
AIMMS User's Guide - Localization and Unicode Support

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com or order your hard-copy at www.lulu.com/aimms.

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

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

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 23

Localization and Unicode Support

When you are creating an end-user interface around your modeling application, you will most likely create the end-user interface in either your native language or in a common language like English. Which language you choose most probably depends on the intended user group of your application. In the case that you are requested to distribute your application to end-users who are not fluent in the language in which you originally developed the end-user interface, AIMMS offers a localization procedure which automatically separates all static texts used in the end-user interface of your application. This allows you to provide a relatively smooth translation path of your application to the native language(s) of your end-users.

*Interface
localization*

If you have end-users on the Asian market who require a native version of your AIMMS application, only making use of AIMMS' built-in localization procedure is not sufficient, as Asian languages require the use of double-byte characters to represent native strings. To support you in such cases, AIMMS is also available in a separate Unicode version. The AIMMS Unicode version allows you to use (double-byte) Unicode characters in strings and set element descriptions in both your model and its end-user interface, and offers full support for communicating with files and databases containing either ASCII or Unicode data.

*AIMMS Unicode
version*

This chapter illustrates how to use the automated localization procedure built into AIMMS, and explains how you can use it to create a foreign version of an end-user application. In addition, it describes the capabilities and limitations of the AIMMS Unicode version, as well as the necessary steps to run a project with the AIMMS Unicode version.

This chapter

23.1 Localization of end-user interfaces

Conceptually, localization of an end-user application consists of a number of basic steps. These basic steps are to

Basic concepts

- find all the strings that are used in the pages and menus of your end-user interface of your application,
- store these strings separate from the other interface components, and

- provide translations in different languages of these separately stored strings.

Through the **Tools-Localization** menu, AIMMS offers an integrated localization tool which can perform the first two steps for you automatically. The result is a list of strings, each with a description of its origin, which can be easily translated to other languages. This section will explain the use of the localization tool built into AIMMS step by step.

If your application consist of multiple library projects (see also Chapter 3), developed and maintained by different modelers, each of these libraries can have its own **Localization** section and identifiers to store its localization strings. When performing the localization conversion on a library project, all localized pages and menus in a library project will refer to the library-specific localization identifiers. This allows a developer of a library project to introduce localization into his library, independently of all other libraries and/or the main project.

Localization and libraries

Before you can start the final localization conversion of your AIMMS application, AIMMS needs to

Setting up localization support

- add a **Localization** section to the main model or library module which contains a default setup for working with a localized end-user interface of either the main project or library project, and
- register the names of the identifiers and procedures which are necessary for storing, loading and saving the strings used in the end-user interface of your application or library.

You can perform these steps through the **Tools-Localization-Setup** menu. As a result, AIMMS will add the (default) **Localization** section to your model or library if such a section has not already been added before. Secondly, through the dialog box presented in Figure 23.1, AIMMS will request the names of the

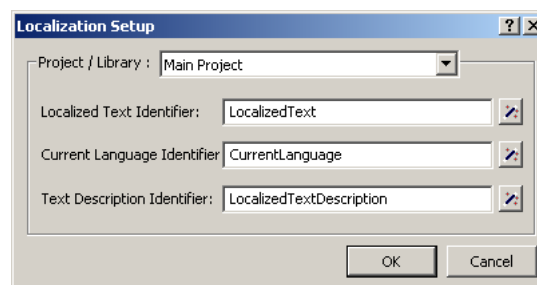


Figure 23.1: Setting up localization support

identifiers to be used further on in the localization process to store the strings used in the end-user interface of the main project or library. By default, AIMMS

proposes the identifiers added for this purpose to the (newly added) **Localization** section. If you change the names of these identifiers, or want to use completely different identifiers, you can execute the **Tools- Localization-Setup** menu again to specify the modified names.

If you are adding localization support to a library project, AIMMS lets you choose whether the language to be used within the library project should follow the global language selection of the entire application, or whether you want the language selection for the end-user interface of your library to be library-specific.

Selecting the language

After the localization setup has been executed for the first time, your model or library module has been extended with a new section called **Localization**. The contents of this model section is illustrated in Figure 23.2. The declaration

Localization section

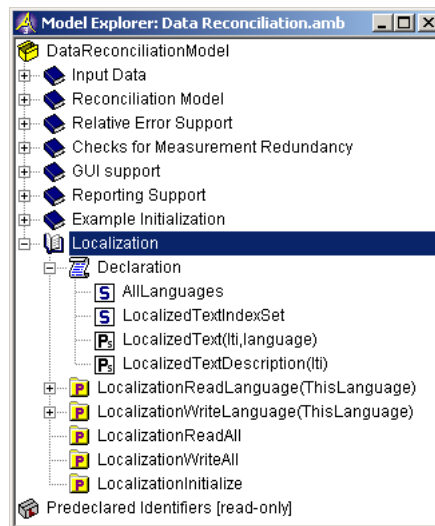


Figure 23.2: Localization section in the model tree

section contained in it declares the default set and string parameters used for storing all localization information.

