
AIMMS User's Guide - User Interface Language Components

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.

Part V

Miscellaneous

Chapter 19

User Interface Language Components

Most of the functionality in the AIMMS graphical user interface that is relevant to end-users of your modeling application can be accessed directly from within the AIMMS modeling language. This chapter discusses the functions and identifiers in AIMMS that you can use within your model

This chapter

- to influence the appearance and behavior of data shown in your end-user interface, or
- to provide (or re-define) direct interaction with the end-user interface through dialog boxes, menus and buttons.

Rather than providing a complete reference of all these functions, this chapter provides you with a global overview of the functions available per functional category. A complete function reference is made available as part of the AIMMS documentation in electronic form.

19.1 Updatability of identifiers

In many applications you, as a modeler, might need to have dynamic control over the updatability of identifiers in the graphical end-user interface of your model. AIMMS provides several ways to accomplish this.

Dynamic control required

A typical example of dynamically changing inputs and outputs is when your model is naturally divided into multiple decision phases. Think of a planning application where one phase is the preparation of input, the next phase is making an initial plan, and the final phase is making adjustments to the initial plan. In such a three-layered application, the computed output of the initial plan becomes the updatable input of the adjustment phase.

Multiple phases in your application

To change the updatability status of an identifier in the graphical interface you have two options.

Indicating input and output status

- You can indicate in the object **Properties** dialog box whether all or selected values of a particular identifier in the object are updatable or read-only.
- With the set `CurrentInputs` you can change the global updatability status of an identifier. That is, AIMMS will never allow updates to identifiers

that are not in the set `CurrentInputs`, regardless of your choice in the properties form of a graphical object.

The set `CurrentInputs` (which is a subset of the predefined set `AllUpdatableIdentifiers`) ultimately determines whether a certain identifier can be treated as an input identifier for objects in an end-user interface. You can change the contents of the set `CurrentInputs` from within your model. By default, AIMMS initializes it to `AllUpdatableIdentifiers`.

*The set
CurrentInputs*

The set `AllUpdatableIdentifiers` is computed by AIMMS when your model is compiled, and contains the following identifiers:

*The set
AllUpdatable-
Identifiers*

- all *sets* and *parameters* without definitions, and
- all *variables* and *arcs*.

Thus, sets and parameters which have a definition can never be made updatable from within the user interface.

19.2 Setting colors within the model

An important aspect of an end-user interface is the use of color. Color helps to visualize certain properties of the data contained in the interface. As an example, you might want to show in red all those numbers that are negative or exceed a certain threshold.

*Color as
indicator*

AIMMS provides a flexible way to specify colors for individual data elements. The color of data in every graphical object in the graphical interface can be defined through an (indexed) “color” parameter. Inside your model you can make assignments to such color parameters based on any condition.

*Setting colors in
the model*

In AIMMS, all *named* colors are contained in the predefined set `AllColors`. This set contains all colors predefined by AIMMS, as well as the set of logical color names defined by you for the project. Whenever you add a new logical color name to your project through the color dialog box, the contents of the set `AllColors` will be updated automatically.

*The set
AllColors*

Every (indexed) element parameter with the set `AllColors` as its range can be used as a color parameter. You can simply associate the appropriate colors with such a parameter through either its definition or through an assignment statement.

*Color
parameters*

Assume that `ColorOfTransport(i,j)` is a color parameter defining the color of the variable `Transport(i,j)` in an object in the end-user interface. The following assignment to `ColorOfTransport` will cause all elements of `Transport(i,j)` that exceed the threshold `LargeTransportThreshold` to appear in red.

Example

```
ColorOfTransport((i,j) | Transport(i,j) >= LargeTransportThreshold) := 'Red' ;
```

19.2.1 Creating non-persistent user colors

During the start up of an AIMMS project, the set `AllColors` is filled initially with the collection of persistent user colors defined through the **Tools-User Colors** dialog box (see also Section 11.4). Through the functions listed below, you can extend the set `AllColors` programmatically with a collection of non-persistent colors, whose lifespan is limited to a single session of a project.

Non-persistent user colors

- `UserColorAdd(colorname,red,green,blue)`
- `UserColorDelete(colorname)`
- `UserColorModify(colorname,red,green,blue)`
- `UserColorGetRGB(colorname,red,green,blue)`

The argument *colorname* must be a string or an element in the set `AllColors`. The arguments *red*, *green* and *blue* must be scalars between 0 and 255.

You can use the function `UserColorAdd` to add a non-persistent color *colorname* to the set `AllColors`. The RGB-value associated with the newly added user color must be specified through the arguments *red*, *green* and *blue*. The function will fail if the color already exists, either as a persistent or non-persistent color.

Adding non-persistent colors

Through the functions `UserColorDelete` and `UserColorModify` you can delete or modify the RGB-value of an existing non-persistent color. The function will fail if the color does not exist, or if the specified color is a persistent color. Persistent colors can only be modified or deleted through the **Tools- User Colors** dialog box.

Deleting and modifying colors

You can obtain the RGB-values associated with both persistent and non-persistent user colors using the function `UserColorGetRGB`. The function will fail if the specified color does not exist.

Retrieving RGB-values

19.3 Creating histograms

The term histogram typically refers to a picture of a number of observations. The observations are divided over equal-length intervals, and the number of observed values in each interval is counted. Each count is referred to as a

Histogram

frequency, and the corresponding interval is called a frequency interval. The picture of a number of observations is then constructed by drawing, for each frequency interval, the corresponding frequency as a bar. A histogram can thus be viewed as a bar chart of frequencies.

The procedures and functions discussed in this section allow you to create histograms based on a large number of trials in an experiment conducted from within your model. You can set up such an experiment by making use of random data for each trial drawn from one or more of the distributions discussed in the AIMMS Language Reference. The histogram frequencies, created through the functions and procedures discussed in this section, can be displayed graphically using the standard AIMMS bar chart object.

Histogram support

AIMMS provides the following procedure and functions for creating and computing histograms.

Histogram functions and procedures

- HistogramCreate(*histogram-id* [, *integer-histogram*] [, *sample-buffer-size*])
- HistogramDelete(*histogram-id*)
- HistogramSetDomain(*histogram-id*, *intervals* [, *left*, *width*] [, *left-tail*] [, *right-tail*])
- HistogramAddObservation(*histogram-id*, *value*)
- HistogramGetFrequencies(*histogram-id*, *frequency-parameter*)
- HistogramGetBounds(*histogram-id*, *left-bound*, *right-bound*)
- HistogramGetObservationCount(*histogram-id*)
- HistogramGetAverage(*histogram-id*)
- HistogramGetDeviation(*histogram-id*)
- HistogramGetSkewness(*histogram-id*)
- HistogramGetKurtosis(*histogram-id*)

The *histogram-id* argument assumes an integer value. The arguments *frequency-parameter*, *left-bound* and *right-bound* must be one-dimensional parameters (defined over a set of intervals declared in your model). The optional arguments *integer-histogram* (default 0), *left-tail* (default 1) and *right-tail* (default 1) must be either 0 or 1. The optional argument *sample-buffer-size* must be a positive integer, and defaults to 512.

