
AIMMS User's Guide - Debugging and Profiling an AIMMS Model

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 8

Debugging and Profiling an AIMMS Model

After you have developed an (optimization) model in AIMMS, it will most probably contain some unnoticed logical and/or programming errors. These errors can cause infeasible solutions or results that are not entirely what you expected. Also, you may find that the execution times of some procedures in your model are unacceptably high for their intended purpose, quite often as the result of only a few inefficiently formulated statements. To help you isolate and resolve such problems, AIMMS offers a number of diagnostic tools, such as a debugger and a profiler, which will be discussed in this chapter.

This chapter

8.1 The AIMMS debugger

When your model contains logical errors or programming errors, finding the exact location of the offending identifier declarations and/or statements may not be easy. In general, incorrect results might be caused by:

Tracking modeling errors

- incorrectly specified attributes for one or more identifiers declared in your model (most notably in the INDEX DOMAIN and DEFINITION attributes),
- logical oversights or programming errors in the formulation of one or more (assignment) statements in the procedures of your model,
- logical oversights or programming errors in the declaration of the variables and constraints comprising a mathematical program, and
- data errors in the parametric data used in the formulation of a mathematical program.

If the error is in the formulation or input data of a mathematical program, the main route for tracking down such problems is the use of the **Math Program Inspector** discussed in Chapter 9. Using the **Math Program Inspector** you can inspect the properties of custom selections of individual constraints and/or variables of a mathematical program.

Errors in mathematical programs

To help you track down errors that are the result of misformulations in assignment statements or in the definitions of defined parameters in your model, AIMMS provides a *source debugger*. You can activate the AIMMS debugger through the **Tools-Diagnostic Tools-Debugger** menu. This will add a **Debug-**

The AIMMS debugger

ger menu to the system menu bar, and, in addition, add the **Debugger** toolbar illustrated in Figure 8.1 to the toolbar area. You can stop the AIMMS debugger




Figure 8.1: The **Debugger** toolbar

through the **Debugger-Exit Debugger** menu.

Using the AIMMS debugger, you can

- set conditional and unconditional breakpoints on a statement within the body of any procedure or function of your model, as well as on the evaluation of set and parameter definitions,
- step through the execution of procedures, functions and definitions, and
- observe the effect of single statements and definitions on the data within your model, either through tooltips within the observed definitions and procedure bodies, or through separate data pages (see also Section 5.4)

*Debugger
functionality*

Within the AIMMS debugger you can set breakpoints on any statement in a procedure or function body (or on the definition of a defined set, parameter or variable) by selecting the corresponding source line in the body of the procedure or function, and choosing the **Debugger-Breakpoints-Insert/Remove** menu (or the **Insert/Remove Breakpoint** button  on the **Debugger** toolbar). After you have set a breakpoint, this is made visible by means of red dot in the left margin of selected source line, as illustrated in Figure 8.2.

*Setting
breakpoints in
the body*

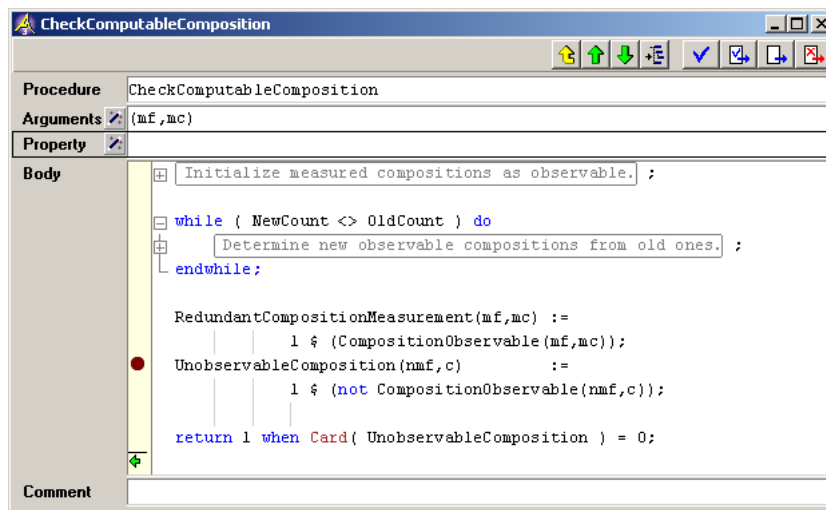


Figure 8.2: Setting a breakpoint in a procedure body

Alternatively, you can set a breakpoint on a procedure, function or on a defined set, parameter or variable by selecting the corresponding node in the **Model Explorer**, and choosing the **Debugger-Breakpoints-Insert/Remove** menu. As a result, AIMMS will add a breakpoint to the first statement contained in the body of the selected procedure or function. The name of a node of any procedure, function or defined set, parameter or variable with a breakpoint is displayed in red in the model tree, as illustrated in Figure 8.3.

