

---

## **AIMMS Language Reference - ASCII Reports and Output Listings**

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](http://www.aimms.com) or order your hard-copy at [www.lulu.com/aimms](http://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](mailto:info@aimms.com)  
WWW: [www.aimms.com](http://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  $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  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 29

# ASCII Reports and Output Listing

The AIMMS system has several reporting features to present model results to you or an end-user.

*Reporting facilities*

- The *graphical (end-)user interface* lets you not only view your model results, but also change input values and run the model interactively. In general, the graphical user interface is the most convenient and direct way to verify model results and view the effect of input changes.
- A *print page* allows you to obtain a hard-copy of your graphical model results. It is created in the graphical user interface of AIMMS and can contain the same objects as pages in the end-user interface. Single print pages or reports composed of multiple print pages can be printed either from within the end-user interface or from within the model. Printing pages and the available functions that you can use in your model to initiate printing is discussed in the AIMMS User's Guide.
- An *ASCII report* lets you save your model results in files. It is created as part of your model using PUT and DISPLAY statements. The result can be written to either a file or to a text window in the graphical user interface. ASCII reports are convenient, for instance, when you need to generate a special format input file for an external program.
- The *listing file* lets you view the contents of all constraints and variables of a particular mathematical program in your model just before or after solving it. The listing file is a convenient medium for debugging the precise contents of the constraints in a mathematical program generated on the basis of your model and data.

This chapter concentrates on the last two reporting media. It explains how to create and print ASCII reports. More specifically, it discusses the FILE declaration, as well as the PUT and DISPLAY statements. It also explains how you can optionally create an ASCII report consisting of pages each built up of a header, footer and data area. The remaining part of the chapter will explain the format of the constraint and solution listings generated by AIMMS.

*This chapter*

## 29.1 The FILE declaration

External file names that you want to use for reporting must be linked to AIMMS identifiers in your model. In this way, external file names become data. Whenever you want to send output to a particular external file, you must refer to its associated identifier. This linking is achieved using a FILE declaration, the attributes of which are given in Table 29.1.

*FILE declaration*

Attribute	Value-type	See also page
NAME	<i>string-expression</i>	
DEVICE	disk, Disk(ASCII), Disk(Unicode), window, void	
MODE	replace, merge	
TEXT	<i>string</i>	19
COMMENT	<i>comment string</i>	19
CONVENTION	<i>convention</i>	377, 462

Table 29.1: FILE attributes

With the NAME attribute you can specify the actual name of the disk file or window that you want to refer to. If the file identifier refers to a disk file, the NAME will be the file name on disk. If it refers to a window the NAME attribute will serve as the title of the window. If you do not specify a name, AIMMS will construct a default name, using the internal identifier name as the root and “.put” as the extension.

*The NAME attribute*

The DEVICE attribute can have five values

- disk (default),
- Disk(ASCII),
- Disk(Unicode),
- window, and
- void.

*The DEVICE attribute*

You can use it to indicate whether the output should be directed to an external file on disk, a window in the graphical user interface, or whether no output should be generated at all. This latter void device is very convenient, for instance, to hide output statements in your code that are useful during the development of your model but should not be displayed in an end-user version. The Disk(ASCII) and Disk(Unicode) devices are only relevant for the AIMMS Unicode version (see Section 23.2 of the User’s Guide).

You can use the `MODE` attribute to specify whether the file or window should be overwritten (replace mode, default), or appended to (merge mode). The graphical window in the user interface differs from a file in that it can be closed manually by the user. In this case, its contents are lost and AIMMS starts writing to a new instance regardless of the mode.

*The `MODE` attribute*

The following `FILE` declarations illustrate the declaration of a link to the external file “result.dat” in the `Output` subdirectory of the project directory, and a text window that will appear with the title “Model results”. The contents of `ResultFile` will be overwritten whenever it is opened, while the window `ResultWindow` will be appended to whenever possible.