Through the procedures HistogramCreate and HistogramDelete you can create and delete the internal data structures associated with each individual histogram in your experiment. Upon success, the procedure HistogramCreate passes back a unique integer number, the *histogram-id*. This reference is required in the remaining procedures and functions to identify the histogram at hand. The observations corresponding to a histogram can be either continuous or integer-valued. AIMMS assumes continuous observations by default. Through the optional *integer-histogram* argument you can indicate that the observations corresponding to a histogram are integer-valued.

Creating and deleting histograms

For every histogram, AIMMS will allocate a certain amount of memory for storing observations. By default, AIMMS allocates space to store samples of 512 observations at most. Using the optional *sample-buffer-size* argument, you can override the default maximum sample size. As long as the number of observations is still smaller than the sample buffer size, all observations will be stored individually. As soon as the actual number of observations exceeds the sample buffer size, AIMMS will no longer store the individual observations. Instead, all observations are then used to determine the frequencies of frequency intervals. These intervals are determined on the basis of the sample collected so far, unless you have specified interval ranges through the procedure `HistogramSetDomain`.

Sample buffer size

You can use the function `HistogramSetDomain` to define frequency intervals manually. You do so by specifying

Setting the interval domain

- the number of fixed-width *intervals*,
- the lower bound of the *left-most* interval (not including a left-tail interval) together with the (fixed) *width* of intervals to be created (optional),
- whether a *left-tail* interval must be created (optional), and
- whether a *right-tail* interval must be created (optional).

The default for the *left* argument is `-INF`. *Note that the left argument is ignored unless the width argument is strictly greater than 0.* Note that the selection of one or both of the tail intervals causes a corresponding increase in the number of frequency intervals to be created.

Whenever an observed value is smaller than the lower bound of the left-most fixed-width interval, AIMMS will update the frequency count of the left-tail interval. If the left-tail interval is not present, then the observed value is lost and the procedure `HistogramAddObservation` (to be discussed below) will have a return value of 0. Similarly, AIMMS will update the frequency count of the right-tail interval, when an observation lies beyond the right-most fixed-width interval.

Use of tail intervals

Whenever, during the course of an experiment, the number of added observations is still below the sample buffer size, you are allowed to modify the interval ranges. As soon as the number of observations exceeds the sample buffer size, AIMMS will have fixed the settings for the interval ranges, and the function `HistogramSetDomain` will fail. This function will also fail when previous observations cannot be placed in accordance with the specified interval ranges.

Adjusting the interval domain

You can use the procedure `HistogramAddObservation` to add a new observed value to a histogram. Non-integer observations for integer-valued histograms will be rounded to the nearest integer value. The procedure will fail, if the observed value cannot be placed in accordance with the specified interval ranges.

Adding observations

With the procedure `HistogramGetFrequencies`, you can request AIMMS to fill a one-dimensional parameter (slice) in your model with the observed frequencies. The cardinality of the index domain of the frequency parameter must be at least as large as the total number of frequency intervals (including the tail interval(s) if created). The first element of the domain set is associated with the left-tail interval, if created, or else the left-most fixed-width interval.

Obtaining frequencies

If you have provided the number of intervals through the procedure `HistogramSetDomain`, AIMMS will create this number of frequency intervals plus at most two tail intervals. Without a custom-specified number of intervals, AIMMS will create 16 fixed-width intervals plus two tail intervals. If you have not provided interval ranges, AIMMS will determine these on the basis of the collected observations. As long as the sample buffer size of the histogram has not yet been reached, you are still allowed to modify the number of intervals prior to any subsequent call to the procedure `HistogramGetFrequencies`.

Interval determination

Through the procedure `HistogramGetBounds` you can obtain the left and right bound of each frequency interval. The bound parameters must be one-dimensional, and the cardinality of the corresponding domain set must be at least the number of intervals (including possible left- and right-tail intervals). The lower bound of a left-tail interval will be `-INF`, the upper bound of a right-tail interval will be `INF`.

Obtaining interval bounds

The functions `HistogramGetObservationCount`, `HistogramGetAverage`, `HistogramGetDeviation`, `HistogramGetSkewness` and `HistogramGetKurtosis` provide further statistical information about the sample collected so far, such as the total number of observations, the arithmetic mean of all observed values, their standard deviation, their skewness and their kurtosis coefficient.

Obtaining statistical information

In the following example, a number of observable outputs `o` of a mathematical program are obtained as the result of changes in a single uniformly distributed input parameter `InputRate`. The interval range of every histogram is set to the interval `[0,100]` in 10 steps, and it is assumed that the set associated with index `i` has at least 12 elements.

Example

```

for (o) do
  HistogramCreate( HistogramID(o) );
  HistogramSetDomain( HistogramID(o), intervals: 10, left: 0.0, width: 10.0 );
endfor;

while ( LoopCount <= TrialSize ) do
  InputRate := Uniform(0,1);
  solve MathematicalProgram;
  for (o) do
    HistogramAddObservation( HistogramID(o), ObservableOutput(o) );
  endfor;
endwhile;

```

```

for (o) do
  HistogramGetFrequencies( HistogramID(o), Frequencies(o,i) );
  HistogramGetBounds( HistogramID(o), LeftBound(o,i), RightBound(o,i) );
  HistogramDelete( HistogramID(o) );
endfor;

```

19.4 Interfacing with the user interface

At particular times, for instance during the execution of user-activated procedures, you may have to specify an interaction between the model and the user through dialog boxes and pages. To accommodate such interaction, AIMMS offers a number of *interface functions* that perform various interactive tasks such as

*Interface
functions*

- opening and closing pages,
- printing pages,
- file selection and management,
- obtaining numeric, string-valued or element-valued data,
- selecting, loading and saving cases and datasets, and
- execution control.

All interface functions have an integer return value. For most functions the return value is 1 (success), or 0 (failure), which allows you to specify logical conditions based on these values. If you are not interested in the return value, the interface functions can still be used as procedures.

Return values

There are some interface functions that also return one or more output arguments. In order to avoid possible side effects, the return values of such functions can only be used in scalar assignments, and then they must form the entire right hand side.

*Limited use in
certain cases*

Whenever an interface function fails, an error message will be placed in the predefined AIMMS string parameter `CurrentErrorMessage`. The contents of this identifier always refer to the message associated with the last encountered error, i.e. AIMMS does not clear its contents. Within the execution of your model, however, you are free to empty `CurrentErrorMessage` yourself.

*Obtaining the
error message*

The following statements illustrate valid examples of the use of the interface functions `FileExists`, `DialogAsk`, and `FileDelete`.