Setting breakpoints in the model tree

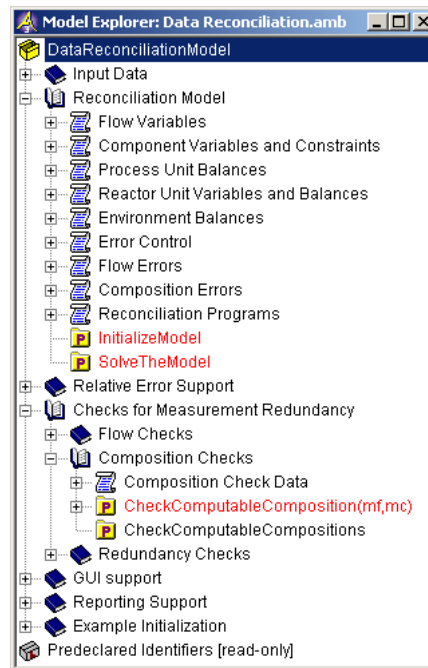



Figure 8.3: Viewing procedures with breakpoints in the **Model Explorer**

Once you have set a breakpoint in your model, AIMMS will automatically stop at this breakpoint whenever a line of execution arrives at the corresponding statement. This can be the result of

Entering the debugger

- explicitly running a procedure within the **Model Explorer**,
- pushing a button on an end-user page which results in the execution of one or more procedures, or
- opening an end-user (or data) page, which requires the evaluation of a defined set or parameter.

Whenever the execution stops at a breakpoint, AIMMS will open the corresponding procedure body (or the declaration form of the defined set, parameter or variable), and show the current line of execution through the breakpoint pointer , as illustrated in Figure 8.4.

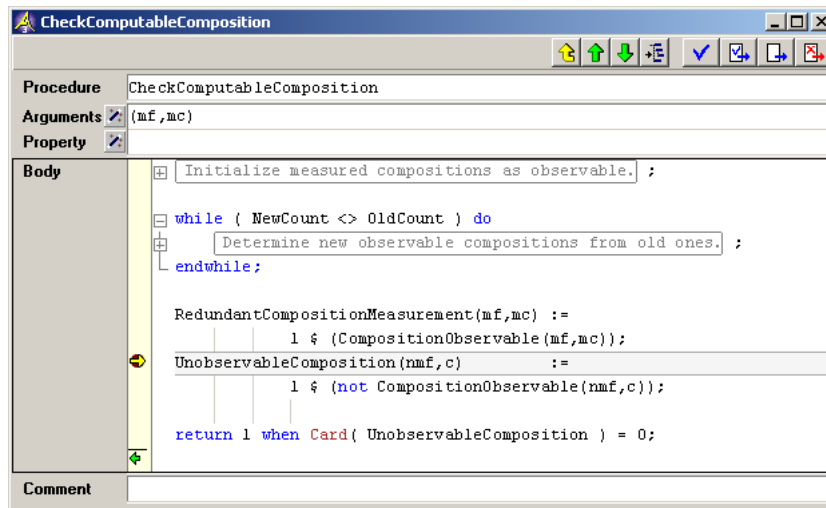
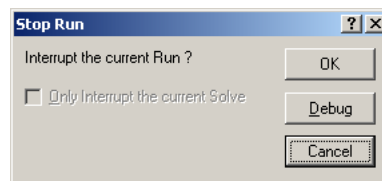


Figure 8.4: Arriving at a breakpoint

Even when you have not set breakpoints, you can still enter the debugger by explicitly *interrupting* the current line of execution through the **Run-Stop** menu (or through the **Ctrl-Shift-S** shortcut key). It will pop up the **Stop Run** dialog box illustrated in Figure 8.5 When you have activated the AIMMS debugger prior

Interrupting execution

Figure 8.5: The **Stop Run** dialog box

to execution, the **Debug** button on it will be enabled, and AIMMS will enter the debugger when you push it. By pushing the **OK** or **Cancel** button, AIMMS will completely stop or just continue executing, respectively.





The above method of interrupting AIMMS will not work when AIMMS is executing a statement or definition that takes a very long time. In that case you can interrupt AIMMS via the `AimmsInterrupt` tool. This tool is available from the Windows start All Programs menu. Upon startup, it will place itself in the system tray. By right-clicking the AIMMS system tray icon, you'll obtain a menu of running AIMMS instances that can be interrupted. In developer mode, the interrupted AIMMS will also popup a debugger showing where it has been interrupted. With that debugger, you can't continue execution, however; as the consistency of the values of the identifier(s) being computed during the interrupt can't be guaranteed. On the other hand, you can start new procedures. In

Interrupting slow statements




end-user mode, the interrupted AIMMS will just issue an error message, indicating the interrupted statement, definition or constraint.

