalendar`, and enter the following statements:

... in an AIMMS procedure

```

CreateTimeTable(
  TimeTable      : DaysInPeriod,
  CurrentTimeSlot : FirstDayInPlanningInterval,
  CurrentPeriod  : FirstPeriodInPlanningInterval,
  PeriodLength   : DesiredNumberOfDaysInPeriod,
  LengthDominates : LengthDominatesNotActive,
  InactiveTimeSlots : InactiveDays,
  DelimiterSlots  : Mondays );

ActualNumberOfDaysInPeriod(t) := (card(DaysInPeriod(t))) [day];

CreateTimeTable(
  TimeTable      : WeekInPeriod,
  CurrentTimeSlot : FirstWeekInPlanningInterval,
  CurrentPeriod  : FirstPeriodInPlanningInterval,
  PeriodLength   : DesiredNumberOfWeeksInPeriod,
  LengthDominates : LengthDominatesNotActive,
  InactiveTimeSlots : InactiveWeeks,
  DelimiterSlots  : Weeks );


```

Note that when calling `CreateTimeTable`, the arguments are preceded by their argument names as displayed in Figure 9.14. The use of argument names in function calls is optional in AIMMS. In the above **Body** attribute, the argument names are used to increase the readability.

Argument names are optional

In order to enforce unit consistency in the above assignment statement, the unitless expression `card(DaysInPeriod(t))` is assigned the unit [day]. Such unit casting requires the entire expression to be enclosed between parentheses.

Overriding units

You can use the **Maximize** button  on the toolbar to temporarily enlarge the size of the **Body** attribute (or any other multi-line attribute) to ease entry. When you have completed the attribute, simply press the **Maximize** button again to restore the original size.

Maximizing attribute fields

The timetable `DaysInPeriod` contains the working days in a week, and explicitly excludes the inactive days such as the weekends and the official holidays. The sole reason why this timetable is created, is to determine the parameter `ActualNumberOfDaysInPeriod` needed to establish the correct level of production.

DaysInPeriod ...

To view the contents of the `DaysInPeriod` timetable, you should first initialize the element parameter `FirstDayInPlanningInterval`. All other input arguments have already been initialized. Execute the following steps:

... requires one more initialization

- ▶ select the procedure `LinkHorizonToCalendar` in the model tree,
- ▶ press the *Enter* key to open its attribute form,

- ▶ position the text cursor somewhere within the string 'FirstDayInPlanningInterval' in the **Body** attribute,
- ▶ press the right-mouse button to activate the pop-up menu,
- ▶ select the **Data...** command,
- ▶ click on the empty right-hand side of the equal sign in the *Data* page,
- ▶ specify '03/07/2000' (without the quotes) as the value on the data page, and
- ▶ press the **Close** button.

You may re-open the page to verify that AIMMS has accepted your input value. If the input format you entered was incorrect, AIMMS will replace your input with the default empty string.

At this point, you can view the contents of the timetable DaysInPeriod by running the procedure and looking at the appropriate data page:

... is first determined


- ▶ position the text cursor somewhere within the string 'LinkHorizonToCalendar' in the **Procedure** attribute,
- ▶ press the right-mouse button to activate the pop-up menu, and
- ▶ select the **Run Procedure** command.

You can ignore all the initialization warnings since the existing default values suffice at this point in the tutorial. Please close the **Errors/Warnings** window and continue.

Next construct the data page corresponding to the timetable DaysInPeriod as shown in Figure 9.15 by executing the following steps:

... and can then be viewed

- ▶ position the text cursor somewhere within the string 'DaysInPeriod' in the **Body** attribute,
- ▶ press the right-mouse button to activate the pop-up menu again, and
- ▶ select the **Data...** command.