Example

```

if ( FileExists( "Project.lock" ) ) then
  Answer := DialogAsk( "Project is locked. Remove lock and continue?",
    Button1 : "Yes", Button2 : "No" );

  if ( Answer = 1 ) then
    FileDelete( "Project.lock" );
  else
    halt;
  endif ;
endif ;

```

The interface function `DialogAsk` has a return value of 1 when the first button is pressed, and 2 when the second button is pressed.

19.4.1 Page functions

The possibility of opening pages from within a model provides flexibility compared to page tree-based navigation (see Section 12.1.2). Depending on a particular condition you can decide whether or not to open a particular page, or you can open different pages depending on the current status of your model.

Model page control

The following functions for manipulating pages are available in AIMMS.

Page functions

- `PageOpen(page)`
- `PageOpenSingle(page)`
- `PageClose([page])`
- `PageGetActive(page)`
- `PageGetFocus(page,tag)`
- `PageSetFocus(page,tag)`
- `PageSetCursor(page,tag,scalar-reference)`
- `PageRefreshAll`
- `PageGetChild(page, result-page)`
- `PageGetParent(page, result-page)`
- `PageGetPrevious(page, result-page)`
- `PageGetNext(page, result-page)`
- `PageGetTitle(page, title)`
- `PageGetUsedIdentifiers(page, identifier-set)`

The arguments *page*, *result-page*, *tag* and *title* are string arguments. The argument *scalar-reference* is a scalar reference to a data element associated with an (indexed) identifier and the output argument *identifier-set* must be a subset of `AllIdentifiers`.

With the `PageOpen` and `PageClose` functions you can open and close specific pages that are part of your model. The `PageOpenSingle` function will, in addition to opening a page, close all other pages that are currently open. You can use it, for instance, to return to the main menu of your application and close all data pages at the same time. If you do not provide a page name in the `PageClose` function, AIMMS will close the currently active page. To obtain the pagename of the currently active page, you can use the function `PageGetActive`.

Opening and closing pages

The function `PageSetFocus` provides you with even more control over the manner in which a page is opened. Every object on a page can be tagged by means of a descriptive string. With the `PageSetFocus` function you can open a page and set the focus on a particular tagged object. If the execution of user-initiated procedures depends on a precise object on a page from which it is called, you can use the function `PageGetFocus` to obtain the current page name as well as the tag of the object which currently has the focus on that page.

Setting and getting the focus

With the function `PageSetCursor` you have maximum control during the opening of a page. Not only can you indicate the object on the page, but you can also specify where the cursor should be positioned within the object. You do this by entering a scalar reference to the particular data element associated within the object that should have the focus. For example, if the cursor is to be positioned within an object at the field associated with the value `Transport('Amsterdam', 'Rotterdam')`, then this value should be entered as the third argument in the function. This function can be convenient for guiding the end-user of your application through a number of interrelated pages or objects.

Setting the cursor

The functions `PageOpen`, `PageOpenSingle`, `PageSetFocus`, and `PageSetCursor` will return immediately for standard end-user pages. When you have specified that a page is a dialog page (see Section 11.3), the page will appear as a dialog box, and these interface functions block until the dialog box is closed by the user. Dialog pages allow you to construct your own customized dialog boxes, while still using the ordinary interface elements offered by AIMMS.

Dialog pages

With the `PageRefreshAll` function you can refresh the contents of all pages during the execution of a procedure. This is useful, for instance, when you want to show intermediate results during a long computation, or want to provide a graphical representation of the progress of a solver, updated at regular intervals (using the solver callback features discussed in the AIMMS Language Reference). Note that AIMMS will automatically refresh all pages after the user-initiated execution of a procedure has ended.

Refreshing page contents

With the functions `PageGetChild`, `PageGetParent`, `PageGetPrevious` and `PageGetNext` you can obtain the first child page, the parent page, the previous and the next page relative to the position of the reference page named *page* in the page tree of your project (see also Section 12.1.2). The function `PageGetNextInTreeWalk` will provide you with the next page while traversing the page tree in a depth-first manner. This function includes hidden pages and ignores separators and can be used, for instance, to initialize the set of all pages that are present in your application. If *page* is an empty string, the location of the result page will be relative to the currently active page in the graphical user interface.

Obtaining navigation info

The function `PageGetTitle` can be used to obtain the title of a page in specified on the page **Properties** dialog box.

Page titles

With the function `PageGetUsedIdentifiers` you can create a list of identifiers that are used on a given page. This list consists of the identifiers being displayed on the page as well as identifiers that are used to specify object properties. This function can be used, for instance, (in combination with the function `PageGetNextInTreeWalk`) to programmatically generate a list of all identifiers that are used in the user interface of your application.

Listing used identifiers

19.4.2 Print functions

AIMMS provides a printing capability in the form of *print pages* (see Chapter 14). Ordinary pages and print pages are constructed in the same way. When you instruct AIMMS to print an ordinary page, the entire contents of the page, as you see it on the screen, are printed. When you print a print page, all interactive objects such buttons, list boxes, check boxes, radio buttons and drop-down lists are ignored. In addition, data objects on a print page that are too large to fit on a single sheet of paper, will be printed on multiple sheets.

Printing facilities

You can instruct AIMMS to print any print page from within the model by using print interface functions. In addition, the print interface functions offer you the capability of composing, and printing, a customized report consisting of multiple print pages. For instance, you could use these facilities to create ready-to-go faxes on the basis of your latest scheduling results.

Printing reports

The following functions are available for printing print pages in AIMMS.

Print functions

- `PrintPage(page[,filename][,from][,to])`
- `PrintStartReport(title[,filename])`
- `PrintEndReport`
- `PrintPageCount(page)`

The arguments *page*, *filename* and *title* are string arguments. The optional arguments *from* and *to* are integer arguments.

With the `PrintPage` function you can print a single print page. If the page contains a data object for which the available data does not fit onto a single page, AIMMS will print the object over multiple pages in a row-wise manner. Through the optional *from* and *to* arguments, you can limit the page range which will actually be printed.

Printing a single page

With the optional *filename* argument you can indicate that the print output should be directed to the specified file, rather than directly sending it to the default printer. The *filename* argument is ignored when the `PrintPage` function is surrounded by calls to the `PrintStartReport` and `PrintEndReport` functions (see below). You can use the *filename* argument, for instance, to make a printed report available to others using a default filename.

Printing to file

When you want to compose a report consisting of several existing numbered print pages, you can use the `PrintStartReport` and `PrintEndReport` functions. Following a call to the `PrintStartReport` function, all pages to be printed by subsequent calls to the `PrintPage` function will be collected. They will be printed as soon as the `PrintEndReport` function is encountered. If you specify the optional *filename* argument, the output will be sent to the indicated file.

Printing customized reports

During the printing of a report AIMMS will number pages consecutively. The page number is available to you through the predefined identifier `CurrentPageNumber`. You can use it on print pages to show the page number. AIMMS will reset the page number to 1 for every *single* page printed, as well as at the beginning of a printed report. By making assignments to `CurrentPageNumber` inside a pair of calls to `PrintStartReport` and `PrintEndReport`, however, you can modify the page numbering within a printed report as you desire.