Once AIMMS has interrupted a line of execution and entered the debugger, you can step through individual statements by using the various step buttons on the **Debugger** toolbar and follow the further flow of execution, or observe the effect on the data of your model. AIMMS offers several methods to step through your code:

Stepping through statements

- the **Step Over**  method runs a single statement, and, when this statement is a procedure call, executes this in its entirety,
- the **Step Into**  method runs a single statement, but, when this statement is a procedure call, sets the breakpoint pointer to the first statement in this procedure,
- the **Step Out**  method runs to the end of the current procedure and sets the breakpoint pointer to the statement directly following the procedure call in the calling context, and
- the **Run To Cursor**  method runs in a single step from the current position of the breakpoint pointer to the current location of the cursor, which should be *within the current procedure*.

In addition, AIMMS offers some methods to continue or halt the execution:

- the **Continue Execution**  method continues execution, but will stop at any breakpoint it will encounter during this execution,
- the **Finish Execution**  method finishes the current line of execution, ignoring any breakpoints encountered,
- the **Halt**  method immediately halts the current line of execution.

Whenever you are in the debugger, AIMMS allows you to interactively examine the data associated with the identifiers in your model, and observe the effect of statements in your source code. The most straightforward method is by simply moving the mouse pointer over a reference to an identifier (or identifier *slice*) within the source code of your model. As a result, AIMMS will provide an overview of the data contained in that identifier (slice) in the form of a tooltip, as illustrated in Figure 8.6. The tooltip will provide global information about the identifier slice at hand, such as

Examining identifier data

- its name and indices,
- the number of elements or non-default data values (in brackets), and
- the first few elements or non-default data value in the form of a list consisting of tuples and their corresponding values.

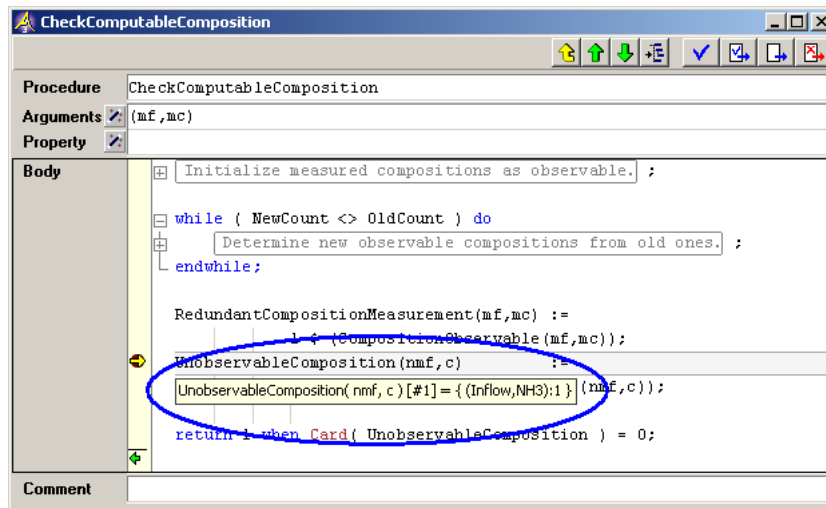



Figure 8.6: Observing the current data of an identifier through a tooltip

If you need to examine the effect of a statement on the data of a particular identifier in more detail, you can simply open a **Data Page**, as described in Section 5.4, or observe the effect on ordinary end-user pages. Within a debugger session, AIMMS supports data pages for both global and local identifiers, thereby allowing you to examine the contents of local identifiers as well. After each step in the debugger AIMMS will automatically update the data on any open end-user or data page.

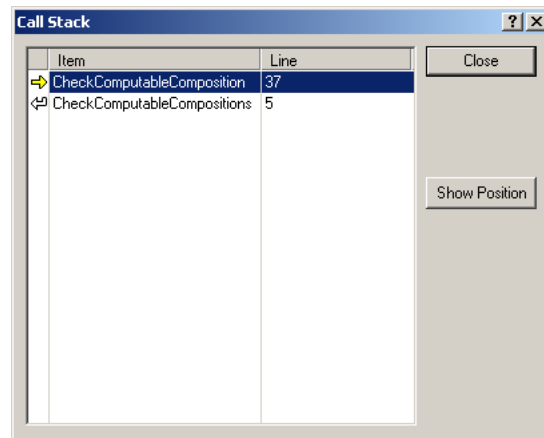
*Detailed
identifier data*


If you are not sure which statement in your model is responsible for changing the data of a (non-defined) set or parameter, you can set a breakpoint on such a set or parameter. Whenever a statement in your model changes the set or parameter data at hand, AIMMS will break on that statement. Notice, however, that breakpoint on data change will not pick up data changes that are due to set or parameter data becoming inactive because of changes to sets or parameters included in the domain or domain condition.