- The set `AllLanguages` contains the names of all languages to which you want to localize your application. You can add as many languages to its definition as necessary. *However, you should make sure that, at any time, the first element in the set is your development language:* during the conversion process described below, AIMMS will associate all strings in the end-user interface with the first language from the set `AllLanguages`.
- Associated with the set `AllLanguages` is an element parameter `CurrentLanguage`, through which you (or your end-users) can select the language in which all texts in the end-user interface are to be displayed.

- The set `LocalizedTextIndexSet` is a subset of the predefined set `Integers`, and is used to number all strings within your end-user interface that are replaced by AIMMS during the conversion process.
- The string parameter `LocalizedText` contains the actual texts for all string objects in your end-user interface for one or more languages. During the localization conversion process, AIMMS will fill this parameter with the texts of your development language.
- The string parameter `LocalizedTextDescription` contains a short description of the origin of all converted string objects, and is filled by AIMMS during the localization conversion.

Through the **Tools-Localization-Setup** menu, you can modify the localization parameters which AIMMS will use during any subsequent conversion process. If you choose to select different identifiers, you should make sure that:

Using other localization identifiers

- the identifier selected for the **Localized Text Identifier** is a 2-dimensional string parameter, the identifier selected for the **Current Language Identifier** is a scalar element parameter, and the identifier selected for the **Text Description Identifier** is a 1-dimensional string parameter.
- the second index set of the **Localized Text Identifier** and the range set of the **Current Language Identifier** coincide. AIMMS will interpret the resulting set as the set of all languages.
- the first index set of the **Localized Text Identifier** and the first index set of the **Text Description Identifier** coincide and is a subset of the predefined set `Integers`. AIMMS will use this set to number all string objects during the conversion process.

In addition to the sets and string parameters discussed above, the **Localization** section also contains a number of procedures added for your convenience to perform tasks such as:

Localization procedures

- loading and saving the localized text for a single language,
- loading and saving the localized texts for all languages, and
- to initialize support for a localized end-user interface.

The statements within these procedures refer to the default localization identifiers created by AIMMS. If you have chosen different identifiers, or want to store the localization data in a nondefault manner, you can modify the contents of these procedures at your will. You must be aware, however, that the facilities within AIMMS to view and modify the localized text entries do not use these procedures, and will, therefore, always use the default storage scheme for localized data (explained later in this section).

The localization procedure **LocalizationInitialize** added to the **Localization** section of your model will read the localized text for a single language. If the element parameter **CurrentLanguage** has been set before the call to **LocalizationInitialize**, AIMMS will read the localized strings for the language selected through **CurrentLanguage**. If **CurrentLanguage** has no value, the procedure will read the localized strings for the first language (i.e. your development language).

The initialization procedure

If your model contains the (default) procedure **MainInitialization** (see also Section 4.2), a call to the procedure **LocalizationInitialize** will be added to the end of the body of **MainInitialization** during the first call to the **Tools-Localization-Setup** menu. This makes sure that the localized strings on pages and in end-user menus of a converted end-user interface contain the proper (original or localized) texts when the project is opened.

Added to Main-Initialization

Through the **Tools-Localization-Convert** menu you can instruct AIMMS to replace all static string occurrences in your (end-user and print) pages, templates and end-user menus by references to the localization identifiers selected during the localization setup. During the conversion, AIMMS

Performing the localization conversion

- scans all pages, templates and menus for static strings,
- creates a new localized entry in the **Localized Text Identifier** for each such string, and
- in the interface component where the static string was found, replaces it by the corresponding reference to the **Localized Text Identifier**. If a localization setup is defined per library, AIMMS will use the library-specific **Localized Text Identifier**.

In addition, AIMMS will, for each localized string, create a description in the **Localized Text Description Identifier**, initialized with the name of the page or menu plus the object in which the corresponding string was found. This may help you to link localization texts to specific objects and pages.