Page numbers

You can use the function `PrintPageCount`, when you are interested in the number of sheets required to print a particular print page prior to actually printing it. The function returns the number of sheets of paper needed to print the page given the current print settings and data contained on the page.

Counting pages

19.4.3 File functions

The interactive execution of your model may involve various forms of file manipulation. For instance, the user might indicate which names to use for particular input and output files, or in which directory they are (to be) stored.

File manipulation

The following functions are available for file manipulation in AIMMS.

File functions

- FileSelect(*filename*[,*directory*][,*extension*][,*title*])
- FileSelectNew(*filename*[,*directory*][,*extension*][,*title*])
- FileDelete(*filename*[,*delete_readonly_files*])
- FileCopy(*oldname*,*newname*[,*confirm*])
- FileMove(*oldname*,*newname*[,*confirm*])
- FileAppend(*filename*,*appendname*)
- FileExists(*filename*)
- FileView(*filename*[,*find*])
- FileEdit(*filename*[,*find*])
- FilePrint(*filename*)
- FileTime(*filename*,*filetime*)
- FileTouch(*filename*,*newtime*)

The arguments *filename*, *directory*, *oldname*, *newname* and *appendname* are string parameters. The arguments *directory*, *extension*, *title*, *filetime* and *newtime* are all string arguments. The optional argument *confirm* must be 0 (default) or 1, while *find* is a string argument. All optional arguments must be tagged with their formal argument name.

The following functions are available for directory manipulation.

Directory functions

- DirectorySelect(*directoryname*[,*directory*][,*title*])
- DirectoryCreate(*directoryname*)
- DirectoryExists(*directoryname*)
- DirectoryGetCurrent(*directoryname*)
- DirectoryDelete(*directoryname*[,*delete_readonly_files*])
- DirectoryCopy(*oldname*,*newname*[,*confirm*])
- DirectoryMove(*oldname*,*newname*[,*confirm*])

The arguments *directoryname*, *directory*, *title*, *oldname* and *newname* are all string parameters. The *directory* and *title* arguments are optional. The optional argument *confirm* must be 0 (default) or 1.

The functions FileSelect and FileSelectNew both open a standard file selection dialog box, and let you select either an existing or a new file. The function DirectorySelect lets you select an existing directory name or create a new one. If you do not specify a starting directory, the dialog box will start in the current working directory. If you specify a *relative* directory path, then the dialog box will start in the specified directory relative to the current directory. Using the optional extension control in the dialog you can filter the files to be shown in the dialog box. The optional *title* argument will appear in the dialog box title.

Selecting a file or directory

You should keep in mind that these functions only bring up a dialog box and register the user's selection and action. Depending on the button clicked (**OK** or **Cancel**) the function returns a value of either 1 or 0. What happens next depends on the way you write your code.

Clicking OK or Cancel

With the functions `FileDelete`, `FileCopy`, `FileMove`, `DirectoryDelete`, `DirectoryCopy` and `DirectoryMove` you can delete a file or directory, or copy or move it to another file or directory. The function `DirectoryCreate` creates the given directory (without needing a dialog box). The functions `FileExists` and `DirectoryExists` let you verify whether the given file or directory exists in the file system. If you specify a relative pathname for a file or directory argument, AIMMS will assume that the path is relative to the current working directory. The working directory itself can be retrieved using the `DirectoryGetCurrent` function. Through the optional argument *delete_readonly_files* (with default 0) of the functions `FileDelete` and `DirectoryDelete` you can indicate whether you want read-only files to be deleted without further notice, or whether you want these functions to fail.

File and directory manipulations

The function `FileTime` will return the time at which a particular file was last saved. The resulting time is returned as a string with the form "YYYY-MM-DD hh:mm:ss". This can be transformed into any numeric time representation using the function `StringToMoment` discussed in the AIMMS Language Reference. The function `FileTouch` can be used to set the file time to the time as specified by the optional *newtime* argument. If omitted the modification time of the file is set to the current time.

File times

The delete, copy and move functions will accept wildcards (*) to delete, copy or move multiple files or directories. In these cases, the second argument must be a directory name to which the files can be copied or moved.

Use of wildcards

Using the `FileView` and `FileEdit` functions, you invoke the AIMMS editor to view or edit ASCII files from within the interface. In view mode it is not possible to modify the file. However, the **Cut**, **Copy**, **Find**, **Print** and **Save As** commands are still allowed. By specifying the optional *find* argument, AIMMS will search for the specified search string, and jump to its first occurrence in the selected file. You can use the `FilePrint` function to print an ASCII file from within your model. The file is sent to the default printer.

View, edit or print a file

19.4.4 Dialog box functions

During the execution of your model, it is very likely that you must communicate particular information with your user at some point in time. AIMMS supports two types of dialog boxes for user communication:

Two types of dialog boxes

- information dialog boxes, and
- data entry dialog boxes.

In addition to these standard dialog boxes available in AIMMS, it is also possible to create customized dialog boxes using dialog pages (see Section 11.3), and open these using the PageOpen function discussed in Section 19.4.1.

The following functions are available in AIMMS for displaying information to the user.

Information dialog boxes

- DialogMessage(*message*[,*title*])
- DialogError(*message*[,*title*])
- DialogAsk(*message*,*button1*,*button2*[,*button3*])
- DialogProgress(*message*[,*percentage*])
- StatusMessage(*message*)

The *message*, *title* and *button* arguments are strings. The *percentage* argument is a number between 0 and 100.

With the DialogMessage and DialogError functions you can display a dialog box containing your own message and an **OK** button. With the optional *title* argument, you can specify the title of the dialog box. In addition, the dialog box will respectively contain an information icon or an error icon.

Dialog box on the screen

You can use the DialogAsk function to obtain a user response. This function displays a dialog box containing a given message and two (or three) buttons with button text as given. The *button3* argument is optional, and has to be tagged with the formal argument name. The return value is either 1, 2, or 3 and matches the button that was pressed.

Getting user response

With the functions DialogProgress and StatusMessage you can provide progress information to the end-user of your model. The function DialogProgress will display a progress dialog box containing a message and (optionally) a progress meter under your control. The dialog box will disappear when you either call it with an empty message string, or when the execution from which it was called has ended. With the function StatusMessage you can display a message in the status bar.

Displaying progress messages

The following functions are available in AIMMS for scalar data entry dialog boxes.

Data entry dialog boxes

- DialogGetString(*message*,*reference*[,*title*])
- DialogGetElement(*title*,*reference*)
- DialogGetElementByText(*title*,*reference*,*element-text*)
- DialogGetElementByData(*title*,*reference*,*element-data*)
- DialogGetNumber(*message*,*reference*[,*decimals*][,*title*])

- `DialogGetPassword(message,reference[,title])`
- `DialogGetDate(title,date-format,date[,nr-rows][,nr-columns])`

The *message* and *title* arguments must be strings, and *reference* must be a (scalar) reference to an identifier of the appropriate type. The *element-text* argument must be a string parameter defined over the set in which *reference* is an element parameter, whereas *element-data* must be a string parameter defined over the *reference* set plus a single additional simple set. The *decimals* argument must be a nonnegative integer.

