AIMMS User's Guide - User Interface Language Components

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

This documentation was typeset by AIMMS B.V. using LATEX and the LUCIDA font family.

Part V

Miscellaneous

Chapter 17

User Interface Language Components

This chapter

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

- 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.

17.1 Updatability of identifiers

Dynamic control required

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.

Multiple phases in your application

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.

Indicating input and output status

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

- 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`

The set `CurrentInputs` (which is a subset of the predefined set `AllUpdatable-Identifiers`) 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 `AllUpdatable-Identifiers`

The set `AllUpdatableIdentifiers` is computed by Aimms when your model is compiled, and contains the following 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.

17.2 Setting colors within the model

Color as indicator

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.

Setting colors in the model

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.

The set `AllColors`

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.

Color parameters

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.

Example

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.

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

17.2.1 Creating non-persistent user colors

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.

- `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.

Adding non-persistent colors

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.

Deleting and modifying 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.

Retrieving RGB-values

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.

17.3 Interfacing with the user interface

Interface functions

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

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

Return values

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.

Limited use in certain cases

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.

Obtaining the error message

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.

Example

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

```
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.

17.3.1 Page functions

Model page control

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.

Page functions

The following functions for manipulating pages are available in AIMMS.

- `PageOpen`(*page*) Opens page *page*.
- `PageOpenSingle`(*page*) Opens page *page* and closes all other.
- `PageClose`([*page*]) Closes page *page*, if *page* is not specified, closes active page.
- `PageGetActive`(*page*) Returns the active page in *page*.
- `PageGetFocus`(*page*,*tag*) Returns the name of the page and object that have focus in *pagePar* and *tag*
- `PageSetFocus`(*page*,*tag*) Sets the focus to object *tag* on page *page*.
- `PageSetCursor`(*page*,*tag*,*scalar-reference*) Position the cursor of object *tag* on page *page* to *scalar-reference*.
- `PageRefreshAll` Ensure that the open pages are refreshed with the current data.
- `PageGetChild`(*page*, *childpage*) Return the name of the page that is the first child of *page* in *childpage*, if any.
- `PageGetParent`(*page*, *parentpage*) Return the name of the page that is the parent of *page* in *parentpage*.
- `PageGetPrevious`(*page*, *previouspage*) Return the name of the page that is the previous page of *page* in *previouspage*.
- `PageGetNext`(*page*, *result-page*) Return the name of the page that is the next page of *page* in *nextpage*.
- `PageGetNextInTreeWalk`(*page*, *nextpage*) Return the name of the page that is the next page of *page* in a depth first tree walk over the page tree.
- `PageGetTitle`(*pageName*, *pageTitle*) Return the title of a specific page.
- `PageGetUsedIdentifiers`(*page*, *identifier_set*) Return the identifiers used in *identifier_set*.

17.3.2 Print functions

Printing facilities

AIMMS provides a printing capability in the form of *print pages*, see Chapter 14.

Print functions

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

- `PrintPage`(*page*[,*filename*][,*from*][,*to*]) Print *page* to file *filename*.
- `PrintStartReport`(*title*[,*filename*]) Start a print job with name *title*.
- `PrintEndReport` End the current print job.

- PrintPageCount(*page*) The number of sheets needed to print *page*.

17.3.3 File functions

File manipulation

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 functions

The following functions are available for file manipulation in Aimms.

- FileSelect(*filename*[,*directory*][,*extension*][,*title*]) Dialog to select an existing file.
- FileSelectNew(*filename*[,*directory*][,*extension*][,*title*]) Dialog to select a new file.
- FileDelete(*filename*[,*delete_readonly_files*) Delete a file.
- FileCopy(*oldname*,*newname*[,*confirm*]) Copy a file.
- FileMove(*oldname*,*newname*[,*confirm*]) Rename or move a file.
- FileAppend(*filename*,*appendname*) Append to an existing file.
- FileExists(*filename*) Is *filename* an existing file?
- FileView(*filename*[,*find*]) Opens *filename* in read only mode.
- FileEdit(*filename*[,*find*]) Opens *filename* for text editing.
- FilePrint(*filename*) Print a text file to printer.
- FileTime(*filename*,*filetime*) Return the modification time.
- FileTouch(*filename*,*newtime*) Set the modification time to now.

Directory functions

The following functions are available for directory manipulation.

- DirectorySelect(*directoryname*[,*directory*][,*title*]) Select an existing directory.
- DirectoryCreate(*directoryname*) Create a directory
- DirectoryExists(*directoryname*) Is *directoryname* an existing directory.
- DirectoryGetCurrent(*directoryname*) Return the directory.
- DirectoryDelete(*directoryname*[,*delete_readonly_files*) Delete a directory.
- DirectoryCopy(*oldname*,*newname*[,*confirm*]) Copy a directory
- DirectoryMove(*oldname*,*newname*[,*confirm*]) Move or rename a directory.

17.3.4 Dialog box functions

Two types of dialog boxes

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:

- 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 17.3.1.

Information dialog boxes

