
AIMMS User's Guide - Procedures and Functions

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. \TeX , \LaTeX , and $\AMS\text{-}\LaTeX$ 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 \LaTeX and the LUCIDA font family.

Chapter 6

Procedures and Functions

This chapter describes how you can add procedures and functions to a model. It also shows how you can add arguments and local identifiers to procedures and functions. In addition, it illustrates how the body of a procedure or function can be broken down into smaller pieces of execution code, allowing you to implement procedures and functions in a top-down approach.

This chapter

6.1 Creating procedures and functions

Procedures and functions are the main means of executing the sequential tasks in your model that cannot be expressed by simple functional relationships in the form of identifier definitions. Such tasks include importing or exporting your model's data from or to an external data source, the execution of data assignments, and giving AIMMS instructions to optimize a system of simultaneous equations.

Procedures and functions

Procedures and functions are added as a special type of node to the model tree, and must be placed in the main model, or in any book section. They cannot be part of declaration sections, which are exclusively used for model identifiers. Procedures and functions are represented by folder icons, which open up when the node is expanded. Figure 6.1 illustrates an example of a procedure node in the model tree.

Creating procedures and functions

After you have inserted a procedure or function node into the tree, you have to enter its name. If you want to add a procedure or function with arguments, you can add the argument list here as well. Alternatively, you can specify the argument list in the attribute window of the procedure or function. The full details for adding arguments and their declaration as identifiers, local to the procedure or function, are discussed in Section 6.2. Whether or not the arguments are fully visible in the tree is configurable using the **View** menu.

Naming and arguments

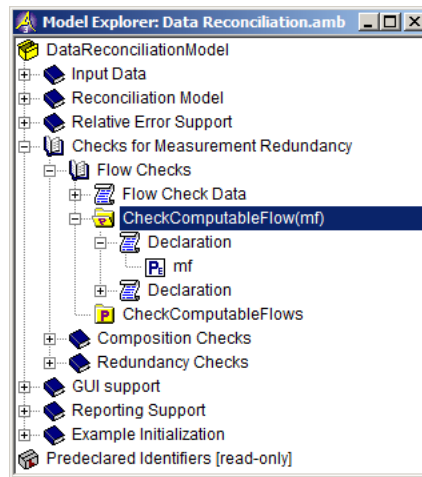


Figure 6.1: Example of a procedure node

The attribute window of a procedure or function lets you specify or view aspects such as its list of arguments, the index domain of its result, or the actual body. The body may merely consist of a SOLVE statement to solve an optimization model, but can also consist of a sequence of execution and flow control statements. An example of the attribute window of a procedure node within

Procedure and function attributes

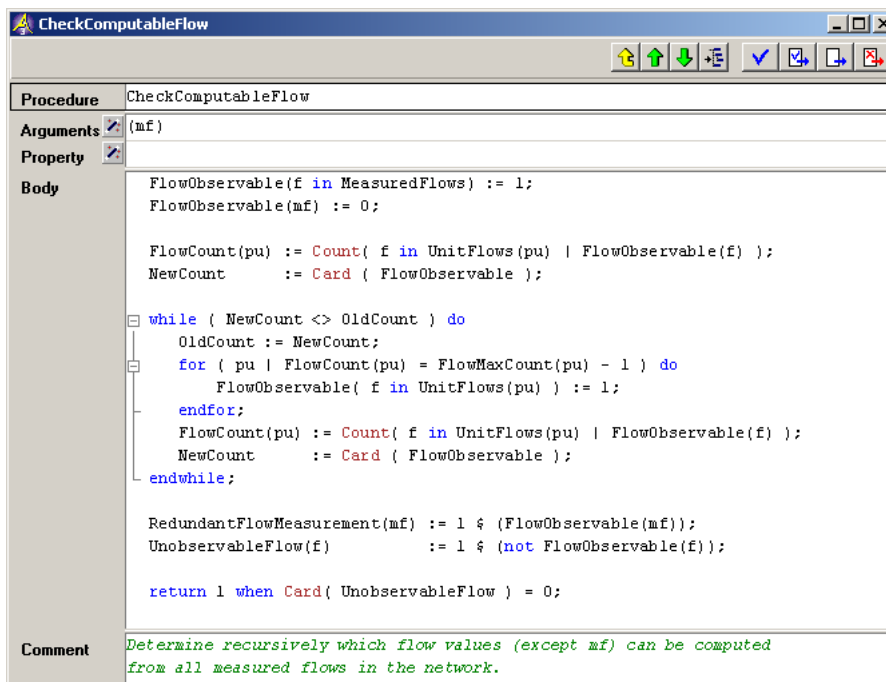


Figure 6.2: Example of procedure attributes

the model tree is illustrated in Figure 6.2. The contents of the **Body** attribute is application-specific, and is irrelevant to a further understanding of the material in this section.

When the resulting value of a function is multidimensional, you can specify the index domain and range of the result in the attribute form of the function using the **Index Domain** and **Range** attributes. Inside the function body you can make assignments to the function name as if it were a local (indexed) parameter, with the same dimension as specified in the **Index Domain** attribute. The most recently assigned values are the values that are returned by the function.