AIMMS offers three basic functions for obtaining a string-valued, element-valued, or numerical, scalar value from the user, `DialogGetString`, `DialogGetElement` and `DialogGetNumber`. These functions let you display a dialog box with your own message and dialog box title and an entry field for the scalar. AIMMS will only allow the user to enter a value that lies within the declared range of the scalar argument. The functions return 0 if the user presses the **Cancel** button, and 1 if the user presses the **OK** button.

Getting a user-supplied value

When displaying a data entry dialog box, the entry field displays the value of the scalar reference at the time of the call. You can use this to provide a default for the value that you want the user to supply. The default value in the function `DialogGetNumber` will be displayed with the number of decimal places as specified in the *decimals* argument.

Default value

When the elements in a set are not very descriptive to an end-user, AIMMS offers you an alternative way to have a user select a set element. With the function `DialogGetElementByText` you can supply an additional string parameter defined over the set from which you want the user to select an element. Instead of the element names, the dialog box will now display these descriptive texts.

Get element by description

Similarly, the function `DialogGetElementByData` displays string data divided into several columns from which the user can select a row corresponding to the desired element of a particular set. The function takes a two-dimensional string argument, defined over the set from which you want the user to select an element, plus an additional set of elements which will be used as column headers.

Description in multiple columns

With the function `DialogGetPassword` you can let the user enter a password. The function behaves like the function `DialogGetString` with the exception that the user-supplied input is not visible but shown as a sequence of '*' characters.

Obtaining password information

You can use the function `DialogGetDate` to let the end-user select a date that plays a role in your model, and store the resulting date in a string parameter. The *date-format* argument must be a date format specification string using the date- and time-specific components explained in Section 31.7.1 of the Language Reference. The *date* argument is an inout argument. If it contains a valid

Obtaining a date

date according to the specified format on input, AIMMS will set the initial date in the date selection dialog equal to the specified date. On output, the *date* argument contains the date selected by the user, according to the specified format.

With the optional *nr-rows* and *nr-columns* arguments of the `DialogGetDate` function, you can specify the number of rows and columns of months displayed in the date selection dialog (maximum 3 and 4 respectively, each with a default of 1). Thus, by specifying the maximum number of rows and columns you will be able to simultaneously display the days of 12 consecutive months within the dialog.

*Displayed
number of
months*

19.4.5 Data management functions

The management of cases and datasets is a very important aspect of a successful decision support system. While the complete data management functionality is available to the end-user within the **Data Manager** or from the **Data** menu, you may want to have additional control over the data management process to perform special tasks.

*Controlled data
management*

The following functions are available in AIMMS for performing case management tasks.

Case functions

- `CaseNew`
- `CaseFind(case-path,case)`
- `CaseCreate(case-path,case)`
- `CaseDelete(case)`
- `CaseLoadCurrent(case[,dialog])`
- `CaseMerge(case[,dialog])`
- `CaseLoadIntoCurrent(case[,dialog])`
- `CaseSelect(case[,title])`
- `CaseSelectNew(case[,title])`
- `CaseSetCurrent(case)`
- `CaseSave([confirm])`
- `CaseSaveAll([confirm])`
- `CaseSaveAs(case)`
- `CaseSelectMultiple([cases-only])`
- `CaseGetChangedStatus`
- `CaseSetChangedStatus(status[,include-datasets])`
- `CaseGetType(case,case-type)`
- `CaseGetDatasetReference(case,data-category,dataset)`
- `CaseCompareIdentifier(case1,case2,identifier,suffix,mode)`
- `CaseCreateDifferenceFile(case,filename,diff-types
 ,absolute-tolerance,relative-tolerance,output-precision)`
- `CaseWriteToSingleFile(outputfileName)`

- `CaseReadFromSingleFile(inputfileName)`

The arguments *case*, *case1* and *case2* are element parameters in the predefined set `AllCases`, whereas *title*, *case-path* and *filename* are string arguments. The *status* argument, and the optional arguments *dialog*, *include-datasets* and *cases-only* must be 0 or 1. The optional *confirm* argument can be 0, 1 or 2. The default of both the *confirm* and *dialog* arguments is 1, *cases-only* is 0 by default. The arguments *case-type*, *data-category*, *dataset*, *suffix* and *mode* are elements of the sets `AllCaseTypes` and `AllDataCategories`, `AllDatasets`, `AllSuffixes` and `AllCaseComparisonModes`, respectively, while the *diff-types* argument is an element parameter indexed over `AllIdentifiers` in the set `AllDifferencingModes`. The *absolute-tolerance*, *relative-tolerance* and *output-precision* arguments are numerical, scalar values.

The following functions are available in AIMMS for performing dataset management tasks.

Dataset functions

- `DatasetNew(data-category)`
- `DatasetFind(data-category,dataset-path,dataset)`
- `DatasetCreate(data-category,dataset-path,dataset)`
- `DatasetDelete(data-category,dataset)`
- `DatasetLoadCurrent(data-category,dataset[,dialog])`
- `DatasetMerge(data-category,dataset[,dialog])`
- `DatasetLoadIntoCurrent(data-category,dataset[,dialog])`
- `DatasetSelect(data-category,dataset[,title])`
- `DatasetSelectNew(data-category,dataset[,title])`
- `DatasetSetCurrent(data-category,dataset)`
- `DatasetSave(data-category[,confirm])`
- `DatasetSaveAll([confirm])`
- `DatasetSaveAs(data-category,dataset)`
- `DatasetGetChangedStatus(data-category)`
- `DatasetSetChangedStatus(data-category,status)`
- `DatasetGetCategory(dataset,data-category)`

The argument *data-category* must be an element parameter in the predefined set `AllDataCategories`, the argument *dataset* must be an element parameter in the set `AllDatasets`, whereas *title* and *dataset-path* are string arguments. The *status* argument, and the optional *dialog* argument, must be 0 or 1. The optional *confirm* argument can be 0, 1 or 2. The default of both the *confirm* and *dialog* arguments is 1.

The `CaseNew`, `CaseLoadCurrent`, `CaseMerge`, `CaseLoadIntoCurrent` functions, and their counterparts for datasets, have the same functionality as the corresponding items on the **Data** menu. With the optional *dialog* argument you can indicate whether you want a dialog box to be presented to the user. Without user input AIMMS will load the case or dataset provided as an argument of the function or return an error if no valid case or data set was provided. On

Data menu functions

its return, the case or dataset argument will contain the element of `AllCases` or `AllDatasets` associated with the case or dataset selected by the user. The return value can be:

- 0 : The user pressed the **Cancel** button.
- 1 : An error occurred during importation.
- 1 : The data was loaded successfully.

