
AIMMS Language Reference - Format of ASCII Data Files

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 26

Format of ASCII Data Files

Data provided in ASCII data files can be provided in scalar, list or tabular format. While the scalar and list formats can also be used in ordinary expressions, the tabular formats are only allowed for data initialization. This chapter discusses the general format of ASCII data files with special emphasis on the two possible tabular formats. Data provided in ASCII files can only be read through the use of the READ statement which is discussed in Chapter [24.2](#).

This chapter

26.1 ASCII data files

ASCII data files must contain one or a sequence of identifier assignments with a *constant* right-hand side. All assignments must be terminated by a semi-colon. The following constant formats can be assigned:

Allowed ASCII formats

- assignment of *scalar constants*,
- assignment of *constant enumerated set expressions*,
- assignment of *constant enumerated list expressions*,
- assignment of *constant tabular expressions*, and
- assignment via *composite tables*.

The first three formats can also be used in ordinary expressions, and have been discussed in Chapters [5](#) and [6](#). The tabular and composite table formats are mostly placed in external data files, and will be discussed in this chapter.

When you use the WRITE statement to write the contents of some or all identifiers in your model to an ASCII file, AIMMS will select the appropriate format and write the resulting output accordingly. If you want actual control over the way identifiers are printed, you should use the PUT or DISPLAY statements (see also Sections [29.2](#) and [29.3](#)).

AIMMS generated output

The ASCII formats allowed in AIMMS are straightforward, and it is not difficult to generate these formats either manually or through an external program. As a result, ASCII files form an ideal input medium when you quickly need to create a small data set to test your AIMMS application, or when data is obtained from a program to which a direct link cannot be made.

Easily generated

26.2 Tabular expressions

For multidimensional quantities the table format often provides the most natural structure for data entry because elements are repeated less often. Tables can be used in ASCII data files and in the INITIAL DATA attribute inside the declaration of an identifier.

Tables for initialization

A table is a two-dimensional view of a multidimensional quantity. The index tuple of the quantity is split into two parts: row identifiers and column identifiers. Indices may not be permuted.

Two-dimensional views

The following example illustrates a simple example of the table format.

Example

```
Distance(i,j) := DATA TABLE
                Rotterdam Antwerp Berlin Paris
!              -----
  Amsterdam      85      170      660      510
  Rotterdam              100      700      440
  Antwerp                      725      340
  Berlin                                1050
;
```

The first line of a table (after the keyword DATA TABLE) contains the column identifiers. Each subsequent line contains a row identifier followed by the table entries.

Row and column identifiers may be set elements, tuples of elements, or tuples containing element ranges. As a result, multidimensional identifiers can still be captured within the two-dimensional framework of a table.

Multi-dimensional entries

Column identifiers must be separated by at least one space. AIMMS keeps track of the column width by maintaining the first and last position used by each column identifier. Any entry must intersect only one column and is understood to be part of that column. AIMMS will reject any entry that intersects two columns, or falls between them.

Proper spacing

Even though the table format is a convenient way to enter data, the number of columns is always restricted by the width of a line. However, by placing a + on a new line you can continue a table by repeating the table format. Row identifiers and column identifiers can be repeated in each block separated by the + sign, but must be unique within a block.

Continuation of tables with +

The following table illustrates a valid example of table continuation, equivalent with the previous example. *Example*

```

Distance(i,j) := DATA TABLE
                Rotterdam  Antwerp
!
  Amsterdam     85        170
  Rotterdam     100
+
                Berlin    Paris
!
  Amsterdam     660     510
  Rotterdam     700     440
  Antwerp       725     340
  Berlin        1050
;
    
```

Tables can be used for the initialization of both parameters and sets. When used for parameter initialization, table entries are either blank or contain explicit numbers, quoted or unquoted set elements and quoted strings. Entries in tables used for set initialization are either blank or contain a "*" denoting membership. *Data and membership tables*

The detailed syntax of a table constant is given by the following diagram, where the symbol "\n" stands for the newline character. *Syntax*

table :

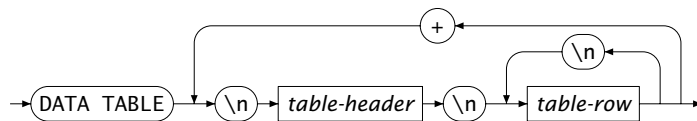


table-header :

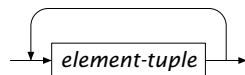
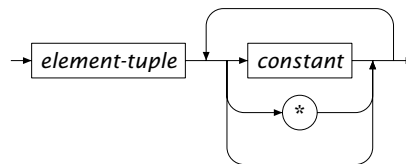


table-row :



26.3 Composite tables

A composite table is a bulk form of initialization, and is similar in structure to a table in a database. Using a composite table you can initialize simple sets, compound sets, parameters, and variables in a single statement. Composite tables always form a single block, and can only be used in ASCII data files. *Multiple identifiers*

The first line of a composite table contains column identifiers that define the index columns and the quantity columns. The subsequent lines contain data entries. Like in a tabular expression, entries in a composite table may be either blank or contain explicit numbers, quoted or unquoted set elements and quoted strings, depending on the type of the identifier associated with a column. Blank entries in the quantity columns are treated as “no assignment.”, while blank entries in the index columns are not allowed. All data entries must lie directly below their corresponding column identifier as in regular tables.

Format

The full index space is declared in the first group of column identifiers, and is comparable to the *primary key* in a database table. The remaining column identifiers declare various quantities that must share the identical index space. Note that, unlike in tabular expressions, index columns in a data entry row of a composite table cannot refer to tuples or ranges of elements, but only to single set elements.

Indices must come first

The following statement illustrates a valid example of a composite table. It initializes the compound set *Routes*, as well as the parameters *Distance* and *TransportCost*, all of which are defined over the index space (i, j).

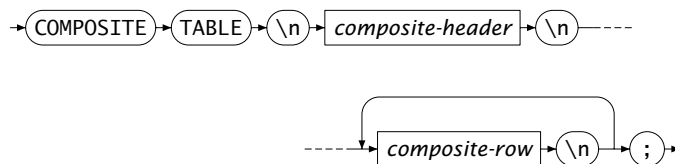
Example

```
COMPOSITE TABLE
!  i          j          Routes  Distance  TransportCost
!  -----  -----  -----  -----  -----
  Amsterdam Rotterdam   *         85         1.00
  Amsterdam  Antwerp    *        170         2.50
  Amsterdam   Berlin   *        660        10.00
  Amsterdam   Paris    *        510         8.25
  Rotterdam  Antwerp   *        100         1.20
  Rotterdam   Berlin   *        700        10.00
  Rotterdam   Paris    *        440         7.50
  Antwerp     Berlin   *        725        11.00
  Antwerp     Paris    *        340         5.00
  Berlin      Paris    *       1050        17.50
;
```

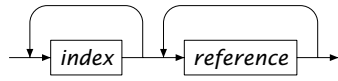
The detailed syntax of the composite table is given by the following diagram, where the symbol “\n” stands for the newline character.

Syntax

composite-table :



composite-header :



composite-row :