*Breakpoint on
data change*

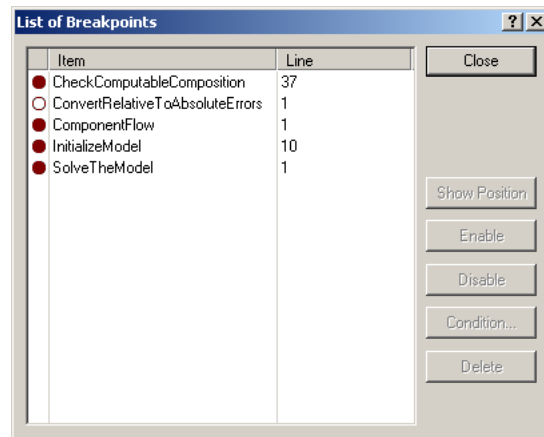
Whenever you are in the debugger, the **Call Stack** button  on the **Debugger** toolbar will display the **Call Stack** dialog box illustrated in Figure 8.7. With it you get a detailed overview of the stack of procedure calls associated with the current line of execution. It enables you to observe the flow of execution at the level of procedures associated with the current position of the breakpoint pointer. After selecting a procedure or definition in the **Call Stack** dialog box, the **Show Position** button will open its attribute window at the indicated line.

*Viewing the call
stack*

Figure 8.7: The **Call Stack** dialog box

After you have inserted a number of breakpoints into your model, you can get an overview of all breakpoints through the **Show All Breakpoints** button . This button will invoke the **List of Breakpoints** dialog box illustrated in Figure 8.8. For each breakpoint, AIMMS will indicate whether it is enabled or

Viewing and modifying breakpoints

Figure 8.8: The **List of Breakpoints** dialog box

disabled (i.e. to be ignored by the AIMMS debugger). Through the buttons on the right hand side of the dialog box you can

- disable breakpoints,
- enable previously disabled breakpoints,
- delete breakpoints, and
- create new breakpoints.

Alternatively, you can disable or remove all breakpoints simultaneously using the **Disable All Breakpoints** button  and the **Remove All Breakpoints**

button 

In addition, by pushing the **Condition** button on the **List of Breakpoints** dialog box, you can add a condition to an existing breakpoint. It will open the **Breakpoint Condition** dialog box illustrated in Figure 8.9. The condition must

Conditional breakpoints

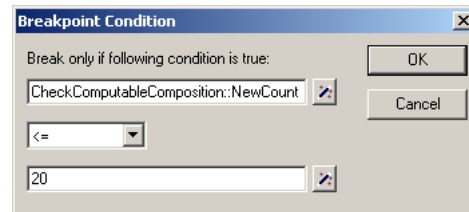


Figure 8.9: The **Breakpoint Condition** dialog box

consist of a simple numerical, element or string comparison. This simple comparison can only involve scalar identifiers, identifier slices or constants. Free indices in an identifier slice are only allowed when they are fixed within the breakpoint context (e.g. through a for loop). AIMMS will only stop at a conditional breakpoint, when the condition that you have specified is met during a particular call. Conditional breakpoints are very convenient when, for instance, a procedure is called very frequently, but only appears to contain an error in one particular situation which can be detected through a simple comparison.

8.2 The AIMMS profiler

Once your model is functionally complete, you may find that the overall computational time requirement set for the application is not met. If your application contains optimization, and most of the time is spent by the solver, finding a remedy for the observed long solution times may not be easy. In general, it involves finding a reformulation of the mathematical program which is more suitable to the selected solver. Finding such a reformulation may require a considerable amount of expertise in the area of optimization.

Meeting time requirements with solvers

It could also be, however, that optimization (if any) only consumes a small part of the total execution time. In that case, the time required for executing the application is caused by data manipulation statements. If total execution time is unacceptably high, it could be caused by inefficiently formulated statements. Such statements force AIMMS to fall back to *dense* instead of *sparse* execution. Chapters 12 and 13 of the Language Reference discuss the principles of the sparse execution engine used by AIMMS, and describe several common pitfalls together with reformulations to remedy them.