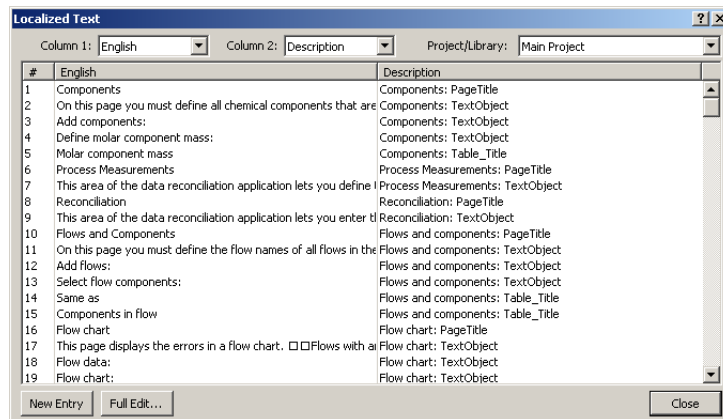
String description

During the localization conversion, AIMMS will warn for any duplicate string it encounters. For such duplicate strings, you have the opportunity to create a new entry in the **Localized Text Identifier** or to re-use an existing entry. Re-using existing entries can be convenient for common strings such as “Open” or “Close” that occur on many pages.

Duplicate occurrences

Once you have performed the localization conversion, you can view all localized strings through the **Tools-Localization-Show Strings** menu, which will open the dialog box illustrated in Figure 23.3. In this dialog box, AIMMS displays a numbered list of all localized strings, along with the description of the origin of each string. The string numbers exactly correspond to the elements of the set **LocalizedTextIndexSet** discussed above.

Editing localized strings

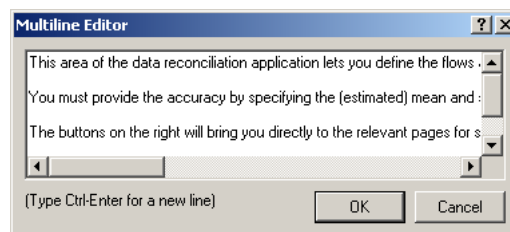
Figure 23.3: The **Localized Text** dialog box

Through the drop down lists at the top of the **Localized Text** dialog box of Figure 23.3, you can select the contents of the first and second string columns, respectively. For each column, you can select whether to display the localized text for any language defined in the set `AllLanguages`, or the description associated with each string. By viewing the localized strings for two languages alongside, you can easily provide the translation of all localized strings for a new language on the basis of the localized strings of, for example, your development language.

Modifying dialog box contents

If a localized string consists of multiple lines, you can invoke a multiline editor dialog box to edit that string through the **Full Edit** button at the bottom of the **Localized Text** dialog box, as illustrate Figure 23.4. To invoke this multi-

Modifying multiline strings


Figure 23.4: The **Multiline Editor** dialog box

line editor for the string corresponding to a particular language, click on the localized text for that language, and press the **Full Edit** button. The multiline editor will now be opened with the exact string that you selected in the **Localized Text** dialog box.

If you have added new pages, page objects, or end-user menus to your project after running the localization conversion procedure for the first time, you have two options to localize such new interface components. More specifically, you can

Localizing new texts

- localize every new component separately through the **Localized Text** wizard present at all text properties of the object, or
- run the localization conversion procedure again.

Whenever a string is associated with a property of a page, page object or menu item, the wizard button  of such a property in the **Properties** dialog box provides access to the **Localized Text** wizard, as illustrated in Figure 23.5. Invoking this wizard will open the **Localized Text** dialog box illustrated in

The Localized Text wizard

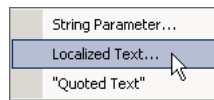


Figure 23.5: The **Localized Text** wizard

Figure 23.3, in which you can either select an existing localized string, or create a new entry through the **New Entry** button. Notice that the **Localized Text** wizard only shows the localization strings for the main or library project you are currently editing, and any of the included library projects which have the localization identifiers in their public interface. After closing the dialog box, AIMMS will add a reference to the localized text identifier in the edit field of the property for which you invoked the wizard, corresponding to the particular string selected in the **Localized Text** dialog box.

If you have added several new interface components without worrying about localization aspects, your safest option is to simply run the localization conversion procedure again. As a result, AIMMS will re-scan all pages, templates and menus for strings that are not yet localized, and add such strings to the list of already localized texts as stored in the localization identifiers associated with your project. Obviously, you still have to manually provide the proper translations to all available languages for all newly added strings.

Performing the conversion procedure again

By default, AIMMS stores the localization data as *project user files* containing standard AIMMS data statements within the project file (see also Section 2.5.2). The localized strings for every language, as well as the string descriptions are stored in separate user project files, as illustrated in Figure 23.6. The read and write statements in the bodies of the localization procedures added to the **Localization** section of your model, assume this structure of project user files for localization support.