Note that each period covers exactly five days due to the fact that the weekends are excluded. The default format of this data page requires you to scroll horizontally. You may select a different view by pressing the **Change view** button , and choosing, for instance, 'Sparse List' as the Type of Object.

| Days | 01/07/2000 | 02/07/2000 | 03/07/2000 | 04/07/2000 | 05/07/2000 | 06/07/2000 | 07/07/2000 | 08/07/2000 | 09/07/2000 | 10/07/2000 | 11/07/2000 |
|-----------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| past | | | | | | | | | | | |
| period-01 | | | | | | | | | | | |
| period-02 | | | | | | | | | | | |
| period-03 | | | | | | | | | | | |
| period-04 | | | | | | | | | | | |
| period-05 | | | | | | | | | | | |
| period-06 | | | | | | | | | | | |
| period-07 | | | | | | | | | | | |
| period-08 | | | | | | | | | | | |
| period-09 | | | | | | | | | | | |
| period-10 | | | | | | | | | | | |

Figure 9.15: The data page of the day-based timetable

The weekly calendar in this tutorial spans a period of roughly one year. The planning horizon in a single step of the overall rolling horizon procedure, however, is just a small subset of weeks. That is why the procedure `ReadDemandDataFromDatabase` is introduced to limit the total amount of demand data that is loaded into memory at any given time.

Reading just a subset of demand data ...

Prior to each subsequent step of the rolling horizon process, it is recommended that you first empty the weekly demand data associated with the old planning interval, and then read the demand data for the weeks in the new planning interval. This can be accomplished by entering the following statements in the **Body** attribute of the procedure `ReadDemandDataFromDatabase`.

... goes as follows

```
empty WeeklyDemand;
read WeeklyDemand(c,w,s) from table CenterTable
    filtering w in WeeksInPlanningInterval;
Demand(c,t,s) := WeeklyDemand(c,WeekInPeriod(t),s);
```

Note that the weekly demand is read for only those weeks that are in the current planning interval. Using the timetable `WeekInPeriod`, the weekly demand is then assigned to period demand as required by the mathematical program to be solved.

The parameter `DeteriorationLevel` registers, for each combination of factory and production line, the amount of time that has elapsed since that line was maintained. Assuming that all lines will be in use for the entire planning interval, it is a straightforward calculation to estimate when a production line should be under maintenance.

Determine when under maintenance

Now comes the slightly tricky requirement: in each factory no more than one production line can be maintained in the first period. If there is more than one candidate, you should maintain just one line, and delay the maintenance of the other candidate(s) to the next period. The final result is then stored in the parameter `LineInMaintenance` declared for each factory, production line

At most one line under maintenance in first period

and period. This parameter is one of the determinants of the production level of a line when in use (see the definition of the parameter PotentialProduction).

Before specifying the **Body** attribute of the procedure DetermineMaintenance, you need to declare the following two local identifiers in a new declaration section within the procedure node DetermineMaintenance.

Entering local declarations

```

ELEMENT PARAMETER:
  identifier : EstimatedMaintenancePeriod
  index domain : (f,p)
  range : Periods

SET:
  identifier : LinesInMaintenanceInFirstPeriod
  index domain : f
  subset of : ProductionLines

```

Figure 9.16 shows the local declaration section of the procedure DetermineMaintenance.

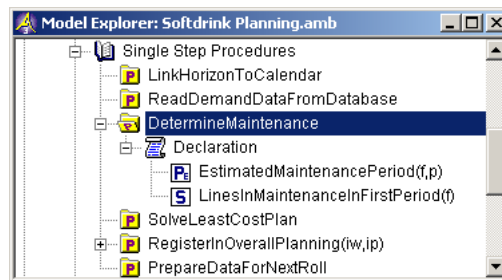


Figure 9.16: The local declaration of the procedure DetermineMaintenance

The following statements in AIMMS have been discussed in the previous paragraph. Please enter them in the **Body** attribute of the procedure DetermineMaintenance.

Entering maintenance calculations

```

EstimatedMaintenancePeriod(f,p) :=
  Element( Periods, max( MaximumDeteriorationLevel(f,p) -
    Floor(DeteriorationLevel(f,p)) + 2, 2 ) );

LinesInMaintenanceInFirstPeriod(f) :=
  { p | EstimatedMaintenancePeriod(f,p) = FirstPeriodInPlanningInterval };

EstimatedMaintenancePeriod( (f,p) |
  Ord(p,LinesInMaintenanceInFirstPeriod(f)) >= 2 ) += 1;

empty LineInMaintenance;
LineInMaintenance((f,p,EstimatedMaintenancePeriod(f,p)) |
  EstimatedMaintenancePeriod(f,p) in Periods.Planning ) := 1;

```

Having completed the first three single step procedures, you are now ready to enter the procedure in which the single step mathematical program is solved. Please enter the following statements in the **Body** attribute of the procedure `SolveLeastCostPlan`.