Meeting time requirements with data execution

AIMMS offers a profiler to help you resolve computational time related issues. The AIMMS profiler enables you to locate the most time-consuming evaluations of

The AIMMS profiler

- procedures and functions,
- individual statements within procedures and functions,
- defined sets and parameters, and
- constraints and defined variables during matrix generation.

You can activate the AIMMS profiler by selecting the **Tools-Diagnostic Tools-Profiler** menu, which will add a **Profiler** menu to the default system menu bar. If the debugger is still active at this time, it will be automatically deactivated, as both tools cannot be used simultaneously.

Activating the profiler

As soon as you have activated the profiler, AIMMS will start gathering timing information during every subsequent procedure run or definition evaluation, regardless whether these are initiated by pushing a button on an end-user page, by executing a procedure from within the **Model Explorer**, or even by means of a call to the AIMMS API from within an external DLL.

Gathering timing information

After you have gathered timing information about your modeling application by executing the relevant parts of your application at least once, you can get an overview of the timing results through the **Profiler-Results Overview** menu. This will open the **Profiler Results Overview** dialog box illustrated in Figure 8.10. In it, you will find a list of all procedures that have been executed and identifier definitions that have been evaluated since the profiler was activated.

Viewing profiler results

For each procedure, function, or defined identifier listed in the **Profiler Results Overview** dialog box, AIMMS will provide the following information:

Detailed timing information

- the number of hits (i.e. the number of times a procedure has been executed or a definition has been evaluated),
- the total gross time (explained below) spent during all hits,
- the total net time (explained below) spent during all hits,
- the average gross time spent during each separate hit, and
- the average net time spent during each separate hit.

The term *gross time* refers to the total time spent in a procedure *including* the time spent in procedure calls or definition evaluations within the profiled procedure. The term *net time* refers to the total time spent *excluding* the time spent in procedure calls or definition evaluations within the profiled procedure.

Gross versus net time

Name	hits	gross time	net time	average gross time	average net time
SolveTheModel	1	0.243	0.187	0.243	0.187
CheckComputableComp	9	0.110	0.091	0.012	0.010
CheckForRedundantMea	1	0.079	0.000	0.079	0.000
CheckComputableComp	1	0.033	0.000	0.033	0.000
InitializeModel	1	0.023	0.022	0.023	0.022
CheckComputableFlows	9	0.018	0.021	0.002	0.002
EnvironmentMassBalanc	2	0.017	0.017	0.009	0.009
CheckComputableFlow	12	0.017	0.031	0.001	0.003
TotalAbsoluteError	2	0.004	0.004	0.002	0.002
ComponentFlow	1	0.002	0.002	0.002	0.002
MeasurementFlowColor	2	0.002	0.002	0.001	0.001
ComponentBalance	1	0.002	0.002	0.002	0.002
CompositionsAddUpToO	1	0.002	0.002	0.002	0.002
MolarFlowMass	1	0.001	0.001	0.001	0.001
CompositionErrorDeterm	1	0.001	0.001	0.001	0.001
MassBalance	2	0.001	0.001	0.000	0.000
FlowErrorDetermination	2	0.001	0.001	0.000	0.000
ConvertRelativeToAbsol	1	0.001	0.000	0.001	0.000
ElementBalance	1	0.001	0.001	0.001	0.001
EnvironmentReactantBa	1	0.001	0.001	0.001	0.001
CompositionMeasuremer	1	0.000	0.000	0.000	0.000
ObservedCompositionEn	1	0.000	0.000	0.000	0.000
ReactantCreated	1	0.000	0.000	0.000	0.000
EnvironmentNonReactar	1	0.000	0.000	0.000	0.000

Figure 8.10: The **Profiler Results Overview** dialog box

With this timing information you can try to locate the procedures and identifier definitions which are most likely to benefit from a reformulation to improve efficiency. To help you locate these procedures and definitions, the list of procedures and definitions in the **Profiler Results Overview** dialog box can be sorted with respect to all its columns. The most likely criterion for this is to sort by decreasing net time or average net time, which will identify those procedures and identifier definitions which take up the most time by themselves, either in total or for each individual call. You can open the attribute form of any identifier in the **Profiler Results Overview** dialog box by simply double-clicking on the corresponding line.

*Locating
time-consuming
procedures*

When you have located a time-consuming procedure, you can open its attribute form and try to locate the offending statement(s). Whenever the profiler has been activated, AIMMS will add additional profiling columns to the body of a procedure, as illustrated in Figure 8.11. Similarly, AIMMS will add these profiling columns to the definition attributes of defined identifiers.