Localized text storage

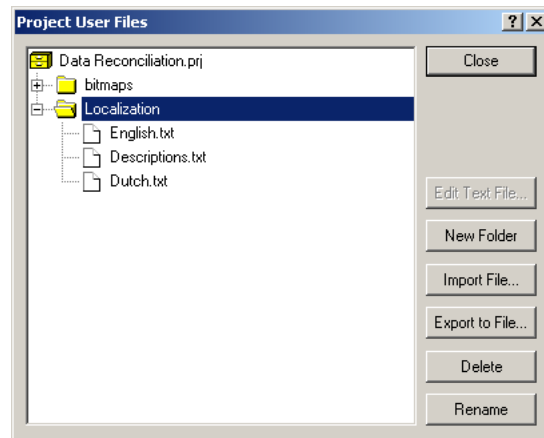


Figure 23.6: Default of localization data as user project files

Whenever you use the **Localized Text** dialog box of Figure 23.3, either through the **Tools-Localization-Show Strings** menu or by invoking the **Localized Text** wizard, AIMMS will make sure that the contents of appropriate localization data files are read in before displaying the localization data for a particular language. Likewise, AIMMS will make sure that the contents of the appropriate project user files are updated when you close the **Localized Text** dialog box.

Automatically updated

By using the import and export facilities for project user files (see also Section 2.5.2), you can also edit the data files containing the localized strings outside of AIMMS. This can be a convenient option if you hire an external translator to provide the localized texts for a particular language, who has no access to an AIMMS system. Obviously, you have to make sure that you do not make changes to these files through the **Localized Text** dialog box, while they are exported. In that case, importing that file again will undo any additions or changes made to the current contents of the project user file.

Manual edits

Besides the static strings in the end-user interface of your AIMMS application, the model itself may also contain references to static strings or to sets whose elements are defined within the model itself. Such strings and set elements are left untouched by AIMMS' localization procedure. If your model contains such string or set element references, you still have the task to replace them by references to a number of appropriate localized string and element parameters.

Static strings in the model

23.2 The AIMMS Unicode version

When you need to distribute a localized version of an AIMMS application to, for instance, Asian or Russian end-users, use of the common single-byte AIMMS version may not be sufficient anymore, as many languages in these regions

AIMMS Unicode version

cannot be represented by means of single-byte characters. To support localization to such languages, a Unicode version of AIMMS is available, in which all strings are represented internally through double-byte characters.

The AIMMS Unicode version is distributed as a separate installation program, and is available on the AIMMS installation CD-ROM or from the AIMMS website. Running the AIMMS Unicode installation program will install the AIMMS Unicode version alongside the ordinary single-byte AIMMS version. The AIMMS Unicode version can be run with your existing AIMMS license. When a valid single-byte AIMMS version has already been installed on your computer, the Unicode installation procedure will automatically copy your existing license files to the AIMMS Unicode installation directory.

Installing the Unicode version

When you use the AIMMS Unicode version, the set element descriptions and the data of string parameters, as well as any other string data used in the model or end-user interface will consume twice as much memory as with the single-byte AIMMS version. In most cases, this will result in a moderate increase in memory usage. If your model contains a lot of set elements or string data, however, you may want to make sure the memory usage of the AIMMS Unicode version is still acceptable.

Increased memory requirements

When you have developed an AIMMS application using the ordinary (single-byte) AIMMS version, you can relatively easily convert your project to the AIMMS Unicode version, as all single-byte strings can be represented by double-byte Unicode strings without problems. As of yet, however, the binary `.amb` model files used by the single-byte and Unicode AIMMS version are incompatible. The project files used by both version are compatible. The next section explains how you can prepare your projects for use with the Unicode version.

Converting to the Unicode version

To prepare your project for use with the AIMMS Unicode version, the following steps are required:

Conversion plan

- open your project with the single-byte AIMMS version, and open your model with the **Model Explorer**,
- save your model as an ASCII `.aim` file through the **File-Save As** menu,
- associate the newly created `.aim` file with your project through the **File-Open-Model** menu (thereby selecting the `.aim` file),
- save the project, and re-open the project with the AIMMS Unicode version.

After these steps you can proceed developing your project with the Unicode version. If you make changes your model, the model will be saved again in a binary `.amb` file. This `.amb` file will, however, contain the model text in Unicode format, and is incompatible with the single-byte AIMMS version.