*Entering a
'solve'
procedure ...*

```
solve LeastCostPlan;
halt with "Least cost mathematical program is not optimal.\nCheck "
      + "input data for infeasibilities."
      when ( LeastCostPlan.ProgramStatus <> 'Optimal' );
```

Note that the second statement illustrates the use of the `halt` statement in AIMMS. Once the program halts, it will provide a two-line message as indicated by the special character `'\n'`. By using the `+` notation in the **Body** attribute, you may divide a single quoted string into several pieces. In the conditional `when` part of the halt statement, there is a reference to a property of the mathematical program, namely the *program status*, using the `'dot'` notation (see Section 15.2 in *The Language Reference*).

*... with a halt
statement*

Following the solution of the single step mathematical program, the results associated with just the first period are kept as `'definite'`. In this tutorial, only the overall planning of maintenance and the overall planning of production line usage are kept. The overall planning is registered in terms of calendar weeks, which implies that period data must be translated into week data. Such translation is achieved with the following two statements, to be added to the **Body** attribute of the procedure `RegisterInOverallPlanning`:

*Register overall
planning*

```
OverallMaintenancePlanning(f,p,iw) := LineInMaintenance(f,p,ip);
OverallLineUsagePlanning(f,p,iw) := ProductionLineInUse(f,p,ip);
```

Once the overall planning has been registered, all that remains is to prepare several data items for the next step. First of all, the first day in the planning interval must be moved forward seven days to the next Monday. Then the current first-period stock and production solution data must become historic data. Finally, the deterioration level of all the production lines must be properly adjusted upwards or downwards. All these assignments are captured in the following **Body** attribute of the procedure `PrepareDataForNextRoll`.

*Preparing data
for next step*

```
FirstDayInPlanningInterval += 7;

Stock(l,'past',s) :=
  Stock(l,FirstPeriodInPlanningInterval,s);
ProductionLineInUse(f,p,'past') :=
  ProductionLineInUse(f,p,FirstPeriodInPlanningInterval);

DeteriorationLevel(f,p) +=
  0.75 * ProductionLineInUse(f,p,FirstPeriodInPlanningInterval) + 0.25;
DeteriorationLevel( (f,p) |
  LineInMaintenance(f,p,FirstPeriodInPlanningInterval) ) := 0;
```

Note that the deterioration level of a productive line is updated by 1 reflecting that the line was in use during the first period in the planning interval. Otherwise, the deterioration level is increased by only 0.25 to reflect that the line remained idle for that week. Of course, if a line is under maintenance during the first period, its deterioration level is reset to zero.

Updating deterioration level

9.3.3 Rolling Procedures

Two rolling horizon procedures can be considered. One of them captures all the procedures needed to execute a single step in the rolling horizon process. You may execute this procedure sequentially by using the corresponding right-mouse action, and examine the results as they are found. The second procedure executes all the remaining single steps in one go. Please update the section Rolling Procedures in your tree structure as shown in Figure 9.17.

Tree structure

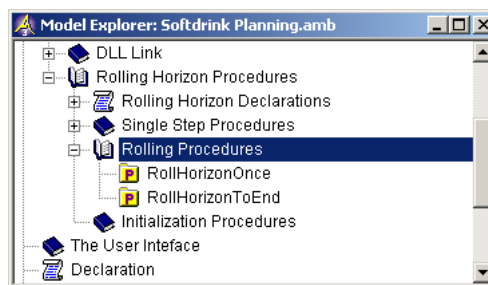


Figure 9.17: The structure of the Rolling Procedures section

The following sequence of statements carries out a single step in the rolling horizon process. Please enter them in the **Body** attribute of the procedure `RollHorizonOnce`. Note that each of the statements is a call to a procedure that was developed in the previous subsection.

Roll horizon once

```
LinkHorizonToCalendar;
ReadDemandDatafromDatabase;
DetermineMaintenance;
SolveLeastCostPlan;
RegisterInOverallPlanning(FirstWeekInPlanningInterval,FirstPeriodInPlanningInterval);
PrepareDataForNextRoll;
```

The following procedure completes the rolling horizon process starting from the current point in the calendar as determined by the element parameter `FirstWeekInPlanningInterval`. In the next subsection, you will encounter a procedure that will allow you to start the rolling horizon process from the beginning of the calendar. Please enter the following statements in the **Body** attribute of the procedure `RollHorizonToEnd`.