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

- `DialogMessage`(*message*[,*title*]), and `DialogError`(*message*[,*title*]) Both show *message* until **OK** button is pressed. They differ in icons displayed.
- `DialogAsk`(*message*,*button1*,*button2*[,*button3*]) Show *message* and offer two or three choices.
- `DialogProgress`(*message*[,*percentage*]) Show *message* and progress bar. Execution is continued.
- `StatusMessage`(*message*) Show *message* at the bottom of the AIMMS window.

Data entry dialog boxes

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

- `DialogGetString`(*message*,*reference*[,*title*]) Get a string.
- `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*])

17.3.5 Case management functions

There are several functions and identifiers available to support case management tasks. The functions can be divided into three groups:

- *Basic* – These functions perform the core case management tasks; they do not involve any dialogs.
- *Dialog* – These functions handle the dialogs around case management functions; they do not do any basic case management tasks.
- *Menu Replacement* – These functions execute similarly as the default actions behind the **data** menu.

Each of these three groups of functions, and the predeclared identifiers, are briefly presented below. For details about a particular function or identifier, the reader is referred to the Function Reference.

Basic case functions

The following functions are available in AIMMS for performing basic case management tasks without invoking dialogs.

- CaseFileLoad(*url*[,*keepUnreferencedRuntimeLibs*]) Load a case file and use its name as the active case.
- CaseFileMerge(*url*[,*keepUnreferencedRuntimeLibs*]) Merge a case file in.
- CaseFileSave(*url*,*contents*) Save the data to a file.
- CaseFileGetContentType(*url*,*contentType*) Get the current content type.
- CaseFileURLtoElement(*url*[,*caseFileElement*]) Find or create an element in AllCases corresponding to *url*.
- CaseCompareIdentifier(*case1*,*case2*,*identifier*,*suffix*,*mode*) Check whether the data of an identifier differs in two case files.
- CaseCreateDifferenceFile(*case*,*filename*,*diff-types* ,*absolute-tolerance*,*relative-tolerance*,*output-precision*)

Here the arguments are:

- *case*, *case1* and *case2* are element parameters in AllCases.
- *url*, *case-path*, and *filename* are strings.
- *contents* an element of AllCaseFileContentTypes
- *contentType* an element parameter in AllSubsetsOfAllIdentifiers
- *keepUnreferencedRuntimeLibs*, 0 or 1, default 1.
- *identifier* in AllIdentifiers
- *suffix* in AllSuffixNames
- *mode* in AllCaseComparisonModes
- *diff-type* in AllDifferencingModes
- *absolute-tolerance*, *relative-tolerance* and *output-precision* arguments are numerical, scalar values.

Case dialog functions

The following functions are available that handle the dialogs around case management, but do not perform the actual case management tasks:

- CaseDialogConfirmAndSave() Handles the standard "Save your data before continuing" dialog.
- CaseDialogSelectForLoad(*url*) Handles the dialog for selecting a case file.
- CaseDialogSelectForSave(*url*, *contentType*) Handles the dialog for saving data and selecting a content type.
- CaseDialogSelectMultiple(*caseSelection*) Handles the selection of multiple cases.

Here the arguments are:

- *url* a string parameter
- *contentType* an element parameter in AllCaseFileContentTypes
- *caseSelection* a subset of AllCases,

Data manamement functions

The function DataManagementExit() checks whether any data should be saved according to the active data management style. If any of the data needs saving, a dialog box is displayed, in which the user can select to save the data, not to save the data, or to cancel the current operation.

Data menu functions

These functions emulate the default menu items of the **Data** menu, they do not have any arguments.

- CaseCommandLoadAsActive() The default action behind the **Data - Load Case - As Active** menu item.
- CaseCommandLoadIntoActive() The default action behind the **Data - Load Case - Into Active** menu item.
- CaseCommandMergeIntoActive() The default action behind the **Data - Load Case - Merging into Active** menu item.
- CaseCommandNew() The default action behind the **Data - New Case** menu item.
- CaseCommandSave() The default action behind the **Data - Save Case** menu item.
- CaseCommandSaveAs() The default action behind the **Data - Save Case As** menu item.

Case file related identifiers

There are a number of predeclared identifiers available for the management of case files. They are:

- the set AllCases, a subset of AllDataFiles, contains the references to the case files accessed during the current AIMMS session,
- the parameter CurrentCase in AllCases is the reference to the current case,
- The parameter CurrentCaseFileContentType specifies the default case content type,
- the set AllCaseFileContentTypes contains those subsets of AllIdentifiers that are used to save data, and
- the string parameter CaseFileURL contains, for each case file referenced, the url as a string.

17.3.6 Execution control functions

Execution control

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.

Control functions

The following execution control functions are available in AIMMS.

- Execute(*executable*[,*commandline*][,*workdir*][,*wait*][,*minimized*])
- ShowHelpTopic(*topic*[,*filename*])
- OpenDocument(*document*)

- `Delay`(*delaytime*)
- `ScheduleAt`(*starttime*,*procedure*)
- `ProjectDeveloperMode`
- `SessionArgument`(*argno*, *argument*)
- `ExitAimms`([*interactive*])

17.3.7 Debugging information functions

Debugging information

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.

Execution information functions

The following execution information functions are available in AIMMS.

- `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*])

Profiler control

The following profiler control functions are available in AIMMS.

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

17.3.8 Obtaining license information

License information functions

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 functions

The following licensing functions are available in AIMMS.

- `LicenseNumber`(*license*)
- `LicenseStartDate`(*date*)
- `LicenseExpirationDate`(*date*)
- `LicenseMaintenanceExpirationDate`(*date*)

- `LicenseType`(*type*,*size*)
- `AimmsRevisionString`(*revision*)