By saving your model in an `.aim` file in the AIMMS Unicode version, you can also convert your project back to the ordinary single-byte AIMMS version. To create an ASCII `.aim` file in the AIMMS Unicode version, which can be read in by the single-byte AIMMS version, you must make sure that the option `aim_output_character_set` is set to `ascii` prior to saving the model to an `.aim` file. *Note, that for the back-conversion to be successful, you must make sure that your Unicode project or model file does not contain any genuine Unicode characters (i.e. double-byte characters which are not representable as single-byte characters).* In that case, the corresponding texts cannot be read into the ordinary single-byte AIMMS version, and loss of parts of your model or end-user interface may result.

Converting to the single-byte version

Both the single-byte and the Unicode AIMMS versions have been extended with Unicode I/O capabilities. The following list contains the Unicode I/O capabilities of both versions.

I/O capabilities of the Unicode and single-byte versions

- Both the single-byte and Unicode AIMMS versions can read and write to database tables containing either ANSI or Unicode string fields. The single-byte AIMMS version will only accept Unicode string data, as long as the ODBC or OLE DB driver manager is able to convert the Unicode string to a single-byte ASCII string.
- The internal text editor built into the AIMMS Unicode version will accept both Unicode and single-byte ASCII text files. Upon saving, AIMMS will only save the file as a Unicode file if it actually contains non-ASCII characters. The text editor of the single-byte AIMMS version only accepts ASCII text files.
- The `DEVICE` attribute of the `FILE` declaration (see also Section 29.1 of the Language Reference) has been extended with the additional devices `disk(ASCII)` and `disk(Unicode)` besides the existing `disk` device. The following rules apply for `PUT`, `DISPLAY` and `WRITE` statements that refer to the `FILE` identifier in the AIMMS Unicode version:
 - if the device is `disk(ASCII)`, AIMMS will always create an ASCII output file, and complain if some of the output contains non-ASCII characters,
 - if the device is `disk(Unicode)`, AIMMS will always create a Unicode output file, regardless whether the output actually contains non-ASCII characters,
 - if the device is `disk`, AIMMS will create an ASCII output file if the option `default_output_character_set` assumes the value `ascii` and a Unicode output file if the option assumes the value `unicode`. By default, the option assumes the value `automatic`, in which case ASCII output will be created with the single-byte AIMMS version and Unicode output with the AIMMS Unicode version.

The single-byte AIMMS version will always create ASCII output files, regardless of the settings of the `DEVICE` attribute.

- The `READ` statement in the AIMMS Unicode version will accept both ASCII

and Unicode data files. The READ statement in the single-byte AIMMS version only accepts ASCII data files.

- A WRITE statement in the AIMMS Unicode version to a data file which is not indicated through a FILE identifier, will create an ASCII or Unicode output file depending on the option `default_output_character_set`. The WRITE statement in the single-byte AIMMS version always creates an ASCII data file.
- The listing file is created by the AIMMS Unicode version as either an ASCII or a Unicode file depending on the option `listing_file_character_set`. A Unicode listing file may be necessary, for instance, to display a constraint listing referring to Unicode set elements.

The Unicode-related options can be found in the **AIMMS-Reporting-Unicode support** folder of the **Options** dialog box (see Section 22.1).

When your AIMMS project makes use of functionality provided by functions in external DLLs linked to your project, you can specify whether string arguments are to be passed as ASCII or Unicode character buffers. For every STRING PARAMETER argument of an EXTERNAL PROCEDURE or FUNCTION, you can specify the `ascii` and `unicode` properties in the PROPERTY attribute. The following rules apply.

*Arguments of
external
procedure calls*

- If you specify the `ascii` property, both the single-byte and Unicode AIMMS versions will pass the string argument as an ASCII string buffer (or an array of string buffers for a multi-dimensional string parameter). A runtime error will result, if the string data contains genuine Unicode characters.
- If you specify the `unicode` property, both the single-byte and Unicode AIMMS versions will pass the string argument as an Unicode string buffer (or an array of such buffers).

Any string argument in a function of the AIMMS API, expects a Unicode character buffer in the AIMMS Unicode version, and an ASCII character buffer in the single-byte AIMMS version. Also, the data type `AimmsValue` and `AimmsString` expect either Unicode or ASCII string data, depending on the AIMMS version. In both cases, the `.Length` field of the data type refers to the length of the string in the appropriate character set, rather than the byte length of the supplied buffers. Therefore, if your executables and DLLs make use of the AIMMS API, you should make sure that the types of all string arguments passed to AIMMS through the API are of the appropriate type, depending on the AIMMS DLL version you are running your AIMMS project with.

*Arguments of
the AIMMS API*