*Example*

```
FILE:
  identifier : ResultFile
  name      : "Output\\result.dat"
  device    : disk
  mode      : replace ;

FILE:
  identifier : ResultWindow
  name      : "Model results"
  device    : window
  mode      : merge ;
```

With the `CONVENTION` attribute you can indicate that AIMMS must assume that the data in the file is to be stored according to the units provided in the specified convention. If the unit specified in the convention differs from the unit in which AIMMS stores its data internally, the data is scaled just prior to data transfer. For the declaration of `CONVENTIONS` you are referred to Section 30.8.

*The `CONVENTION` attribute*

---

## 29.2 The `PUT` statement

AIMMS provides two statements to create a customized ASCII output report in either a file or in a text window in the user interface. They are the `PUT` and the `DISPLAY` statements. The result of these statements must always be directed to either a single file or a window.

*Customized ASCII reports*

The following steps are required to create a customized ASCII report:

*Basic steps*

- direct the output to the appropriate `FILE` identifier, and
- print one or more strings, set elements, numerical items, or tabular arrangements of data to it.

These basic operations are the subject of the subsequent subsections. At the end of the section, an extended example will illustrate most of the discussed features.

AIMMS can produce ASCII reports in two modes. They are:

- *stream mode*, in which all lines are printed consecutively, and
- *page mode*, where the report is divided into pages of equal length, each consisting of a header, a footer and a data area.

*Stream versus  
page mode*

Most aspects, such as opening files, output direction, and formatting, are the same for both reporting modes. Only the structuring of pages is an extra aspect of the page mode, and is discussed in Section 29.4.

---

### 29.2.1 Opening files and output redirection

Disk files and windows are opened automatically as soon as output is written to them. You can send output to a particular file by providing the associated FILE identifier as the first argument of a PUT statement, which designates the file as the *current file*.

*Opening files*

If you use the DISPLAY statement or any of the PUT operators listed in Table 29.2 without a file identifier, AIMMS will direct the output to the current file, i.e., the file last opened through the PUT statement.

*PUT without file  
identifier*

When you have not yet selected a current file yet, AIMMS will send the output of any PUT or DISPLAY statement to the standard listing file associated with your model.

*Undirected  
output*

The following statements illustrates how to send output to a particular file.

*Example*

```
PUT ResultFile ;
PUT "The model results are:" ;
DISPLAY Transport ;
PUTCLOSE;
```

The first PUT statement sets the current file equal to ResultFile, causing the output of the subsequent PUT and DISPLAY statements to be directed to it. The final PUTCLOSE statement will close ResultFile.

Unlike other statements like READ and WRITE which allow you to represent files by strings or string parameters as well, the PUT statement requires that you use a FILE identifier to represent the output file. The way in which output to a file is generated by the PUT statement is completely controlled by the suffices associated with the corresponding FILE identifier (see also Section 29.4).

*File identifiers  
only*

AIMMS has two pre-defined identifiers that provide access to the current file. They are

*Accessing the current file*

- the element parameter `CurrentFile` (into the set `AllIdentifiers`) containing the current FILE identifier, and
- the string parameter `CurrentFileName`, containing the file name or window title associated with the current file identifier.

The parameter `CurrentFileName` is output only.

To select another current file, you can use either of two methods:

*Changing the current file*

- use the PUT statement to (re-)direct output to a different file, or
- set the identifier `CurrentFile` to the FILE identifier of your choice.

Closing an external file can be done in two ways:

*Closing external files*

- automatically, by quitting AIMMS at the end of a session, or
- manually by calling “`PUTCLOSE file-identifier`” during execution.

If you leave a file open during the execution of a procedure, AIMMS will temporarily close it at the end of the current execution, and re-open it in *append mode* at the beginning of a subsequent execution. This enables you to inspect the PUT files in between runs.