*Specifying
function domain
and range*

6.2 Declaration of arguments and local identifiers

All (formal) arguments of a procedure or function must be specified as a parenthesized, comma-separated, list of non-indexed identifier names. All formal arguments must also be declared as local identifiers in a declaration section local to the procedure or function. These local declarations then specify the further domain and range information of the arguments. If an argument has not been declared when you create (or modify) a procedure or function, AIMMS will open the dialog box illustrated in Figure 6.3 which helps you add the appropriate declaration quickly and easily. After completing the dialog box, AIMMS will

*Specifying
arguments*

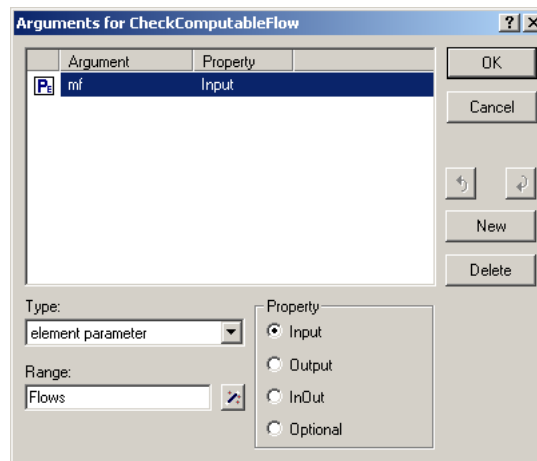


Figure 6.3: **Argument Declaration** dialog box

automatically add a declaration section to the procedure or function, and add the arguments displayed in the dialog box to it, as illustrated in Figure 6.1.

An important aspect of any argument is its input-output type, which can be specified by selecting one of the Input, InOut, Output or Optional properties in the **Argument Declaration** dialog box. The input-output type determines whether any data changes you make to the formal arguments are passed back to the actual arguments on leaving the procedure. The precise semantic meaning of each of these properties is discussed in the AIMMS Language Reference book.

Input-output type

The choices made in the **Argument Declaration** dialog box are directly reflected in the attribute form of the local identifier added to the model tree by AIMMS. As an example, Figure 6.4 shows the attribute form of the single argument `mf` of the procedure `CheckComputableFlow` added in Figure 6.3. In the dialog

Argument attributes

Attribute	Value
Type	Element Parameter
Identifier	mf
Index domain	
Text	
Range	Flows
Default	
Property	Input
Comment	

Figure 6.4: Declaration form of a procedure argument

box of Figure 6.3 it is not possible to modify the dimension of a procedure or function argument directly. If your procedure or function has a multidimensional argument, you can specify this with the **Index Domain** attribute of the argument after the argument has been added as a local identifier to the model tree.

For every call to the procedure or function, AIMMS will verify whether the types of all the actual arguments match the prototypes supplied for the formal arguments, including the supplied index domain and range. For full details about argument declaration refer to the AIMMS Language Reference book.

Prototype checking

In addition to arguments, you can also add other local identifiers to declaration sections within procedures and functions. Such local identifiers are only known inside the function or procedure. They are convenient for storing temporary data that is only useful within the context of the procedure or function, and have no global meaning or interpretation.

Local declarations

Not all identifier types can be declared as local identifiers of a procedure or function, because of the global implications they may have for the AIMMS execution engine. When you try to add a local identifier to a procedure or function, AIMMS will only offer those identifier types that are actually supported within a procedure or function. An example of an identifier type that cannot be declared locally is a constraint.

Not all types supported

In addition, for local identifiers, AIMMS may only support a subset of the attributes that are supported for global identifiers of the same type. For instance, AIMMS does not allow you to specify a **Definition** attribute for local sets and parameters. In the attribute window of local identifiers such non-supported attributes are automatically removed when you open the associated attribute form.

Not all attributes supported



6.3 Specifying the body

In the **Body** attribute of a procedure or function you can specify the

Statements

- assignments,
- execution statements such as SOLVE or READ/WRITE,
- calls to other procedures or functions in your model, and
- flow control statements such as FOR, WHILE or IF-THEN-ELSE

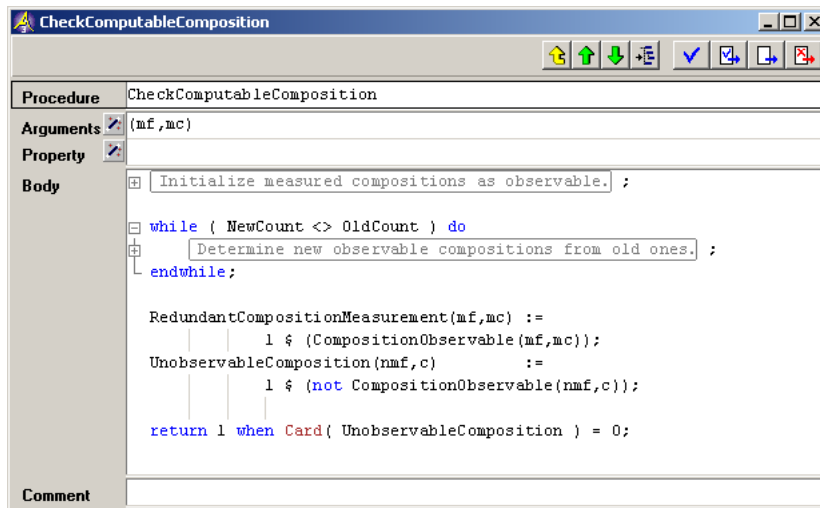
which perform the actual task or computation for which the procedure or function is intended. The precise syntax of all execution statements is discussed in detail in the AIMMS Language Reference book.

