AIMMS

User's Guide

AIMMS

Dec 11, 2023

CONTENTS

CHAPTER

ONE

AIMMS BASICS

1.1 Creating a new project

Project components

Every AIMMS application consists of two main components:

- an AIMMS project file (with a `.aimms` extension), which contains references to the main application project and all library projects contained in your application,
- for the main project and every library project all source files are stored in a separate folder:
  - the `Project.xml` file holding a reference to the project's main model source file (with an `.ams` extension), as well as all additional model source files included in the main model source file, together containing all identifier declarations, procedures and functions that are relevant to the project,
  - `PageManager.xml`, `TemplateManager.xml` and `MenuBuilder.xml` files describing the page, template and menu tree defined in the project, with all individual pages and templates being stored in the `Pages` and `Templates` subfolder,
  - `Settings` and `Tools` subfolders containing the options for the execution engine, and the saved settings for user colors, fonts, and the various tools in the AIMMS IDE, and
  - the `User Files` folder for storing all user files that you store within the project.

Creating a new project

Within an AIMMS session, you can create a new project through the **File-New Project** menu. Note that this menu is only available when no other project is currently open. It will open the AIMMS **New Project** wizard illustrated in Fig. 1.1.

In this wizard you can enter the name of the new project, along with the directory in which the project is to be stored, and the model file (with the `.ams` extension) to be associated with the project.

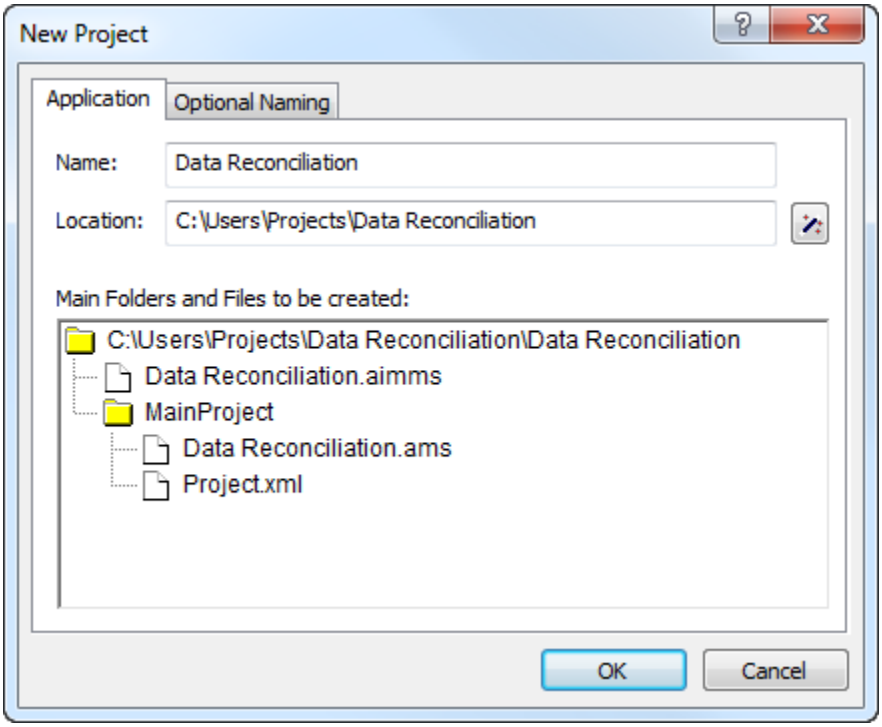


Fig. 1.1: The AIMMS **New Project** wizard

Project directory

By default, the AIMMS **New Project** wizard suggests that the new project be created in a new subdirectory with the same name as the project itself. You can use the wizard button to the right of the location field to modify the location in which the new project is created. However, as AIMMS creates a number of additional files and directories in executing a project, you are strongly advised to store each AIMMS project in a separate directory.

Model file

By default, the AIMMS **New Project** wizard assumes that you want to create a new model file with the same name as the project file (but with a different extension). You can modify the name suggested by the wizard to another existing or nonexisting model file. If the model associated with the project does not yet exist, it will be automatically created by AIMMS.

The Model Explorer

After you have finished with the **New Project** wizard, AIMMS will open the **Model Explorer**, an example of which is illustrated in Fig. 1.2.

The Model Explorer is the main tool in AIMMS to build an AIMMS model, the starting point of building any AIMMS application. In the Model Explorer, the model is presented as a tree of identifier declarations, allowing you to organize your model in a logical manner and make it easy-both for you and others who have to inspect your model-to find their way around. Besides the Model Explorer, AIMMS provides a number of other development tools for model building, GUI building and data management. An overview of these tools is given in *Modeling tools* (page 4).