*Files left open*

---

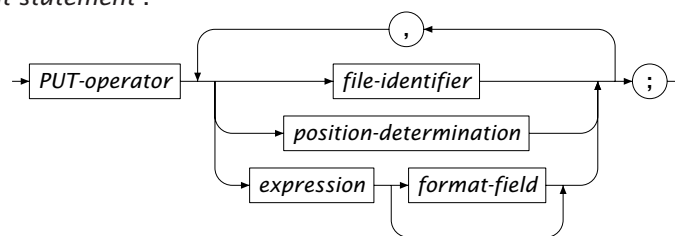
## 29.2.2 Formatting and positioning PUT items

Besides selecting the current file, the PUT statement can be used to output one or more individual strings, numbers or set elements to an external ASCII file or window. Each item can be printed in either a default or in a customized manner. The syntax of the PUT statement follows.

*The PUT statement*

*put-statement :*

*Syntax*



All possible variants of the PUT operator are listed in Table 29.2. The PUT and PUTCLOSE operators can be used in both stream mode and page mode. The operators PUTHD, PUTFT and PUTPAGE only make sense in page mode, and are discussed in Section 29.4.

*PUT operators*

Statement	Description	Stream mode	Page mode
PUT	Direct output or write output	•	•
PUTCLOSE	PUT and close current file	•	•
PUTHD	Write in header area		•
PUTFT	Write in footer area		•
PUTPAGE	PUT and output current page		•

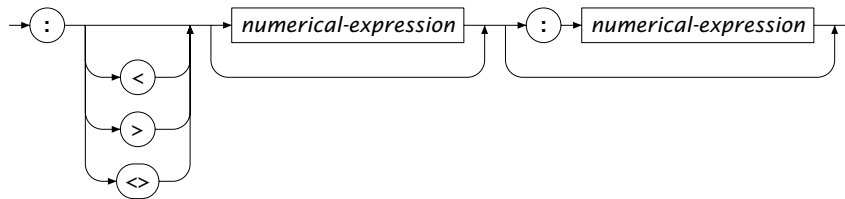
Table 29.2: PUT keywords

All PUT operators only accept scalar expressions. For each scalar item to be printed you can optionally specify a format field, with syntax:

*Put items are always scalar*

*format-field :*

*Syntax*



With the format field you specify

*Format fields*

- whether the item is to be centered, left aligned or right aligned,
- the field width associated with an identifier, and
- the precision.

Customized default values for the justification, field width and precision can be specified through PUT-related options, which can be set via the Options menu. Table 29.3 shows a number of examples of format fields, where  $m$  and  $n$  are expressions evaluating to integers.

For numerical expressions the precision is the number of decimals to be displayed. For strings and set elements the precision is the maximum number of characters to be displayed. The numbers or characters are placed into a field with the indicated width, and are positioned as specified.

*Interpretation of precision*

PUT argument	Justification	Field width (characters)	Precision
<i>item</i>	default	default	default
<i>item:m</i>	default	<i>m</i>	default
<i>item:m:n</i>	default	<i>m</i>	<i>n</i>
<i>item:&lt;m:n</i>	left	<i>m</i>	<i>n</i>
<i>item:&gt;m:n</i>	right	<i>m</i>	<i>n</i>
<i>item:&lt;&gt;m:n</i>	centered	<i>m</i>	<i>n</i>

Table 29.3: Format specification of PUT arguments

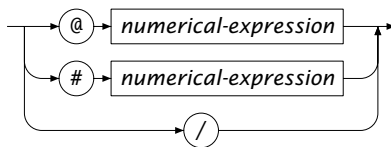
The PUT syntax for formatting and displaying multiple items on a single line is somewhat similar to the reporting syntax in programming languages like FORTRAN or PASCAL. If you are a C programmer, you may prefer to construct and format a single line of text using the `FormatString` function (see also Section 5.3.2). In this case you only need the PUT statement to send the resulting string to an ASCII report or window.

*Using the  
FormatString  
function*

For advanced reporting the PUT statement allows you to directly position the cursor at a given row or column. The syntax is shown in the following syntax diagram.

*Position  
determination*

*position-determination :*



There are three special arguments for the PUT statement that can be used to position printable items in a file:

*How to position*

- the “@” operator—for horizontal positioning on a line,
- the “#” operator—for vertical positioning, and
- the newline operator “/”.