The functions `CaseSave`, `CaseSaveAll`, `CaseSaveAs`, and their counterparts for datasets, offer the same functionality as the corresponding items on the **Data** menu. With the optional *confirm* argument you can specify whether you want the save to be confirmed by the user. The possible values are:

- 0 : never confirm,
- 1 : only confirm when required by the case, or
- 2 : always confirm.

The *confirm* argument defaults to 1. The return values of the save functions are as above.

When you do not want your end-users to select a case type when saving a new case in the case-save-as dialog box, you can preset the case type from within the modeling language through the predefined element parameter `CurrentDefaultCaseType`. When this element parameter has a nonempty value, AIMMS will remove the case type drop-down list, and use the case type specified through `CurrentDefaultCaseType`.

With the functions `CaseFind`, `CaseCreate`, `DatasetFind` and `DatasetCreate` you can obtain the element of either the set `AllCases` or the set `AllDatasets` which is associated with a path to the indicated case or dataset. The functions `CaseFind` and `DatasetFind` will return 0 if no such case or dataset exists, while the functions `CaseCreate` and `DatasetCreate` will create nonexistent cases and datasets in exactly the same manner as if you were inserting new case or dataset nodes in the **Data Manager**.

With the functions `CaseDelete` and `DatasetDelete` you can delete cases from the case and data category tree without using the AIMMS **Data Manager**. If you are deleting the active case or an active dataset, AIMMS will retain the data associated with that case or dataset, but remove its reference from the active case and dataset settings.

The functions `CaseLoadCurrent` and `DatasetLoadCurrent` load a case or dataset as active, and set the current case or dataset to the loaded data file. You can *import* a case or dataset into your current case through either the `CaseMerge` or `CaseLoadIntoCurrent` functions and their counterparts for datasets.

*Saving cases
and datasets*

*Setting the
default case
type*

*Finding cases
and datasets*

*Deleting cases
and datasets*

*Loading cases
and datasets*

The functions `CaseSetCurrent` and `DatasetSetCurrent` let you set the current case or dataset *without loading any data*. As subsequent saves will save data in the current case or dataset, you should use these functions with care, and make sure that no data is inadvertently lost.

Setting the current case

With the functions `CaseSelect`, `CaseSelectNew`, `DatasetSelect` and `DatasetSelectNew` you can let the user select an existing or new case or dataset without actually opening it, or saving the current data to it. You can then further use this case, for instance, to import or export data using `READ` and `WRITE` statements.

Selecting a case or dataset

The function `CaseSelectMultiple` displays the **Multiple Cases** dialog box from the **Data** menu. With it, the user can select multiple cases from the case management tree, and use the selection for multiple case objects, calculations involving multiple cases, or creating your own batch run of cases. The selection made by the user is available to you through the predefined set `CurrentCaseSelection`. You can use it, for instance, to import selected data from all cases, and perform advanced case comparisons.

Selecting multiple cases

Sometimes you may want to check if the user has made changes to the data in the currently loaded case, or you may even want to change that status. The functions `CaseGetChangedStatus` and `CaseSetChangedStatus` do this. The status can be either 1 (case changed), or 0 (case unchanged). With the optional argument *include-datasets* you can indicate whether you also want to modify the status of all datasets included in the case. Similar functions are available for datasets.

Checking the case status

The function `CaseCompareIdentifier` can be used to compare two cases for a given identifier. For numerical identifiers this function returns the minimum, maximum, sum, average or total number of all data differences (depending on the *mode* argument). For non-numerical identifiers the total number of data differences is returned. To compare the data for all identifiers in a case at once and to dump the results in a text file you can use the function `CaseCreateDifferenceFile`. The resulting text file can then be used in a `READ` statement to apply the same differences to some other data instance.

Comparing identifier data

Through the functions `CaseWriteToSingleFile` and `CaseReadFromSingleFile` you can write single cases to and read single cases from ordinary disk files. Using ordinary disk files to store cases can be convenient when you want to share cases between multiple AIMMS sessions. A good example is when you use cases to store the state of a stateful web service session. This allows a single web service session to be serviced by multiple AIMMS instances in case the number of concurrent sessions is higher than the number of available AIMMS instances.

Saving single cases to disk

All data categories, datasets, case types and cases in an application are accessible in the model through a number of predefined sets and parameters. They are:

Case and dataset related identifiers

- the set `AllDataCategories`, containing the names of all data categories defined in the data manager setup window,
- the set `AllCaseTypes`, containing the names of all case types defined in the data manager setup window,
- the integer set `AllDataFiles`, representing all datasets and cases available with a particular project,
- the set `AllDatasets`, a subset of `AllDataFiles`, representing the collection of all datasets available in the project,
- the set `AllCases`, a subset of `AllDataFiles`, representing the set of all cases available for the project,
- the indexed element parameter `CurrentDataset` in `AllDatasets` and defined over `AllDataCategories`, containing the currently active datasets,
- the scalar element parameter `CurrentCase` in `AllCases`, and
- the scalar element parameter `CurrentDefaultCaseType` in `AllCaseTypes`.

You can obtain the case type for each case through the function `CaseGetType`. For every dataset you can ask AIMMS to return its data category through the function `DatasetGetCategory`. With the function `CaseGetDatasetReference` you can, for every data category, obtain a reference to the dataset of that category included in the case. If no dataset is included, the *dataset* is set to the empty element, and the function returns 1. If an included dataset is nonexistent, the *dataset* is also set to the empty element, but the function now returns 0.

Obtaining case type or data category

In the AIMMS **Data Manager**, data categories and case types are specified as a subcollection of identifiers from the model tree. Through the following functions you can obtain the contents of data categories and case types, should you need this information.

Data categories and case types

- `DataCategoryContents(data-category,identifier-set)`
- `CaseTypeContents(case-type,identifier-set)`
- `CaseTypeCategories(case-type,category-set)`

The argument *data-category* is an element of the set `AllDataCategories`. The argument *case-type* is an element of the set `AllCaseTypes`. The output arguments *identifier-set* and *category-set* must be subsets of `AllIdentifiers` and `AllDataCategories`, respectively.

The function `CaseTypeContents` will return a subset of identifiers which includes both the list of identifiers added to the case type itself, and the identifiers which are part of the data categories included in the case type. With the function `CaseTypeCategories` you can obtain the subset of data categories included in a case type, while the function `DataCategoryContents` returns the set of identifiers contained in a data category.

Case type contents

The mapping of the integer set `AllDataFiles` and its subsets onto the datasets and cases in the project is maintained by the **Data Manager**, and is not editable from within the model. Moreover, the numbering of cases and datasets may be different in every new session. During a session, however, the following functions give you access to all the information stored inside a datafile.