Rolling horizon to end

```

while ( LastWeekInPlanningInterval < LastWeekInCalendar ) do
  RollHorizonOnce;
endwhile;

for ( t | t > FirstPeriodInPlanningInterval ) do
  RegisterInOverallPlanning(WeekInPeriod(t),t);
endfor;

```

Note that the maintenance and line usage planning of the final planning interval is not only registered for the first period through the procedure `RollHorizonOnce`, but also for the remaining periods through the execution of the `for` statement.

*Complete
overall planning*

9.3.4 Initialization procedures

There are three initialization procedures to be considered. One of them is the system-supplied procedure `MainInitialization` that is executed every time a project is started. The other two initialization procedures have been embedded in `MainInitialization`, but can also be called separately. Please update your tree structure as shown in Figure 9.18. Be sure not to create a `MainInitialization` procedure, because one is already present in your model. Simply move it from the end of the model tree to its desired position (using either the cut-and-paste or the drag-and-drop facility in AIMMS).

Tree structure

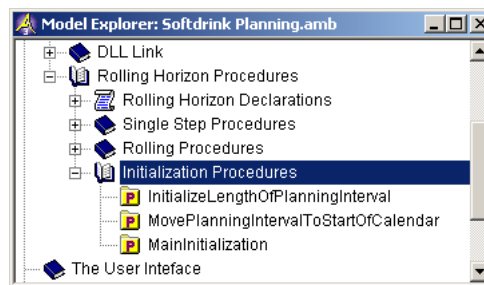


Figure 9.18: The structure of the Initialization Procedures section

In the procedure `InitializeLengthOfPlanningInterval`, two crucial parameters in the rolling horizon are set. Their values determine the amount of time considered in a single step of the rolling horizon process. You may change these values if you want to consider different planning intervals. Please enter the following statements into the **Body** attribute.

*Initializing
Periods*

```

NumberOfPeriods           := 10;
NumberOfPeriodsInPlanningInterval := 8;

```

The procedure `MovePlanningIntervalToStartOfCalendar` first empties any existing overall maintenance and line usage planning, and then assigns all starting values known at the beginning of the calendar to the appropriate variables and parameters. This procedure can be called at any time, causing any activated rolling horizon procedures to start at the beginning of the calendar. Please enter the following statements into the **Body** attribute.

Starting at the beginning

```
empty OverallMaintenancePlanning, OverallLineUsagePlanning;

Stock(l,'past',s)           := StockAtStartOfCalendar(l);
ProductionLineInUse(f,p,'past') := 1 onlyif ProductionLineLevelAtStartOfCalendar(f,p);

DeteriorationLevel(f,p)     := DeteriorationLevelAtStartOfCalendar(f,p);
FirstDayInPlanningInterval := first( Mondays );
WeekInPeriod(t)            := Element( Weeks, Ord(t) );
```

The procedure `MainInitialization`, executed by `AIMMS` at the start of each run, is a natural starting point for reading data, initializing various parameters and starting other procedures that also initialize your model data. In this tutorial, the procedure `MainInitialization` reads essentially all the problem data from the database tables. The only exception is the demand data, which are read one section at a time for the current planning horizon from within the procedure `RollHorizonOnce`. Following this, the unit transport costs are obtained by calling the external function developed in Section 9.2. Finally, the data initialization required for the rolling horizon procedures is completed by calling the two procedures described above. Please replace the content of the `MainInitialization` procedure by the following statements.

MainInitialization

```
ReadFromDatabase;
UnitTransportCost(f,c) := (ExternalUnitTransportCost(f,c)) [$/TL];
InitializeLengthOfPlanningInterval;
MovePlanningIntervalToStartOfCalendar;
empty LengthDominatesNotActive, InactiveWeeks;
```

Note that the unit `[$/TL]` is attributed to the output of the external function. This requires you to place the parentheses around the function call as illustrated above.