These three operators are explained in Table 29.4, where the symbols *k* and *l* are expressions evaluating to integers.

Using the vertical positioning operator # only makes sense when you are printing in page mode. When printing in stream mode all lines are numbered consecutively from the beginning of the report, and added to the output file or window as soon as AIMMS encounters the newline character /. In page mode, AIMMS prints pages in their entirety, and lines are numbered per page. As a result, you can write to any line within the current page.

*Page mode only*

Operator	Meaning
@k	Start printing the next item at column k of the current line.
#l	Goto line l of current page (page mode only).
/	Goto new line.

Table 29.4: Position determination

### 29.2.3 Extended example

This example builds upon the transport model used throughout the manual. The following group of statements will produce an ASCII report containing the contents of the identifiers Supply(i), Demand(j) and Transport(i,j), in a combined tabular format separated into right aligned columns of length 12.

*Example*

```
! Direct output to ResultFile
put ResultFile ;

! Construct a header for the table
put @13, "Supply":>12, @30, "Transport":>12, /, @30 ;

for ( j ) do  put j:>12 ;  endfor ;
put // ;

! Output the values for Demand
put "Demand", @30 ;
for ( j ) do  put Demand(j):>12:2 ;  endfor ;
put // ;

! Output Supply and Transport
for ( i ) do
  put i:<12, Supply(i):>12:2, @30 ;
  for ( j ) do  put Transport(i,j):>12:2 ;  endfor ;
  put / ;
endfor ;

! Close ResultFile
putclose ResultFile ;
```

*The statements*

For a particular small data set containing only three Dutch cities, the above statements could result in the following report being generated.

*The produced report*

	Supply	Transport Amsterdam	Rotterdam	Den Haag
Demand		5.00	10.00	15.00
Amsterdam	10.00	2.50	2.50	5.00
Rotterdam	12.50	2.50	5.00	5.00
Den Haag	7.50	0.00	2.50	5.00

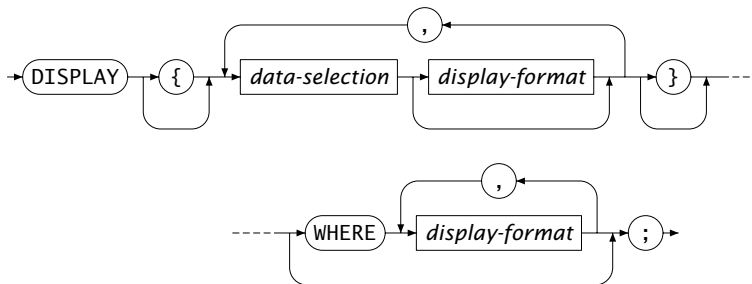
### 29.3 The DISPLAY statement

You can use the DISPLAY statement to print the data associated with sets, parameters and variables to a file or window in AIMMS format. As this format is also very easy to read, the DISPLAY statement is an excellent alternative for printing indexed identifiers.

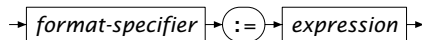
*Output in  
AIMMS format*

*display-statement :*

*Syntax*



*display-format :*



All data selections of a DISPLAY statement are printed by AIMMS in the form of a data assignment.

*Display format*

- Sets are printed in the form of a set assignment with an *enumerated set* on the right-hand side.
- (Slices of) parameters and variables are printed in the form of data assignments, which can be either a table format, a list format, or a composite table.

For indexed parameters and variables AIMMS uses a default display format which is dependent on the dimension.

You can override the default AIMMS format by specifying a *display format*, consisting of one or more format specifications, in the WHERE clause. AIMMS supports the following format specifiers:

*Overriding the  
display format*

- DECIMALS: the number of decimals to be printed for each entry,
- ROWDIM: the dimension of the row space,
- COLDIM: the dimension of the column space, and
- COLSPERLINE: the desired numbers of columns per line.

When a format specifier is not specified, AIMMS will use the system default.