Obtaining datafile info

- `DataFileName(datafile,name)`
- `DataFileGetAcronym(datafile,acronym)`
- `DataFileGetPath(datafile,path)`
- `DataFileGetDescription(datafile,description)`
- `DataFileGetComment(datafile,comment)`
- `DataFileGetTime(datafile,time)`
- `DataFileGetOwner(datafile,user)`
- `DataFileGetGroup(datafile,group)`
- `DataFileReadPermitted(datafile)`
- `DataFileSetAcronym(datafile,acronym)`
- `DataFileSetComment(datafile,comment)`
- `DataFileWritePermitted(datafile)`
- `DataFileExists(datafile)`

The argument *datafile* is an element of the set `AllDataFiles`. The arguments *name*, *acronym*, *path*, *time*, *description*, *comment*, *user* and *group* are string parameters. The time that a case or dataset was last saved will be returned as for ordinary files.

You can use the functions `DataFileReadPermitted` and `DataFileWritePermitted` to check whether a read or write action is permitted by the current user before actually performing that action. With the functions `DataFileGetOwner` and `DataFileGetGroup` you can obtain the user name and associated user group of the owner of the data file as they are stored in the datafile by AIMMS. More details about case and dataset security are contained in Section [21.4](#).

Checking security

Because the AIMMS data tree can be accessed by multiple users, some elements in the set `AllDataFiles` may refer to data files that have been removed by other users. Using the function `DataFileExists` you can check for the existence of a particular datafile referenced by an element of `AllDataFiles`. Thus, you can prevent your users from receiving error message about nonexistent data files, which may have little meaning to them.

Checking data file existence

To copy an existing case or dataset the following function is available.

Copying data files

- `DataFileCopy(datafile-src,acronym,datafile-dest)`

The arguments *datafile-src* and *datafile-dest* are elements of the predefined set `AllDataFiles` and *acronym* argument is a string parameter.

The import and export facilities in the AIMMS **Data Manager** allow you transfer parts of the case and dataset tree to another user, or vice versa. Whenever you want such imports or exports to take place automatically, for example when particular datasets must be imported on a regular basis, the following functions are available to perform such tasks from within your model.

Importing and exporting data files

- `DataManagerExport(filename,datafiles)`
- `DataManagerImport(filename[,overwrite])`
- `DataImport220(filename)`

The argument *filename* is a string parameter. The argument *datafiles* is a subset of the predefined set `AllDataFiles`. The optional argument *overwrite* can be either 0, 1 or 2, and defaults to 0.

The function `DataManagerExport` exports the given set of data files to a newly created data manager file, deleting any previous contents. If the set *datafiles* contains cases with references to datasets which are not contained in *datafiles*, such datasets will also be exported. This ensures that any exported case refers to exactly the same data when imported by another user.

Exporting data files

The function `DataManagerImport` imports *all* cases and datasets within the given data manager file into the current case tree. With the optional *overwrite* argument, you can specify AIMMS' behavior when any case or dataset in the import file already exists in the case tree. The following values are allowed:

Importing data files

- 0 : the end-user decides (default),
- 1 : existing entries are overwritten, or
- 2 : AIMMS creates new nodes if existing entries are present.

When AIMMS creates a new node alongside an existing entry, the name of the existing node is prefixed with the string 'Imported', followed by a number if there are multiple imported copies for the imported node.

The function `DataImport220` allows you to import case files belonging to AIMMS 2.20 projects, which are incompatible with the new AIMMS 3 data storage scheme. You can use this function in an upgraded AIMMS 2.20 model, to upgrade cases created by end-users with your old AIMMS 2.20 project to the data manager tree of the (upgraded) AIMMS 3 project. The use of this function is especially useful when upgrading a case in your model requires additional data manipulation for example to store the label text of all set elements (which is no longer supported by AIMMS 3) in the original case file as string parameters in your AIMMS 3 model. The function returns 1 if the import succeeded, 0 if the user canceled the action, and -1 if the import failed.

Importing AIMMS 2.20 cases

19.4.6 Execution control functions

During the execution of your AIMMS application you may need to execute other programs, delay the execution of your model, get the command line arguments of the call to AIMMS, or even close your AIMMS application.

Execution control

The following execution control functions are available in AIMMS.

Control functions

- `Execute(executable[,commandline][,workdir][,wait][,minimized])`
- `ShowHelpTopic(topic,filename)`
- `OpenDocument(document)`
- `Delay(delaytime)`
- `ScheduleAt(starttime,procedure)`
- `ProjectDeveloperMode`
- `SessionArgument(argno, argument)`
- `ExitAimms([interactive])`

The arguments *executable*, *commandline*, *workdir*, *filename*, *document*, *topic*, *starttime*, *argument* and *marker* are string arguments, the argument *delaytime* is a real number, while the arguments *wait*, *minimized* and *interactive* must be all either 0 or 1. The argument *procedure* must be an element of the predefined set `AllProcedures`. The argument *argno* must be an integer greater than or equal to 1.

With the `Execute` function you can start another application. You can optionally supply a command line argument for the application, indicate whether AIMMS should wait for the termination of the application, and whether the application should be started in a minimized state or not. As a general rule, you should not wait for interactive window-based applications. Waiting for the termination of a program is necessary when the program carries out some external data processing which is required for the further execution of your model. If you do not specify a working directory, AIMMS assumes that the current directory is the working directory.

The Execute function

Be aware that certain commands (such as “dir” or “copy”) and features such as output redirection to file (“>” or “>>”) are executed by the DOS command shell rather than being executables themselves. If you want to make use of such features of the command shell, the application you call should be `command.com` (or `cmd.exe` if you want to make use of features of the Windows NT command shell) followed by the `/c` option to specify the specific command you want to be executed by the command shell, as illustrated in the following example.

Executing DOS commands

```
Execute( "command.com", "/c dir > dir.out" );
```

The function `ShowHelpTopic` starts up the help program with the indicated help file, and displays the requested topic. The function supports all of the help file formats described in Section 11.2. If you do not provide a particular help file, AIMMS will assume the default help file associated with your project.

*The function
ShowHelpTopic*

The function `OpenDocument` opens the indicated document using the default viewer associated with the document extension. You can use it, for instance, to display an HTML file using the default web browser installed on a particular machine. The *document* argument need be a local file name, it could be a URL pointing to a page on the World Wide Web as well.

*The function
OpenDocument*

With the `Delay` function you can block the execution of your model for the indicated delay time. You can use this function, for instance, when you want to change the particular slice of an identifier to be displayed on a page in the end-user interface at regular intervals. The delay time is specified in seconds.

*The Delay
function*

With the `ScheduleAt` function you can tell AIMMS that you want a particular procedure within your application to be run at a particular start time. The start time must be provided in the default format “YYYY-MM-DD hh:mm:ss”. The function `ScheduleAt` will return immediately, while the indicated procedure will be run at the first opportunity after the given time when no other (interactive) execution is taking place. This form of scheduled execution is useful, for instance, when you want to initiate data retrieval from an external source at regular intervals.