When you are constructing a procedure or function whose execution consists of a large number of (nested) statements, it may not always be easy or natural to break up the procedure or function into a number of separate procedures. To help you maintain an overview of such large pieces of execution code, AIMMS will automatically add outlining support for common flow control statement and multiline comments. The minus button  that appears in front of the statement allows you to collapse the statement to a single line block and the and plus button  allows you to expand the statement to its full extent again.

Automatic outlining

In addition, you can break down the body of a procedure or function into manageable pieces using one or more execution blocks. Any number of AIMMS statements enclosed between BLOCK and ENDBLOCK keywords can be graphically collapsed into a single block. The text in the single line comment following the BLOCK keyword is used as display text for the collapsed block. An example of a procedure body containing two collapsed execution blocks is given in Figure 6.5.

Execution blocks



When you are entering statements into a body of a procedure or function, AIMMS can help you to add identifier references to the body quickly and easily:

Identifier references

- you can drag and drop the names from the model tree into text
- with the name completion feature you can let AIMMS complete the remainder of the name based on only the first characters typed.

The precise details of drag-and-drop support and name completion of identifiers are discussed in Sections 4.3 and 5.2.

When you are entering the body of a procedure or function, you may want to review the attributes or current data of a particular identifier referenced in the body. AIMMS offers various ways to help you find such identifier details:

Viewing identifier details

- through a text based search in the model tree, you can locate the specific identifier node and open its attribute form (see Section 4.3),
- by clicking on a particular identifier reference in the body, you can open its attributes form through the **Attributes** item in the right-mouse pop-up menu,
- you can locate the identifier declaration in the model tree through the **Location in Model Tree** item in the right-mouse pop-up menu, and
- you can view (or modify) the identifier's data through the **Data** item in the right-mouse pop-up menu (see Section 5.4).

Similarly, while you are referencing a procedure or function inside the body of another procedure or function, AIMMS will provide prototype information of such a procedure or function as soon as you enter the opening bracket (or when you hover with the mouse pointer over the procedure or function name). This will pop up a window as illustrated in Figure 6.6. This tooltip window

*Viewing
procedure
details*

```

if ( AutoCheckRedundancy ) then
  Check For Redundant Measurements;
else
  Empty Redundant Flow Measurement, Redundant Composition Measurement;
  Check Computable Composition(
endif;

```

Check Computable Composition(
 [Input] mf AS Element Parameter,
 [Input] mc AS Element Parameter)

Figure 6.6: Prototype info of a procedure

displays all arguments of the selected procedure or function, their respective data types, as well as their *Input-Output* status. The latter enables you to assess the (global) effect on the actual arguments of a call to the procedure.

6.4 Syntax checking, compilation and execution

Using either **Check and commit** or **Check, commit and close** as discussed in Section 5.3 AIMMS will compile the procedure or function in hand, and point out any syntax error in its body. If you do not want to compile a procedure or function, but still want to commit the changes, you should use the **Commit and close** button. All edits are ignored when you close the window using the **Discard** button.

*Performing a
syntax check*

Before executing any procedure in your model, AIMMS will automatically verify whether your model needs to be recompiled, either partially or fully. In most cases, there is no need for AIMMS to recompile the entire model after a modification or addition of a new identifier, a procedure or a function. For instance, when you have only changed the body of a procedure, AIMMS needs only to recompile that particular procedure.

*Partial
recompilation*

However, if you change the index domain of an identifier or the number of arguments of a procedure or function, each reference to such an identifier, procedure or function needs to be verified for correctness and possibly changed. In such cases, AIMMS will (automatically) recompile the entire model before any further execution can take place. Depending on the size of your model, complete recompilation may take some time. Note that either partial or complete recompilation will only retain the data of all identifiers present prior to compilation, to the extent possible (data cannot be retained when, for instance, the dimension of an identifier has changed).

*Complete
recompilation*

AIMMS supports several methods to initiate procedural model execution. More specifically, you can run procedures

Running a procedure

- from within another procedure of your model,
- from within the graphical user interface by pressing a button, or when changing a particular identifier value, or
- by selecting the **Run procedure** item from the right-mouse menu for any procedure selected in the **Model Explorer**.

The first two methods of running a procedure are applicable to both developers and end-users. Running a procedure from within the **Model Explorer** a useful method for testing the correct operation of a newly added or modified procedure.