All format specifications in a WHERE clause are applied to the entire collection of data selections printed in the DISPLAY statement. By specifying a DECIMAL format specifier for a particular data selection in the DISPLAY statement, you can also override the number of decimals printed for each data selection individually. You cannot specify other format specifiers for individual data selections.

*Number of  
decimals*

If you have set the dimension of either the row or column space to zero, AIMMS will print the identifier in list format. If both the dimension of the row and column space are greater than zero, AIMMS will print the identifier as a table. AIMMS will honor your request to print the desired number of columns per line if the resulting width does not exceed the default page width. In the latter case, AIMMS will reduce the number of columns until they fit within the requested page width. The default page width can be set as an option within your project.

*Obtaining lists  
and tables*

If the sum of the dimensions of the row and column space is less than the dimension of the parameter or variable to be displayed, AIMMS will display the identifiers as slices of the requested format, where the slices are taken by fixing the first indices in the domain.

*Outer indices  
for slicing*

When all arguments of the DISPLAY statement have the same domain and you enclose them by braces, AIMMS will print their values as a single composite table. In this case, you can only specify the precision with which each column must be printed. AIMMS will ignore any of the other display options in combination with the composite table format.

*Composite  
tables*

The following statements illustrate the use of the DISPLAY statement and its various display options.

*Example*

- The following statement will display the data of the variable Transport with 2 decimals and in the default format.

```
display Transport where decimals := 2;
```

The execution of this statement results in the following output being generated.

```
Transport :=
data table
      Amsterdam  Rotterdam  'Den Haag'
! -----
Amsterdam      2.50         2.50         5.00
Rotterdam      2.50         5.00         5.00
'Den Haag'          2.50         5.00
;
```

- The following statement displays the subselection of the slice of the variable Transport consisting of all transports departing from the set LargeSupplyCities.

```
display Transport(i in LargeSupplyCities,j) where decimals := 2;
```

This statement will result in the following table, assuming that LargeSupplyCities contains only Amsterdam and Rotterdam.

```
Transport :=
data table
      Amsterdam Rotterdam 'Den Haag'
! -----
Amsterdam 2.50      2.50      5.00
Rotterdam 2.50      5.00      5.00
;
```

- The following DISPLAY statement displays Transport with no rows, two columns (i.e. in list format), and two entries per line.

```
display Transport where decimals:=2, rowdim:=0, coldim:=2, colsperline:=2;
```

The resulting output looks as follows.

```
Transport := data
{ ( Amsterdam , Amsterdam ) : 2.50, ( Amsterdam , Rotterdam ) : 2.50,
  ( Amsterdam , 'Den Haag' ) : 5.00, ( Rotterdam , Amsterdam ) : 2.50,
  ( Rotterdam , Rotterdam ) : 5.00, ( Rotterdam , 'Den Haag' ) : 5.00,
  ( 'Den Haag' , Rotterdam ) : 2.50, ( 'Den Haag' , 'Den Haag' ) : 5.00 } ;
```

- In the following DISPLAY statement the row and column display dimensions do not add up to the dimension of Transport.

```
display Transport where decimals:=2, rowdim:=0, coldim:=1, colsperline:=3;
```

As a result AIMMS considers the indices corresponding to the dimension deficit as outer, and displays Transport by means of three one-dimensional displays, each of the requested dimension.

```
Transport('Amsterdam', j) := data
{ Amsterdam : 2.50, Rotterdam : 2.50, 'Den Haag' : 5.00 } ;

Transport('Rotterdam', j) := data
{ Amsterdam : 2.50, Rotterdam : 5.00, 'Den Haag' : 5.00 } ;

Transport('Den Haag', j) := data
{ Rotterdam : 2.50, 'Den Haag' : 5.00 } ;
```

- The following DISPLAY statement illustrates how a composite table can be obtained for identifiers defined over the same domain, with a different number of decimals for each identifier.

```
display { Supply decimals := 2, Demand decimals := 3 };
```

Execution of this statement results in the creation of the following one-dimensional composite table.

```
Composite table:
  i      Supply   Demand
! -----
Amsterdam  10.00   5.000
Rotterdam  12.50   10.000
'Den Haag'  7.50   15.000
;
```

## 29.4 Structuring a page in page mode

In addition to the continuous stream mode of operation of the PUT statement discussed in the previous section, AIMMS also provides a page-based file format. AIMMS divides a page-based file into pages of a specified length, each consisting of a header, a body, and a footer. Figure 29.1 gives an overview of a page in a page-based report.

