AIMMS Language Reference - Execution of Nonprocedural Components

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.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

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

Chapter 7

Execution of Nonprocedural Components

This chapter

The collection of all set and parameter definitions form a system of functional relationships which AIMMS keeps up-to-date automatically. This chapter discusses the dependency structure of the system, the kind of expressions and statements allowed inside the definitions, and the way in which the relationships are re-computed.

Spreadsheets

The nonprocedural execution mechanism discussed in this chapter resembles the execution of spreadsheets. Definitions can be placed in any order by the model builder, but the logical order of execution is determined by the system. As a result, you can easily formulate spreadsheet-based applications in the AIMMS modeling language by merely using definitions for sets and parameters. Of course, the modeling language in AIMMS goes beyond the modeling paradigm of spreadsheets, as AIMMS also offers procedural execution which is found in programming languages but not in spreadsheets.

7.1 Dependency structure of definitions

Dependency graph

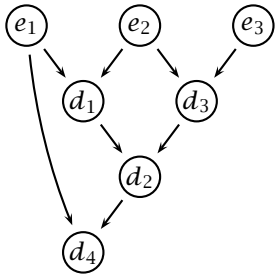
The definitions inside the declarations of global sets and parameters together form a system of interrelated functional relationships. AIMMS automatically determines the dependency between the defined identifiers and the inputs that are used inside these relationships. Such dependencies can be depicted in the form of a directed graph, called the *dependency graph*. From this dependency graph, AIMMS determines the minimal set of identifiers that must be recomputed—and in which order—to get the total system of functional relationships up-to-date.

Example

Consider the system of definitions

$$
\begin{aligned}
d_1 &\equiv e_1 + e_2 \\
d_2 &\equiv d_1 + d_3 \\
d_3 &\equiv e_2 + e_3 \\
d_4 &\equiv e_1 + d_2.
\end{aligned}
$$

Its dependency graph, with identifiers as nodes and dependencies as directed arcs, looks as follows. Note that a change to the input parameter e_3, for in-



stance, requires the re-computation of the defined parameters $d_2, \ldots, d_4$—but not of d_1—to update the entire system.

Dependencies must be a-cyclic

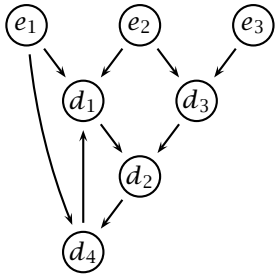
The dependency graph associated with the set and parameter definitions must be a-cyclic, i.e. must not contain circular references. In this case, every change to one or more input parameters of defined sets or parameters will result in a *finite* sequence of assignments to update the system. If the dependency graph is cyclic, a simultaneous system of relations will result. Such a system may not have a (unique) solution, and can only be solved by a specialized solver. Simultaneous systems of relations are handled inside AIMMS through the use of constraints and mathematical programs.

Example

An illegal set of dependencies results if the definition of d_1 in the last example is changed as follows.

$$d_1 \equiv d_4 + e_1 + e_2.$$

This results in the following cyclic dependency graph. Now, a change to any



of the input parameters $e_1, \ldots, e_3$ will result in a simultaneous system for the parameters d_1, d_2 and d_4.

AIMMS will check

AIMMS computes the dependency structure between the parameter and set definitions while compiling your model. If AIMMS detects a cyclic dependency, an error will result, because AIMMS can, in general, not deal with cyclic dependencies without relying on specialized numerical solvers. In that case you need to remove the cyclic dependencies before you can execute the model without further modifications. If you are unable to remove the cyclic dependencies, you have essentially two alternatives. You can either formulate a mathematical program, or define your own solution method inside a procedure.

Variables for simultaneous systems

The cyclic system can be turned into a mathematical program by changing the parameters with cyclic definitions into variables. This results in a simultaneous system of equalities which can be solved through a SOLVE statement. The declaration of mathematical programs is discussed in Chapter 15.

