AIMMS Language Reference - Language Preliminaries

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

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

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

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

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

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

Chapter 2

Language Preliminaries

This reference guide

This language reference describes the syntax and semantics of the AIMMS language. It is recommended that you read the chapters in sequence, but this is not essential. Both the contents and index are useful for locating the specifics of any topic. Illustrative examples throughout the text will give you a quick understanding of each subject.

2.1 Managing your model

Models in AIMMS

AIMMS is a language for the specification and implementation of multidimensional modeling applications. An AIMMS *model* consists of

- a *declarative part* which specifies all sets and multidimensional identifiers defined over these sets, together with the fixed functional relationships defined over these identifiers,
- an *algorithmic part* consisting of one or more procedures which describes the sequence of statements that transform the input data of a model into the output data, and
- a *utility part* consisting of additional identifier declarations and procedures to support a graphical end-user interface for your application.

Optimization included ...

The declarative part of a model in AIMMS may include the specification of optimization problems containing simultaneous systems of equations. In the algorithmic part you can call a special SOLVE statement to translate such optimization problems to a format suitable for a linear or nonlinear solver.

... but not necessary

Although optimization modeling will be an important part of most AIMMS applications, AIMMS is also a convenient tool for other types of applications.

- The purely symbolic representation of set and parameter definitions with their automatic dependency structure provides spreadsheet-like functionality but with the benefit of much greater maintainability.
- Because of its simple data structures and power of expression, AIMMS lends itself for use as a rapid prototyping language.

Interfacing with the GUI

Although it is possible to create a simple end-user interface showing your model's data in the form of tables and graphs, a much more advanced user interface is possible by exploiting the capabilities of the AIMMS interface builder. Mostly, this involves the introduction of various additional sets and parameters in your model, as well as the implementation of additional procedures to perform special interface-related tasks.

The model tree

Modeling in AIMMS is centered around a graphical tool called the *model explorer*. In the model explorer the contents and structure of your model is presented in a tree-like fashion, which is also referred to as the *model tree*. The model tree can contain various types of nodes, each with their own use. They are:

- *structuring* sections, which you can use to partition the declarations and procedures that are part of your model into logical groups,
- *declaration* sections which contain the *declarations* of the global identifiers (like sets, parameters and variables) in your model, and
- *procedures* and *functions* which contain the statements that describe the algorithmic part of your application.

Creating new models

When you start a new model AIMMS will automatically create a skeleton model tree which is suitable for small applications. The skeleton contains the following nodes:

- a single *declaration section* where you can store the declarations used in your model,
- the predefined procedure `MainInitialization` which is called directly after compiling your model and can be used to initialize your model,
- the predefined procedure `MainExecution` where you can put all the statements necessary to execute the algorithmic part of your application, and
- the predefined procedure `MainTermination` which is called just prior to leaving AIMMS.

Changing the skeleton

Whenever the number of declarations in your model grows too large to be easily managed within a single declaration section, or when you want to divide the execution associated with your application into several procedures, you are free to change the skeleton model tree created by AIMMS. You can group particular declarations into separate declaration sections with a meaningful name, and introduce new procedures and functions.

Structuring your model

When you feel that particular groups of declarations, procedures and functions belong together in a logical manner, you are encouraged to create a new structuring section with a descriptive name within the model tree, and store the associated model components underneath it. When your application grows in size, a clear hierarchical structure of all the information stored will help tremendously to find your way within your application easily and quickly.

Storage on disk

The contents of a model is stored in one or more text files with the ".ams" (AIMMS model source) extension. By default the entire model is stored in a single file, but for each structural section you can indicate that you want to store the subtree underneath it in a separate source file. This is especially useful when particular parts of your application are shared with other AIMMS applications, or when there are multiple developers, each responsible for a particular part of the model.