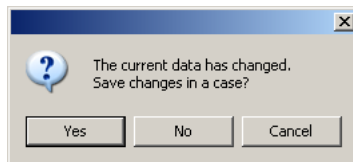
FormatString and unit casting

9.4 Running the model

As indicated previously, the statements that you entered in the `MainInitialization` procedure are executed when the project is opened. Even though you could run this procedure directly using the right-mouse **Run Procedure** command, you may as well try out the default action by first closing the project and then re-opening it. To do so, execute the following steps to close your project:

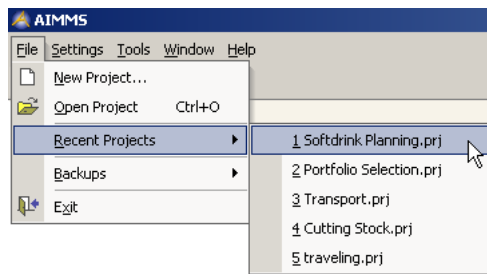
Closing your project

- ▶ select the **Close Project** command from the AIMMS **File** menu,
- ▶ answer 'No' when being asked to compile your model before closing the project,
- ▶ answer 'No' in the dialog box that asks whether you want to save your data (see Figure 9.19),
- ▶ answer 'Yes' to save the changed project.

Figure 9.19: The **Save Changes** dialog box

Opening a project that you have just closed, is straightforward. AIMMS keeps track of the last five projects opened. Just select the 'Softdrink Planning' project from project list displayed in the AIMMS **Start Page**. Alternatively, you can select the recent project from the **File** menu (see Figure 9.20).

Opening the project

Figure 9.20: The **File** menu of AIMMS

You are now ready to test the rolling horizon process starting from the beginning of the calendar. To run the procedure `RollHorizonOnce` you should perform the following actions:

Running a procedure

- ▶ select the procedure `RollHorizonOnce` in the model tree, and
- ▶ in the right-mouse menu select the **Run Procedure** command (see Figure 9.21).

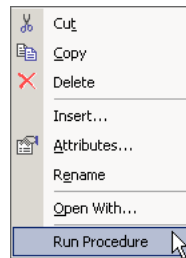
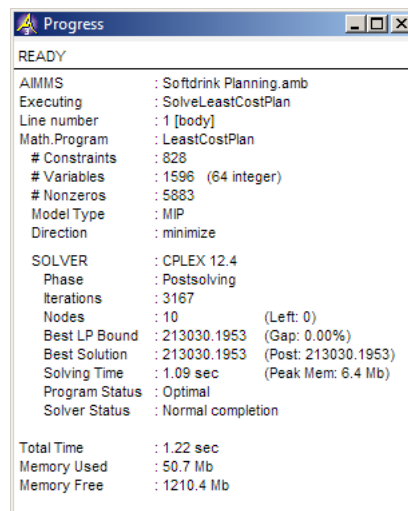


Figure 9.21: The right-mouse menu of the procedure RollHorizonOnce

The **Progress** window lets you to monitor the progress made by AIMMS and the solver during the generation and solution of a mathematical program. By pressing the *Ctrl-P* key combination, the **Progress** window as shown in Figure 9.22 will appear. Once the solution has been found, AIMMS will again display warnings about data not yet initialized. These warnings can be ignored at this stage of the tutorial.

Monitoring the progress

Figure 9.22: The **Progress** window

Once the procedure RollHorizonOnce has finished, you can view the results. For instance, you could open the data page associated with the variable TotalCost, and compare its value to the one in the **Progress** window in Figure 9.22. Similarly, you can inspect the value of any of the decision variables. For example, the optimal values for the variable Production are displayed in Figure 9.23

Viewing the solution

| | period-01 | period-02 | period-03 | period-04 | period-05 | period-06 | period-07 | period-08 |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Eindhoven | 3000 | 3000 | 3000 | 3000 | 3000 | 3000 | 3000 | 3000 |
| Haarlem | 7750 | 7750 | 7750 | 4650 | 3150 | 4650 | 4650 | 4650 |
| Zwolle | 2750 | 2750 | 1650 | 1650 | 1650 | 1650 | 1650 | 1650 |

Figure 9.23: The data page of the variable Production

By default AIMMS will display non-scalar data in a pivot table. For variables and constraints, additional information (e.g. marginal values, basic status) will also be shown in the pivot table when available. Notice that in the data page of the variable Production the basic status is displayed.

Additional information in a pivot table

At this point in the tutorial, you have reached a major milestone in that the complete model description of a rolling horizon application has been completed. In the next part of this tutorial, you will concentrate on building a graphical user interface for the end-user of this application.

Ready for GUI building