Feedback loops

The alternative is to implement a customized solution procedure by breaking the simultaneous system into a simulation with a feedback loop linking inputs and outputs. To accomplish this, you must first remove the cyclic definitions from the declarations, and then add a procedure that implements the feedback loop. If you have sufficient knowledge of the process you are describing, this route may result in fast convergence behavior.

Dependency is global only

AIMMS only allows a definition for globally declared sets and parameters. Consequently, a single global dependency graph suffices to express the functional relationships between all defined sets and parameters.

Dependency is symbolic

In addition, the dependency structure between set and parameter definitions is purely based on symbol references. As a result, AIMMS' automatic evaluation scheme will always recompute an indexed (output) parameter depending on an indexed (input) parameter *in its entirety*, even when only a single input value has changed.

Inefficiency may occur

This evaluation behavior may lead to severe inefficiencies when you use a high-dimensional defined parameter that is re-evaluated repeatedly during the execution of a loop in your model. In such cases it is advisable to refrain from using a definition for such a parameter, but replace it by one or more assignments at the appropriate places in your model. This issue is discussed in full detail in Section 13.2.3.

7.2 Expressions and statements allowed in definitions

Complicated definitions

In most applications, the functional relationship between input and output identifiers in the definition of a set or a parameter can be expressed as an ordinary set-valued, set element-valued or numerical expression. In rare occa-

sions where a functional relationship cannot be written as a single symbolic statement, a function or procedure can be used instead.

Allowed definitions

In summary, you may use one of the following items in set and parameter definitions:

- a set-valued expression,
- an element-valued expression,
- a numerical expression,
- a call to a function, or
- a call to a procedure.

Limited self-referencing allowed

Under some conditions, expressions used in the definition of a particular parameter can contain references to the parameter itself. Such self-referencing is allowed if the *serial* computation of the definition over all elements in the index domain of the parameter does not result in a cyclic reference to the parameter at the individual level. This is useful, for instance, when expressing stock balances in a functional manner with the use of lag operators.

Example

The following definition illustrates a valid example of a self-reference.

```
Parameter Stock {
    IndexDomain  : t;
    Definition   : {
        if ( t = FirstPeriod ) then BeginStock
            else Stock(t-1) + Supply(t) - Demand(t) endif
    }
}
```

If `t` is an index into a set `Periods = {0..3}`, and `FirstPeriod` equals `0`, then at the individual level the assignments with self-references are:

```
    Stock(0) := BeginStock ;
    Stock(1) := Stock(0) + Supply(1) - Demand(1) ;
    Stock(2) := Stock(1) + Supply(2) - Demand(2) ;
    Stock(3) := Stock(2) + Supply(3) - Demand(3) ;
```

Since there is no cyclic reference, the above definition is allowed.

Functions and procedures

You can use a call to either a function or a procedure to compute those definitions that cannot be expressed as a single statement. If you use a procedure, then only a single output argument is allowed. In addition, the procedure cannot have any side-effects on other global sets or parameters. This means that no direct assignments to other global sets or parameters are allowed.

Arguments and global references

The identifiers referenced in the actual arguments of a procedure call, as well as the global identifiers that are referenced in the body of the procedure, will be considered as input parameters for the computation of the current definition. That is, data changes to any of these input identifiers will trigger the re-execution of the procedure to make the definition up-to-date. The same applies to functions used inside definitions.

Examples

The following two examples illustrate the use of functions and procedures in definitions.

- Consider a function TotalCostFunction which has a single argument for individual cost coefficients. Then the following declaration illustrates a definition with a function reference.

```
Parameter TotalCost {
    Definition : TotalCostFunction( CostCoefficient );
}
```

  AIMMS will consider the actual argument CostCoefficient, as well any other global identifier referenced in the body of TotalCostFunction as input parameters of the definition of TotalCost.
- Similarly, consider a procedure TotalCostProcedure which performs the same computation as the function above, but returns the result via a (single) output argument. Then the following declaration illustrates an equivalent definition with a procedure reference.

```
Parameter TotalCost {
    Definition : TotalCostProcedure( CostCoefficient, TotalCost );
}
```

One procedure for several definitions

Whenever the values of a number of identifiers are computed simultaneously inside a single procedure without arguments, then this procedure must be referenced inside the definition of each and all of the corresponding identifiers. If you do not reference the procedure for all corresponding identifiers, a compile-time error will result. All other global identifiers used inside the body of the procedure count as input identifiers.