Character encoding used in text files

A text is a sequence of characters. A text file contains such a text whereby the characters are encoded into numbers. The mapping between these characters in a text and these numbers in a file is called an encoding. The historically prevailing encoding is ASCII which defines the encoding for some control characters, the English alphabet, digits, and frequently used punctuation characters for the values 1 .. 127. However, as characters are stored in bytes, the values 128 .. 255 are free and these are used at different locales for different purposes. These locale specific extensions of ASCII are also called code pages. As a consequence, the characters displayed of an ASCII file containing some of the numbers 128 .. 255, depend on the active code page selected. The problem here is that the contents of ASCII files were ambiguous when the code page to be used was not known (see also en.wikipedia.org/wiki/Codepage). In order to circumvent this problem, the Unicode consortium enumerated all characters into more than 64 thousand so-called code points. The first 127 Unicode code points match the first 127 characters of ASCII. These Unicode code points can be encoded, again, in various ways in a file. To emphasize that a particular number is a Unicode point, such a number is often denoted as U+xxxx whereby xxxx is a hexadecimal number. An example Unicode encoding is UTF8, which stores the first 127 code points in a single byte. This makes a UTF8 file closely resemble ASCII when no values above 127 are used. To identify the Unicode encoding used in a file, a so-called Byte Order Mark (BOM) can be used in the first few bytes of that file. See also www.unicode.org and en.wikipedia.org/wiki/Byte_order_mark.

UTF8 encoding preferred

UTF8 is a popular encoding; it resembles ASCII for the first 127 code points and can be used by applications deployed at different locales to unambiguously exchange data. Most modern text editors, including the one in AIMMS, are able to handle UTF8 text files. We recommend UTF8 encoding for AIMMS files, especially when AIMMS is used inside international organizations. AIMMS system files, including the .ams model file and the .aimms project file, use the UTF8 encoding.

Version control

After each editing session AIMMS will only save the last version of your model files, and will not retain a backup of the previous version of your model files. You are therefore strongly encouraged to use a version control system to keep a history of the changes you made to your model.

Other model files

In addition to the model files AIMMS stores a number of other files with each model. They are:

- a *project file* containing the pages of the graphical (end-)user interface that you have created for your application and all other relevant information such as project options, user menus, fonts, etc., and
- a *data tree file* containing all the stored datasets and cases associated with your application.

2.2 Identifier declarations

Identifier types

Identifiers are the unique names through which you can refer to entities in your model. The most common identifier types in AIMMS are:

- *set*—used for indexing parameters and variables,
- *parameter*—for (multidimensional) data storage,
- *variable* and *arc*—entities of constraints that must be determined,
- *constraint* and *node*—relationships between variables or arcs, usually in the form of (in)equalities,
- *mathematical program*—an objective and a collection of constraints, and
- *procedure* and *function*—code segments to initiate execution.

Declaration forms

The declarations of all identifiers, procedures and functions within an AIMMS application can be provided by means of a uniform attribute notation. For every node within the model tree you can view and change the value of these attributes through a graphical declaration form. This form will show all the attributes that are associated with a particular identifier type, along with their values for the identifier at hand.

Notation used in this manual

In this manual we have chosen to use a textual style representation of all model declarations, which closely resembles the graphical representation in the model tree. In view of the large number of declarations in this manual, we found that a purely graphical presentation in the text was visually distracting. In contrast, the adopted textual representation is succinct and integrates well with the surrounding text.

The `Text` and `Comment` attributes

With every declaration in a model you can associate a `Text` and a `Comment` attribute. The `Comment` attribute is aimed at the modeler, and can be used to describe the contents of a particular node in the model tree, or make remarks that are relevant for later reference. The `Text` attribute is intended for use in the graphical user interface and reporting. It can contain a single line description of the identifier at hand. Many objects in the AIMMS user interface allow you to display this text along with the identifier value(s).

Predefined identifiers

Not only does an AIMMS model consist of sets, parameters and variables that have been defined by you, and thus are specific for your application, AIMMS also provides a number of predefined system identifiers. These identifiers characterize either

- a set of *all* objects with a particular property, for instance the set of `AllIdentifiers` or the set of `AllCases`, or
- the *current* value of a particular modeling aspect, for instance the parameter `CurrentCase` or the parameter `CurrentPageNumber`.

In most cases these identifiers are read-only, and get their value based on the declarations and settings of your model.