*Page-based files*

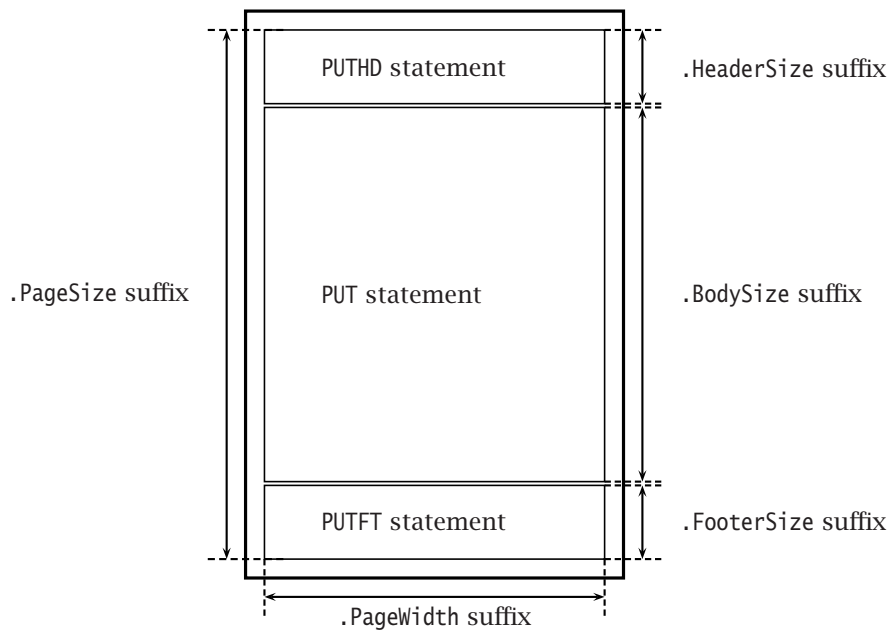


Figure 29.1: Overview of a page in a page based report

You can switch between page and stream by setting the `.PageMode` suffix of a file identifier to 'on' or 'off' (the elements of the predefined set `OnOff`), respectively, as in the statement `ResultFile.PageMode := 'on'`. The value of the `.PageMode` suffix is 'off' by default. When switching to another mode AIMMS will begin with a new page or close the last page.

*Switching to page mode*

The default page size is 60 lines. You can overwrite this default by setting the `.PageSize` suffix of the file identifier to another positive integer value. For instance, `ResultFile.PageSize := 10` will give short pages with only ten lines per page. The default page width is 132 columns. You can change this default by setting the `.PageWidth` suffix of the file identifier.

*Page size and width*

The header and footer of a document can be specified by using the `PUTHD` and `PUTFT` statements. They are equivalent to the `PUT` statement but write in the header and footer area instead of in the page body. The size of the header and footer is not preset, but is determined by the contents of the `PUTHD` and `PUTFT` statements. The header and footer keep their contents from page to page.

*Headers and footers*

There are no specific attributes for either the top, bottom, left or right margins of a page. You essentially control these margins by either resizing the header or footer of a page, or by positioning the `PUT` items in a starting column of your choice using the `@` operator of the `PUT` statement.

*Margins*

Table 29.5 summarizes the file attributes for structuring pages. With the exception of the page body size (read only) you can modify their defaults by using assignment statements.