*Locating
offending
statements*

For each statement in the body of a procedure, AIMMS can display various types of profiling data in the profiling columns of an attribute form. As you can see next, this information is even more extensive than for procedures as a whole. The following information is available:

*Profiling column
information*

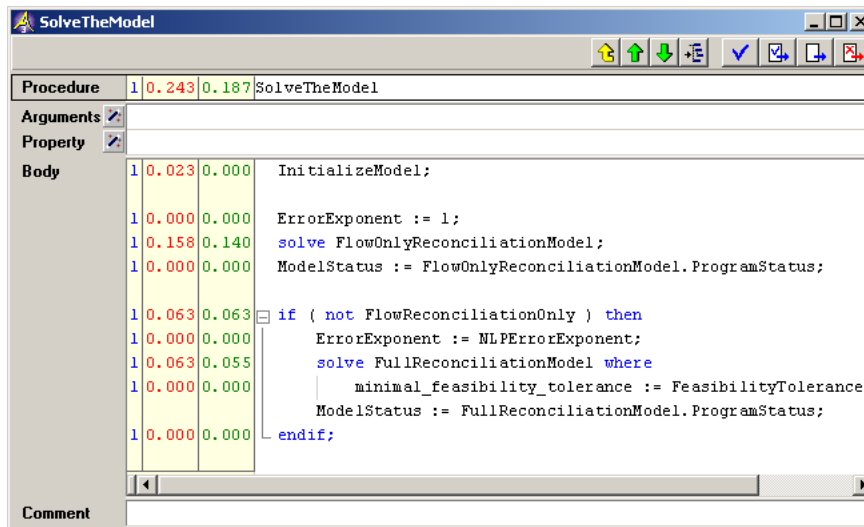


Figure 8.11: Profiling information in an attribute form

- the number of hits (i.e. the number of times a particular statement has been executed),
- the total gross time spent during all hits,
- the total net time spent during all hits,
- the average gross time spent during each separate hit,
- the average net time spent during each separate hit,
- the actual number of expression evaluations,
- the maximum number of expression evaluations, and
- the sparsity index (i.e. actual number of evaluations as percentage of maximum number of evaluations).

In the context of a procedure body, the difference between gross and net time need not always refer only to the time spent in other procedures (as in the **Profiler Results Overview** dialog box). For selected statements both numbers may have a somewhat different, yet meaningful, interpretation. The list of exceptions is:

Gross versus net time for particular statements

- in flow control statements such as the IF, WHILE and FOR statement (see also Section 8.3 of the Language Reference), the net time refers to the time required to evaluate the statement itself (for instance, its condition) whereas the gross time refers to the time required to execute the entire statement,
- in the SOLVE statement (see also Section 15.3 of the Language Reference), the net time refers to the time spent in the solver, while the gross time refers to the time spent in the solver plus the time required to generate the model.

The actual and maximum expression evaluation count and the sparsity index, may give you a valuable clue why a particular statement takes a long time to execute. The key observation in reducing execution times is that you should avoid the *dense* execution of statements as much as possible, especially when the domain of execution is high-dimensional. You will find detailed information about sparse versus dense execution, as well as many useful hints how to avoid dense execution, in the Chapters 12 and 13 of the Language Reference.

*Counting
expression
evaluations*

The expression evaluation count forms a good measure for detecting dense execution. They provide the following information.

*Interpretation
of evaluation
counts*

- The *actual expression evaluation count* is the number of expression evaluations as counted by AIMMS during all evaluations of a particular statement. This number can depend on the cardinalities of all sets and multi-dimensional identifiers involved in the statement.
- The *maximum expression evaluation count* is the number of actual expression evaluations that would be necessary if the statement were to be executed in a dense manner.
- The *sparsity index* displays the actual evaluation count as a percentage of the maximum evaluation count.

You can use the following strategy to locate and reformulate time-consuming, densely executed, statements.

*Locating
densely
executed
statements*

- Locate those statements in a time-consuming procedure that take a disproportional amount of time.
- If the maximum expression evaluation count and the sparsity index are high, then you have very likely found a high-dimensional statement that is executed in a completely or nearly dense manner.
- Search in Chapter 13 whether the statement contains a construct which forces AIMMS to execute this statement in a dense manner and follow the associated instructions to reformulate the construct.
- If you cannot find the reason for the disproportional execution time, contact Paragon for their expert advise.