Section identifiers

The structuring sections in your model tree are also considered as AIMMS identifiers. The blanks in a section description are replaced by underscores to form a legal AIMMS identifier name. The identifier thus formed is a subset of `AllIdentifiers`. This subset contains all the model identifiers that have been declared underneath the associated node. You can conveniently use such sets in, for instance, the `EMPTY` statement to clean a entire group of identifiers in a single statement, or to construct your own subsets of `AllIdentifiers` using the set operations available in AIMMS.

2.3 Lexical conventions

Lexical conventions

Before treating the more intricate features of the AIMMS language, we have to discuss its lexical conventions. That is, we have to define the basic building blocks of the AIMMS language. Each one is described in a separate paragraph.

Characters

The set of characters recognized by AIMMS consists of the set of all printable characters, together with the tab character. Tab characters are not expanded by AIMMS. The character immediately following a tab character is positioned at column 9, 17, 25, 33, etc. All other unprintable or control characters are illegal. The presence of an illegal character causes a compiler error.

Numbers

Numerical values are entered in a style similar to that in other computer languages. For data storage AIMMS supports the integer data type as well as the real data type (floating point numbers). During execution, however, AIMMS will always use a double precision floating point representation.

Scientific notation

Following standard practice, the letter `e` denotes the scientific notation allowing convenient representation of very large or small numbers. The number following the `e` can only be a positive or negative integer. Two examples of the

use of scientific notation are given by

$$1.2e5 = 1.2 \times 10^5 = 120,000$$
$$2.72e-4 = 2.72 \times 10^{-4} = 0.000272$$

Special numbers

In addition to the ordinary real numbers, AIMMS allows the special symbols INF, -INF, UNDF, NA, and ZERO as numbers. The precise meaning and use of these symbols is described later in Section 6.1.1.

No blanks within numbers

Blanks cannot be used inside a number since AIMMS treats a blank as a separator. Thus, valid examples of expressions recognized as numbers by AIMMS are

```
0         0.0       .0        0.        +1        1.
0.5       .5        +0.5      +.5       -0.3      -.3
2e10      2e+10     2.e10     0.3e-5    .3e-5     -.3e-05
INF       -INF      NA        ZERO
```

Machine precision

The range of values allowed by AIMMS and the number of significant digits is machine-dependent. AIMMS takes advantage of the accuracy of your machine. This may cause different results when a single model is run on two different machines. Expressions that cause arithmetic under- or overflow evaluate to the symbols ZERO and INF, respectively. Functions and operators requiring integer arguments also accept real numbers that lie within a machine-dependent tolerance of an integer.

Identifiers

Identifiers are the unique names given to sets, indices, parameters, variables, etc. Identifiers can be any sequence of the letters a–z, the digits 0–9 and the underscore _. They must start with either a letter or an underscore. The length of an identifier is limited to 255 characters. Examples of legal identifiers include:

```
a       b78     _c_
A_very_long_but_legal_identifier_containing_underscores
```

The following are not identifiers:

```
39      39id    A-ident      a&b
```

Namespaces

In principle, AIMMS operates with a global namespace for all declared identifiers. By introducing modules into your model (see also Section 35.4), you can introduce multiple namespaces, which can be convenient when a particular model section contains logic that can be shared by multiple AIMMS models. Procedures and functions automatically create a separate namespace, allowing for local identifiers with the same name as global identifiers in your model.

You can use the *namespace resolution* operator :: to refer to an identifier in a particular namespace (see also Section 35.4).

Redeclaring AIMMS *keywords*

In general, you are not allowed to redeclare AIMMS keywords as identifiers, unless a keyword refers to a non-essential feature of the language. Whenever you try to redeclare an existing AIMMS keyword, AIMMS will produce a compiler error when a keyword cannot be redeclared, or will give you a one-time option to redeclare a non-essential keyword as a model identifier. In the latter case, the non-essential feature will be permanently unavailable within your project.

Case sensitivity

The AIMMS language is *not* case sensitive. This means that upper and lower case letters can be mixed freely in identifier names but are treated identically by AIMMS. However, AIMMS is *case aware*, in the sense that it will try to preserve or restore the original case wherever possible.