*The ScheduleAt
function*

The function `ProjectDeveloperMode` lets you verify, from within your model, whether a project is run in developer mode or in end-user mode. In either case, you might want to perform different actions, e.g. activate a different menu, or open a different set of pages. The function returns 1 if the project is run in developer mode, or 0 otherwise.

*The function
ProjectDeveloperMode*

When you open an AIMMS project from the command line, AIMMS allows you to add an arbitrary number of additional arguments directly after the project name. You can use these arguments, for instance, to specify a changeable data source name from which you want to read data into your model. With the function `SessionArgument` you can obtain the (string) value of argument *argno* (≥ 1). The function fails if the specified argument number has not been specified.

*The function
SessionArgument*

To allow you to quit your application from within your model, AIMMS offers the function `ExitAimms`. Using this function, you can close your application without user intervention. You can optionally indicate whether the application must be closed in an interactive manner (i.e. whether the user must be able to answer

*The ExitAimms
function*

any additional dialog box that may appear), or that the default response is assumed.

19.4.7 Debugging information functions

To help you investigate the execution of your model AIMMS offers several functions to control the debugger and profiler from within your model. In addition, a number of functions are available that help you investigate memory issues during execution of your model.

*Debugging
information*

The following execution information functions are available in AIMMS.

*Execution
information
functions*

- IdentifierMemory(*identifier*,*include-permutations*)
- MemoryStatistics(*filename*,*append-mode*],[*marker-text*],[*show-leaks-only*],[*show-totals*],[*show-since-last-dump*],[*show-mem-peak*],[*show-small-block-usage*])
- IdentifierMemoryStatistics(*identifier-set*,*filename*,*append-mode*],[*marker-text*],[*show-leaks-only*],[*show-totals*],[*show-since-last-dump*],[*show-mem-peak*],[*show-small-block-usage*],[*aggregate*])

The argument *filename* is a string argument, while the arguments *include-permutations*, *append-mode*, *marker-text*, *show-leaks-only*, *show-totals*, *show-since-last-dump*, *show-mem-peak*, *show-small-block-usage* and *aggregate* must be all either 0 or 1. The argument *identifier* must be an element of the pre-defined set AllIdentifiers and the argument *identifier-set* must be a subset of this same set.

The function IdentifierMemory can be used to retrieve information about the total amount of memory that is occupied by a specific identifier. Its *include-permutations* arguments can be used to indicate whether or not permutations of the identifier should be included in the reported number. Permutations are used by AIMMS to quickly perform all kinds of operations over permuted instances of an identifier.

*Displaying
memory
information*

Through the function MemoryStatistics you can let AIMMS print statistics collected by AIMMS' memory manager to the specified file. Memory statistics will only be collected if the global AIMMS option `memory_statistics` is on. Through the *append-mode* option you can indicate whether output is to be appended to an existing file, or if any existing contents is to be overwritten. If multiple statistics are printed to the same file, you can specify a *marker-text* that will be printed at the top of statistics. All other optional arguments are additional settings to select specific types of statistics to be printed.

*Displaying
memory
statistics*

To restrict the collection of statistics to a subset of identifiers you can use the `IdentifierMemoryStatistics`. This function has two additional arguments, one to indicate the subset of identifiers subject to the collection process and one optional argument to indicate whether a individual report for every identifier in the set or a single aggregated report should be created.

*Displaying
identifier
statistics*

The following profiler control functions are available in AIMMS.

Profiler control

- `ProfilerStart()`
- `ProfilerPause()`
- `ProfilerContinue()`
- `ProfilerRestart()`

The profiling functions to control the AIMMS profiler allow for some extra flexibility over their use through the menu commands in the sense that it is easier to restrict profiling to a certain part of you model using these functions. The functions `ProfilerStart`, `ProfilerPause` and `ProfilerContinue` are available to start, pause and continue a profiler session. With the function `ProfilerRestart` the measurment data of all statements and definitions is reset.

19.4.8 Obtaining license information

The licensing functions discussed in this section allow you to retrieve licensing information during the execution of your model. Based on this information you may want to issue warnings to your end-user regarding various expiration dates, or adapt the execution of your model according to the capabilities of the license.

*License
information
functions*

The following licensing functions are available in AIMMS.

*License
functions*

- `LicenseNumber(license)`
- `LicenseStartDate(date)`
- `LicenseExpirationDate(date)`
- `LicenseMaintenanceExpirationDate(date)`
- `LicenseType(type,size)`
- `VARLicenseCreate(var-license,license-number,module-code[,userdata
[,expiration-date][,days-left-warning][,number-of-users][,is-network])`
- `VARLicenseExpirationDate(var-license,date)`
- `AimmsRevisionString(revision)`

All arguments are strings. The *var-license* argument is input, all other arguments are output arguments, except for the function `VARLicenseCreate` which has only input arguments.

Through the function `LicenseNumber` you can retrieve the license number of the currently active AIMMS license. It will return a string such as “015.090.010.007” if you are using an AIMMS 3 license, or a string such as “1234.56” if you are using an AIMMS 2 license.

License number

You can use the functions `LicenseStartDate`, `LicenseExpirationDate` and `LicenseMaintenanceExpirationDate` to obtain the start date, expiration date and maintenance expiration date of the currently active AIMMS license, respectively. All dates will be returned in the format “YYYY-MM-DD”. If a particular date has not been specified in the AIMMS license, AIMMS will return “No start date”, “No expiration date” or “No maintenance expiration date”, respectively.

License dates

The function `LicenseType` will return type and size information of the currently active AIMMS license. Upon success, the *type* argument contains the license type description (e.g. "Economy") and the *size* argument contains a description of the license size (e.g. "Large").

License type

Through the function `VARLicenseCreate` you can create a VAR license to be used with your project. In the *var-license* argument you must specify the name of the VAR license (file), as you have specified it in the **Project Security** dialog box, or in the attribute window of the main model node or a section of your model. The *license-number* argument must be specified in the format “015.090.010.007”, and the *expiration-date* argument in the format “YYYY-MM-DD”. All other arguments directly correspond to the input fields in the **VAR License Manager** dialog box discussed in Section 21.2.

VAR license creation

Through the function `VARLicenseExpirationDate` you can obtain the expiration date of a VAR license that is used within your project. In the *var-license* argument you must specify the name of the VAR license (file), as you have specified it in the **Project Security** dialog box, or in the attribute window of the main model node or a section of your model. The expiration date will be returned in the format “YYYY-MM-DD”. You can find more information about VAR licensing in Section 21.2.

VAR license expiration

You can use the function `AimmsRevisionString` if you want to obtain the revision number of the currently running AIMMS executable. The revision string returned by the function has the format “x.y.b” where *x* represents the major AIMMS version number (e.g. 3), *y* represents the minor AIMMS version number (e.g. 0), and where *b* represents the build number (e.g. 476) of the current executable. You can use this function, for instance, to make sure that your end-users use an AIMMS version that is capable of certain functionality which was not available in earlier AIMMS releases.

AIMMS revision