*Page structure*

Suffix	Description	Default
<code>.PageMode</code>	Mode	'off'
<code>.PageSize</code>	Page size	60
<code>.PageWidth</code>	Page width	132
<code>.PageNumber</code>	Current page number	1
<code>.BodyCurrentColumn</code>	Body current column	-
<code>.BodyCurrentRow</code>	Body current row	-
<code>.BodySize</code>	Body size	-
<code>.HeaderCurrentColumn</code>	Header current column	-
<code>.HeaderCurrentRow</code>	Header current row	-
<code>.HeaderSize</code>	Header size	-
<code>.FooterCurrentColumn</code>	Footer current column	-
<code>.FooterCurrentRow</code>	Footer current row	-
<code>.FooterSize</code>	Footer size	-

Table 29.5: Page structure attributes

The positioning operators @, #, and / explained in Section 29.2 are also applicable in page mode. However, AIMMS offers you additional file attributes for positioning items in a page-based file.

*Positioning in page mode*

Whenever you PUT an item into a header, footer, or page body, there is a current row and a current column. AIMMS keeps track of which row and column are current through the suffices .BodyCurrentRow and .BodyCurrentColumn of the FILE identifier. You can either read or overwrite these values using assignment statements. Similar suffices also exist for the header and the footer area.

*Current row and column*

After having specified the header, footer, or page body, you may want to change their size at some stage during the process of writing pages. By specifying the .BodySize, .HeaderSize and .FooterSize suffices you can modify the size (or empty) the page body, the header, or the footer. The value of the .BodySize suffix can be at most the value of the .PageSize suffix minus the value of the .HeaderSize and .FooterSize suffices.

*Modifying size of page sections*

Whenever you write the contents of the .PageNumber suffix of a FILE identifier in its header or the footer area, AIMMS will replace it with the current page number whenever it prints a page of a page based report. By default, the first page will be numbered 1, but you can override this by assigning another value to the .PageNumber suffix.

*Printing the page number*

---

## 29.5 The standard output listing

AIMMS produces a standard output listing file for each run of a procedure and each solution of a mathematical program. The name of this listing is the base name of the model file with the extension “.lis”. The listing is optionally generated during the first execution in a session and, depending on the option settings, can also be generated during subsequent execution—after updates to parameters and variables.

*The .lis extension*

A standard output listing file can contain one or more of the following items:

*Contents of a listing file*

- a *source listing*—the source code as compiled,
- a *constraint listing*—a printout of the generated individual constraints of a mathematical program,
- a *solution listing*—the solution values for its variables and constraints,
- a *solver status file*—a progress report on the solution process, and
- any *undirected ASCII output* produced from PUT or DISPLAY statements within your model.

By default, the standard output listing will be empty unless you set options that activate AIMMS to print one or more of the items in the list above. By not setting options, you avoid the creation of lengthy output files every time you run a model. In addition, you speed up the solution process by avoiding unnecessary overhead.

*Output  
activated via  
options*

Whenever you want to inspect the model at the individual constraint level, or want to examine the performance of the solver in some detail, then a listing file is your ultimate source of information. The required options for the production of this file can be set from within the model text or from within the graphical interface of AIMMS. They are retained with your project. For more specific information on each of the available options, please consult the AIMMS help file.

*Examining the  
solution process*

After setting the option `constraint_listing` to 1, AIMMS produces the following standard listing for the transport model used throughout this manual. The model uses a small example data set containing just a few Dutch cities. A detailed explanation of the listing format is given at the end.

*Example*

This is the first constraint listing of `TransportModel`.