Identifiers with suffices

Some AIMMS data types have additional data associated with them. You have access to this extra data through the identifier name plus a suffix, where the suffix is separated from the identifier by a dot. Examples of suffices are:

```
c.Derivative      Transport.ReducedCost      OutputFile.PageSize
```

You can use a suffix expression associated with a particular identifier as if it were an identifier itself.

Case referencing

In addition, AIMMS also uses the dot notation to refer to the data associated from another case file. An example is given below.

```
CaseDifference(i,j) := Transport(i,j) - ReferenceCase.Transport(i,j);
```

In this example the values of a variable `Transport(i,j)` currently in memory are compared to the values in a particular reference case on disk, identified by the case identifier `ReferenceCase`. You will find more information about case references in Section 6.1.3.

Value types

Any constant or parameter in AIMMS must assume one of the following value types:

- number (either integer or floating point),
- string,
- set element, or
- unit expression.

All value types except unit expressions are discussed below. Unit expressions are explained in Section 32.6.

Strings

Constants of string type in AIMMS are delimited by a double quote character “"”. To include the double quote character itself in a string, it should be escaped by the backslash character “\” (see also Section 5.3.2). Strings can be used as constants in expressions, as arguments of procedures and functions, and in the initialization of string-valued parameters. The size of strings is limited to 64 Kb.

Sets and set elements

A set is a group of like elements. Sets can be *simple* (one-dimensional) or *compound* (multidimensional). The elements of a simple set are represented either by

- an integer number,
- a single-quoted string of a length less than 255 characters, or
- an unquoted string subject to conditions explained below.

The elements of a compound set are represented by tuples of such integers or strings.

Integer elements

The elements of an integer set can be used in expressions as if they were integer numbers. Reversely, you can use integer-valued numerical expressions to indicate an element of an integer set.

Quoted string elements

The characters allowed in a quoted string elements are the set printable characters except for tab and newline.

Unquoted string elements

For your convenience, the elements of a string set need not be delimited by a single quote when all of the following conditions are met:

- the string used as a set element consists only of letters, digits, underscores and the sign characters “+” and “-,”
- the set element is not a reserved word or token, and
- the set element is used inside a constant expression such as a constant *enumerated set* or *list* expression (see also Sections 5.1.1 and 6.1.2), or inside *table* or a *composite table* used for the initialization of parameters and variables (see also Sections 28.2 and 28.3).

String-valued set elements that are referenced explicitly under any circumstance other than the ones mentioned above, must be quoted unconditionally. To include a single quote character in a set element, it should be preceded by the backslash character “\”.

Examples of set elements

The following set elements are examples of set elements that can be used without quotation marks under the conditions mentioned above:

```
label1          1998            1997-12         1997_12
january         january-1998    h2so4           04-Mar-47
```

The following character strings are also valid as set elements, but must be quoted in all cases.

```
'An element containing spaces'
'label with nested quotes: "a*b"'
```

String elements do not have a value

Contrary to integer set elements, string elements do *not* have an associated number value. Thus, the string element `'1993'` does not have the value 1993. If you use string elements to represent numbers, you can use the `Val` function to obtain the associated value. Thus, `Val('1993')` represents the number 1993.

Delimiters

The following delimiters are used by AIMMS:

- a space " " separates keywords, identifiers and numbers,
- a pair of single quotes "'" or double quotes """ delimits set elements and strings, respectively,
- a semicolon ";" separates statements,
- braces "{" and "}" denote the beginning and end of sets and lists,
- a comma "," separates elements of sets and lists,
- parentheses "(" and ")" delimit expressions, tuples of indices and set elements, as well as argument lists of functions and references, and
- square brackets "[" and "]" are used to delimit unit expressions as well as numeric and element ranges. They can also be used as parentheses in expressions and argument lists of functions and references, and for grouping elements in components of an element tuple (see also Section 5.1.1).

In most other expressions parentheses and square brackets can be used interchangeably as long as they match. This feature is useful for making deeply nested expressions more readable.

Limits in AIMMS

The following limits apply within AIMMS.