Example

Consider a procedure ComputeCosts which computes the value of the global parameters FixedCost(m,p) and VariableCost(m,p) simultaneously. Then the following example illustrates a valid use of ComputeCosts inside a definition.

```
Parameter FixedCost {
    IndexDomain  : (m,p);
    Definition   : ComputeCosts;
}
Parameter VariableCost {
    IndexDomain  : (m,p);
    Definition   : ComputeCosts;
}
```

Omitting ComputeCosts in either definition will result in a compile-time error.

7.3 Nonprocedural execution

Execution based on definitions

Execution based on definitions is typically not controlled by the user. It takes place automatically, but only when up-to-date values of defined sets or parameters are needed. Basically, execution can be triggered automatically from within:

- the body of a function or procedure, or
- an object in the graphical user interface.

Relating definitions and procedures

Consider a set or a parameter with a definition which is referenced in an execution statement inside a function or a procedure. Whenever the value of such a set or parameter is not up-to-date due to previous data changes, AIMMS will compute its current value just prior to executing the corresponding statement. This mechanism ensures that, during execution of functions or procedures, the functional relationships expressed in the definitions are always valid.

Lazy evaluation

During execution AIMMS minimizes its efforts and updates only those values of defined identifiers that are needed at the current point of execution. Such *lazy evaluation* can avoid unnecessary computations and reduces computational time significantly when the number of dependencies is large, and when relatively few dependencies need to be resolved at any particular point in time.

GUI requests

For the graphical objects in an end-user interface you may specify whether the data in that object must be up-to-date at all times, or just when the page containing the object is opened. AIMMS will react accordingly, and automatically update all corresponding identifiers as specified.

The set CurrentAutoUpdatedDefinitions

Which definitions are automatically updated in the graphical user interface whenever they are out-of-date, is determined by the contents of the predefined set CurrentAutoUpdatedDefinitions. This set is a subset of the predefined set AllIdentifiers, and is initialized by AIMMS to the union of the sets AllDefinedSets and AllDefinedParameters by default.

Exclude from auto-updating

To prevent auto-updating of particular identifiers in your model, you should remove such identifiers from the set CurrentAutoUpdatedDefinitions. You can change its contents either from within the language or from within the graphical user interface. Typically, you should exclude those identifiers from auto-updating whose computation takes a long time to finish. Instead of waiting for their computation on every input change, it makes much more sense to collect all input changes for such identifiers and request their re-computation on demand.

Requesting updates

All identifiers that are not contained in CurrentAutoUpdatedDefinitions must be updated manually under your control. AIMMS provides several mechanisms:

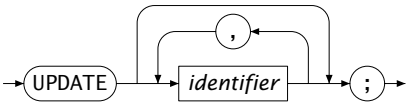
- you can call the UPDATE statement from within the language, or
- you can attach update requests of particular identifiers as actions to buttons and pages in the end-user interface.

The UPDATE statement

The UPDATE statement can be used to update the contents of one or more identifiers during the execution of a procedure that is called by the user. In this way, selected identifiers which are shown in the graphical user interface and not kept up-to-date automatically, can be made up-to-date once the procedure is activated by the user.

Syntax

update-statement :



Allowed identifiers

The following selections of identifiers are allowed in the UPDATE statement:

- identifiers with a definition,
- identifiers associated with a structural section in the model-tree, and
- identifiers in a subset of the predefined set AllIdentifiers.

Example

The following execution statement inside a procedure will trigger AIMMS to update the values of the identifiers FixedCost, VariableCost and TotalCost upon execution.

```
Update FixedCost, VariableCost, TotalCost;
```