In addition to observing the profiling times in the **Profiler Results Overview** and through the profiling columns in attribute windows, AIMMS also provides profiling tooltips in both the **Model Explorer** and in the attribute windows of procedures and defined identifiers, as long as the profiler is active. An example of a profiling tooltip is given in Figure 8.12. Profiling tooltips can provide a convenient method to quickly observe the profiling information without requiring any further interaction with AIMMS. If you do not want AIMMS to display profiling tooltips while moving your mouse through either the **Model Explorer** or procedure bodies, you can disable them through the **Profiler Setup** dialog box described below, by unchecking the **Show Profiler Values** check mark (see also Figure 8.13).

Profiler tooltips

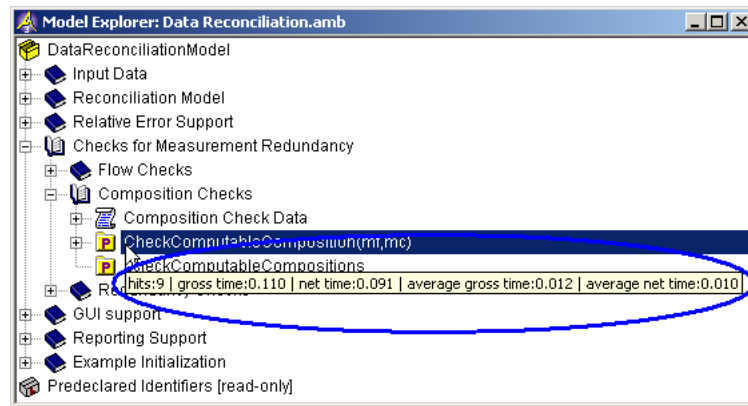


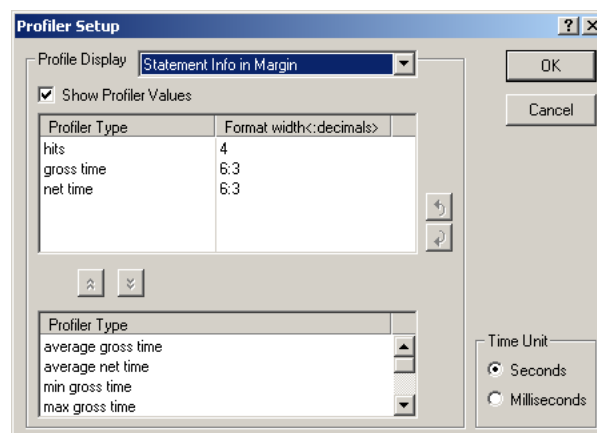
Figure 8.12: Observing profiling information through a tooltip

If you are interested in a profiling overview comprising your entire modeling application, you can get this through the **Profiler-Create Listing File** menu. This will create a common source listing file of your model text extended with profiling information wherever this is available. Through the **Profiler Setup** dialog box described below you can determine which profiler information will be added to the profiler listing.

Profiler listing

For every new project, AIMMS uses a set of default settings to determine which profiling information is displayed in the various available methods to display profiling information. You can modify these settings through the **Profiler-Setup** menu, which will open the **Profiler Setup** dialog box illustrated in Figure 8.13. In this dialog box you can, on a project-wide basis, determine

*Setting up
profiling
columns*

Figure 8.13: The **Profiler Setup** dialog box

- which of the profiling display methods described are enabled (through

- the **Show Profiler Values** check mark), and
- per such display method, which profiling information is to be displayed, their order, and their corresponding display format.

The settings selected in the **Profiler Setup** dialog box are saved along within the project file, and will be restored when you reopen the project in another AIMMS session.

Through the **Profiler-Pause** menu, you can temporarily halt the gathering of profiling information by AIMMS, while the existing profiling information will be retained. You can use this menu, for example, when you only want to profile the core computational procedures of your modeling application and do not want the profiling information to be cluttered with profiling information that is the result of procedure calls and definition evaluations in the end-user interface. You can resume the gathering of profiling information through the **Profiler- Continue** menu.

Pausing and continuing the profiler

With the **Profiler-Reset** menu, you can completely reset all profiling counters to zero. You can use this menu, if the profiling information of your application has become cluttered. For instance, some procedures may have been executed multiple times and, thus, disturb the typical profiling times required by your entire application. After resetting the profiling counters, you can continue to gather new profiling information which will then be displayed in the various profiling displays.

Resetting the profiler

You can completely disable the AIMMS profiler through the **Profiler-Exit Profiler** menu. As a result, the gathering of profiling information will be completely discontinued, and all profiling counters will be reset to zero. Thus, when you restart the profiler, all profiling information of a previous session will be lost.

Exiting the profiler

8.3 Observing identifier cardinalities

Another possible cause of performance problems is when one or more multi-dimensional identifiers in your model have missing or incorrectly specified domain conditions. As a result, AIMMS could store far too much data for such identifiers. In addition, computations with these identifiers may consume a disproportional amount of time.