*The constraint  
listing*

```

---- MeetDemand

MeetDemand('Amsterdam') .. [ 1 | 1 | after ]

    + 1 * Transport('Amsterdam' , 'Amsterdam' ) + 1 * Transport('Rotterdam' , 'Amsterdam' )
    >= 5.88 ; (lhs=5.88)

MeetDemand('Rotterdam') .. [ 1 | 2 | after ]

    + 1 * Transport('Amsterdam' , 'Rotterdam' ) + 1 * Transport('Rotterdam' , 'Rotterdam' )
    >= 12.4 ; (lhs=12.4)

MeetDemand('Den Haag') .. [ 1 | 3 | after ]

    + 1 * Transport('Amsterdam' , 'Den Haag' ) + 1 * Transport('Rotterdam' , 'Den Haag' )
    >= 12.8 ; (lhs=12.8)

---- MeetSupply

MeetSupply('Amsterdam') .. [ 1 | 4 | after ]

    + 1 * Transport('Amsterdam' , 'Amsterdam' ) + 1 * Transport('Amsterdam' , 'Rotterdam' )
    + 1 * Transport('Amsterdam' , 'Den Haag' )
    <= 16 ; (lhs=15.1)

MeetSupply('Rotterdam') .. [ 1 | 5 | after ]

    + 1 * Transport('Rotterdam' , 'Amsterdam' ) + 1 * Transport('Rotterdam' , 'Rotterdam' )
    + 1 * Transport('Rotterdam' , 'Den Haag' )
    <= 16 ; (lhs=16)

---- TotalCost_definition

```

```
TotalCost_definition .. [ 1 | 6 | after ]
+ 1 * TotalCost
- 3.34 * Transport('Amsterdam', 'Amsterdam') - 11.7 * Transport('Amsterdam', 'Rotterdam')
- 13 * Transport('Amsterdam', 'Den Haag' ) - 9 * Transport('Rotterdam', 'Amsterdam')
- 2 * Transport('Rotterdam', 'Rotterdam') - 3 * Transport('Rotterdam', 'Den Haag' )
= 0 ; (lhs=0)
```

The above listing contains all the individual constraints generated by AIMMS on the basis of the model formulation and the particular data set loaded at the time of the SOLVE statement. Each individual constraint name is followed by three entries within square brackets.

*Explanation*

- The first entry represents the number of times that a SOLVE statement has been executed.
- The second entry is a consecutive number assigned to each individual constraint being printed.
- The third entry indicates when the constraint listing is generated (either “before” or “after” a SOLVE statement has been executed).

Bracketed at the end of each constraint is the value of the left-hand side. You can compare this with the right-hand side to evaluate the status of the constraint. By setting the option `constraint_variable_values` to 1 you get a more extensive listing that also includes the values and bounds of the variables that are included in each constraint.

The following solution listing results from setting the option `solution_listing` to 1. Note that the listing includes values for each of the suffices attached to variables and constraints. The `status` column for variables indicates whether or not the variable is basic, frozen, at bound, or bound exceeded. Similarly, the status column for constraints indicates the same basis and bound information as for variables.

*Solution listing*

This is the first solution report of `TransportModel` after a solve.

*Example  
solution listing*

```
The 1 scalar variable:
Name      Lower  level  Upper  ReducedCost  Status
-----
TotalCost -inf   172.079  inf           0  Basic

The variable "Transport(i,j)" contains the following 6 columns:
      i      j  Lower  level  Upper  ReducedCost  Status
-----
Amsterdam Amsterdam  0  5.880  inf     0.000  Basic
Amsterdam Rotterdam  0  9.200  inf     0.000  Basic
Amsterdam 'Den Haag'  0  0.000  inf     0.300  At bound
Rotterdam Amsterdam  0  0.000  inf    15.360  At bound
Rotterdam Rotterdam  0  3.200  inf     0.000  Basic
Rotterdam 'Den Haag'  0 12.800  inf     0.000  Basic
```

The 1 scalar constraint:

Name	ShadowPrice	Status
-----	-----	-----
TotalCost_definition	0	

The constraint "MeetDemand(j)" contains the following 3 rows:

j	Lower	level	Upper	ShadowPrice	Status
-----	-----	-----	-----	-----	-----
Amsterdam	5.880	5.880	inf	3.340	At bound
Rotterdam	12.400	12.400	inf	11.700	At bound
'Den Haag'	12.800	12.800	inf	12.700	At bound

The constraint "MeetSupply(i)" contains the following 2 rows:

i	Lower	level	Upper	ShadowPrice	Status
-----	-----	-----	-----	-----	-----
Amsterdam	-inf	15.080	16	0.000	Basic
Rotterdam	-inf	16.000	16	-9.700	At bound