- the length of a line is limited to 255 characters,
- the number of set elements per set is at most 2^{30},
- the number of indices associated with an identifier is at most 32, and
- the number of running indices used in iterative operations such as `SUM` and `FOR` is at most 16.

2.4 Expressions and statements

Model execution

The creation of an AIMMS model is implemented using two separate but interacting mechanisms. They are:

- automatic updating of the functional relationships specified through expressions in the `Definition` attributes of sets and parameters in your model, and

- manual execution of the statements that constitute the Body attribute of the procedures and functions defined in your application.

The precise manner in which these components are executed, and the way they interact, is discussed in detail in Chapters 7 and 8. This section discusses the general structure of an AIMMS model as well as the requirements for the Definition and Body attributes.

Line length and empty lines

The length of any particular line in the Definition attribute of an identifier or the Body attribute of a procedure or function is limited to 255 characters. Although this full line length may be convenient for data instantiation in the form of large tables, it is recommended that you do not exceed a line length of 80 characters in these attributes in order to preserve maximum readability. Empty lines can be inserted anywhere for easier reading.

Commenting

Expressions and statements in the Body attribute of a procedure or function can be interspersed with comments that are ignored during compilation. AIMMS supports two kinds of comments:

- the tokens "/*" and "*/" for a block comment, and
- the exclamation mark "!" for a one line comment.

Each block comment starts with a "/*" token, and runs up to the matching "*/" token, and cannot be nested. It is a useful method for entering pieces of explanatory text, as well as for temporarily commenting out one or more execution statements. A one-line comment starts anywhere on a line with an exclamation mark "!", and runs up to the end of that line.

Expressions

The value of a Definition attribute must be a valid expression of the appropriate type. An expression in AIMMS can result in either

- a set,
- a set element,
- a string,
- a numerical value,
- a logical value, or
- a unit expression.

Set, element and string expressions are discussed in full detail in Chapter 5, numerical and logical expressions in Chapter 6, while unit expressions are discussed in Chapter 32.

Statements

AIMMS statements in the body of procedures and functions constitute the algorithmic part of a modeling application. All statements are terminated by a semicolon. You may enter multiple statements on a single line, or a single statement over several lines.

Execution statements

To specify the algorithmic part of your modeling application, the following statements can be used:

- assignments—to compute a new value for a data item,
- the SOLVE statement—to solve a mathematical program for the values of its variables,
- flow control statements like IF-THEN-ELSE, FOR, WHILE, REPEAT, SWITCH, and HALT—to manage the flow of execution,
- the OPTION and Property statements—to set identifier properties and options dealing with execution, output, progress, and solvers,
- the data control statements EMPTY, CLEANUP, READ, WRITE, DISPLAY, and PUT—to manage the contents of internal and external data.
- procedure calls—to execute the statements contained in a procedure.

The precise syntax of these execution statements is discussed in Chapters 8 and further.

2.5 Data initialization

Initialization syntax

The initialization of sets, parameters, and variables in an AIMMS application can be done in several ways:

- through the *InitialData attribute* of sets, and parameters,
- by reading in data from an *text file* in AIMMS data format,
- by reading in data from a previous AIMMS session stored in a binary *case file*,
- by reading in the data from an external *ODBC-compliant database*, or
- by initializing an identifier through algebraic assignment statements.

Order of initialization

When starting up your AIMMS application, AIMMS will initialize your model identifiers in the following order.

- Following compilation each identifier is initialized with the contents of its InitialData attribute.
- Subsequently, AIMMS will execute the predefined procedure MainInitialization. You can use it to specify READ statements to read in data from text files, case files or databases. In addition, it can contain any other algebraic statement necessary to initialize one or more identifiers in your model. Of course, you can also leave this procedure empty if you so desire.

The full details of model initialization are discussed in Chapter 25.

Entering the InitialData attribute

The InitialData attribute of an identifier can contain any *constant* set-valued, set element-valued, string-valued, or numerical expression. In order to construct such expressions (consisting of mostly tables and lists), AIMMS offers so-called *data pages* which can be created on demand. These pages help you enter the data in a convenient and graphical manner.