Observing identifier cardinalities

To help you locate identifiers with missing or incorrectly specified domain conditions, AIMMS offers the **Identifier Cardinalities** dialog box illustrated in Figure 8.14. You can invoke it through the **Tools-Diagnostic Tools-Identifier Cardinalities** menu.

Locating cardinality problems

Identifier	Cardinality	Maximal	Density	Active	Inactive	Mem Usage
MeasurementFlowColor(f,mc)	35	35 (7x5)	100.00%	35	0	1.00 Kb
Admissable(pu,c,f,g)	32	784 (4x4x7x7)	4.08%	32	0	1.75 Kb
CompositionErrorColor(f,c)	28	28 (7x4)	100.00%	28	0	1016
MappedComposition(f,c)	22	28 (7x4)	78.57%	22	0	920
Composition(nmf,c)	22	28 (7x4)	78.57%	22	0	1.00 Kb
ComponentFlow(f,c)	22	28 (7x4)	78.57%	22	0	1.00 Kb
MappedFlowComponent(f,c)	22	28 (7x4)	78.57%	22	0	920
IndicatorFlowComponent(nmf,c)	22	28 (7x4)	78.57%	22	0	1.25 Kb
NodeCoordinate(n,crd)	14	14 (7x2)	100.00%	14	0	904
CompositionObservableAtUnitCount(pu,c)	13	16 (4x4)	81.25%	13	0	672
CompositionMeasurementCount(f,mcat)	11	21 (7x3)	52.38%	11	0	832
MolarFlowMass(nmf)	7	7	100.00%	7	0	680
MappedFlowComponents(f)	7	7	100.00%	7	0	784
NodeMap(u)	7	7	100.00%	7	0	568
NonMappedFlows	7	7	100.00%	7	0	64
Flow(f)	7	7	100.00%	7	0	680
IndicatorMeasuredComponent(nmf,c)	7	28 (7x4)	25.00%	7	0	760
AbsoluteCompositionErrorDeviation(nmf,c)	7	28 (7x4)	25.00%	7	0	760
NodeColor(n)	7	7	100.00%	7	0	560
RelativeCompositionErrorDeviation(nmf,c)	7	28 (7x4)	25.00%	7	0	760
FlowComponents(nmf)	7	7	100.00%	7	0	784

Figure 8.14: The **Identifier Cardinalities** dialog box

The **Identifier Cardinalities** dialog box displays the following information for each identifier in your model:

Available information

- the *cardinality* of the identifier, i.e. the total number of non-default values currently stored,
- the *maximal cardinality*, i.e. the cardinality if all values would assume a non-default value,
- the *density*, i.e. the cardinality as a percentage of the maximal cardinality,
- the number of *active* values, i.e. of elements that lie within the domain of the identifier,
- the number of *inactive* values, i.e. of elements that lie outside of the domain of the identifier, and
- the *memory usage* of the identifier, i.e. the amount of memory needed to store the identifier data.

The list of identifier cardinalities can be sorted with respect to any of these values.

You can locate potential dense data storage problems by sorting all identifiers by their cardinality. Identifiers with a very high cardinality and a high density can indicate a missing or incorrectly specified domain condition. In most real-world applications, the higher-dimensional identifiers usually have relatively few tuples, as only a very small number of combinations have a meaningful interpretation.

Locating dense data storage

If your model contains one or more identifiers that appear to demonstrate dense data storage, it is often possible to symbolically describe the appropriate domain of allowed tuple combinations. Adding such a condition can be helpful to increase the performance of your model by reducing both the memory usage and execution times.

Resolving dense storage

Another type of problem that you can locate with the **Identifier Cardinalities** dialog box, is the occurrence of *inactive* data in your model. Inactive data can be caused by

Locating inactive data

- the removal of elements in one or more domain sets, or
- data modifications in the identifier(s) involved in the domain restriction of the identifier.

In principle, inactive data does not directly influence the behavior of your model, as the AIMMS execution engine itself will never consider inactive data. Inactive data, however, can cause unexpected problems in some specific situations.

Problems with inactive data

- One of the areas where you have to be aware about inactive data is in the AIMMS API (see also Chapter 32 of the Language Reference), where you have to decide whether or not you want the AIMMS API to pass inactive data to an external application or DLL.
- Also, when inactive data becomes active again, the previous values are retained, which may or may not be what you intended.

As a result, the occurrence of inactive data in the **Identifier Cardinalities** dialog box may make you rethink its consequences, and may cause you to add statements to your model to remove the inactive data explicitly (see also Section 23.3 of the Language Reference).