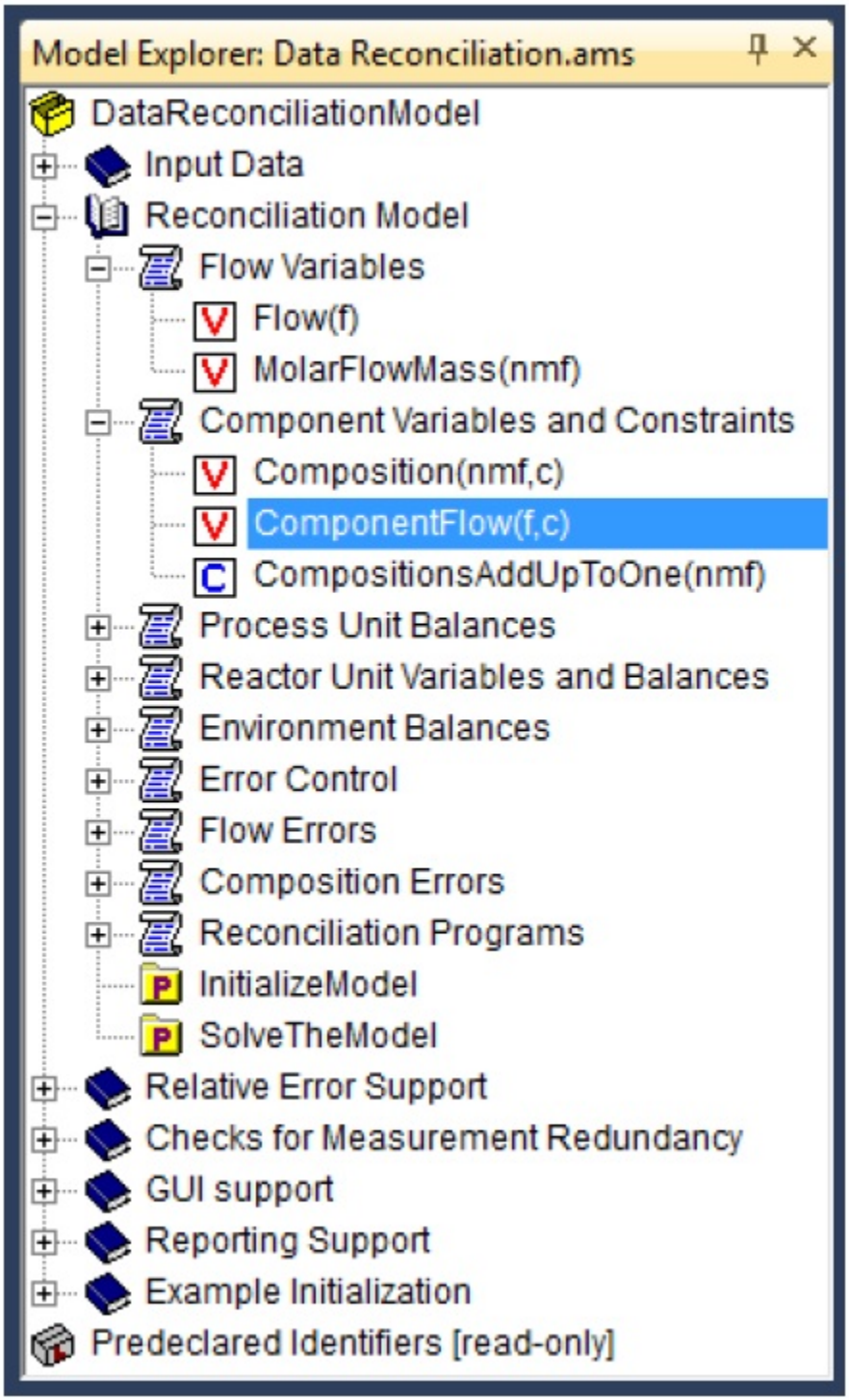


Fig. 1.2: The AIMMS Model Explorer

Starting an existing project

You can open an existing AIMMS project in two ways. You can either

- start AIMMS and open the project via the **File-Open Project** menu, or
- double click on the AIMMS project file (with a `.aimms` extension) in Windows Explorer.

After opening a project, AIMMS may take further actions (such as automatically opening pages or executing procedures) according to the previously stored project settings.

1.2 Modeling tools

Modeling tools

Once you have created a new project and associated a model file with it, AIMMS offers a number of graphical tree-based tools to help you further develop the model and its associated end-user interface. The available tools are:

- the *Model Explorer*,
- the *Identifier Selector*,
- the *Page Manager*,
- the *Template Manager*, and
- the *Menu Builder* tool.

These tools can be accessed either through the **Tools** menu or via the project toolbar. They are all aimed at reducing the amount of work involved in developing, modifying and maintaining particular aspects of your model-based end-user application. Fig. 1.3 provides an overview of the windows associated with each of these tools.

The Model Explorer

The AIMMS **Model Explorer** provides you with a simple graphical representation of all the identifiers, procedures and functions in your model. All relevant information is stored in the form of a tree, which can be subdivided into named sections to store pieces of similar information in a directory-like structure. The leaf nodes of the tree contain the actual declarations and the procedure and function bodies that make up the core of your modeling application. The **Model Explorer** is discussed in full detail in *The Model Explorer* (page 21).

The Identifier Selector

While the **Model Explorer** is a very convenient tool to organize all the information in your model, the **Identifier Selector** allows you to select and simultaneously view the attributes of groups of identifiers that share certain functional aspects in your model. By mutual comparison of the important attributes, such overviews may help you to further structure and edit the contents of your model, or to discover oversights in a formulation. The **Identifier Selector** is discussed in full detail in *Viewing Identifier Selections* (page 54)

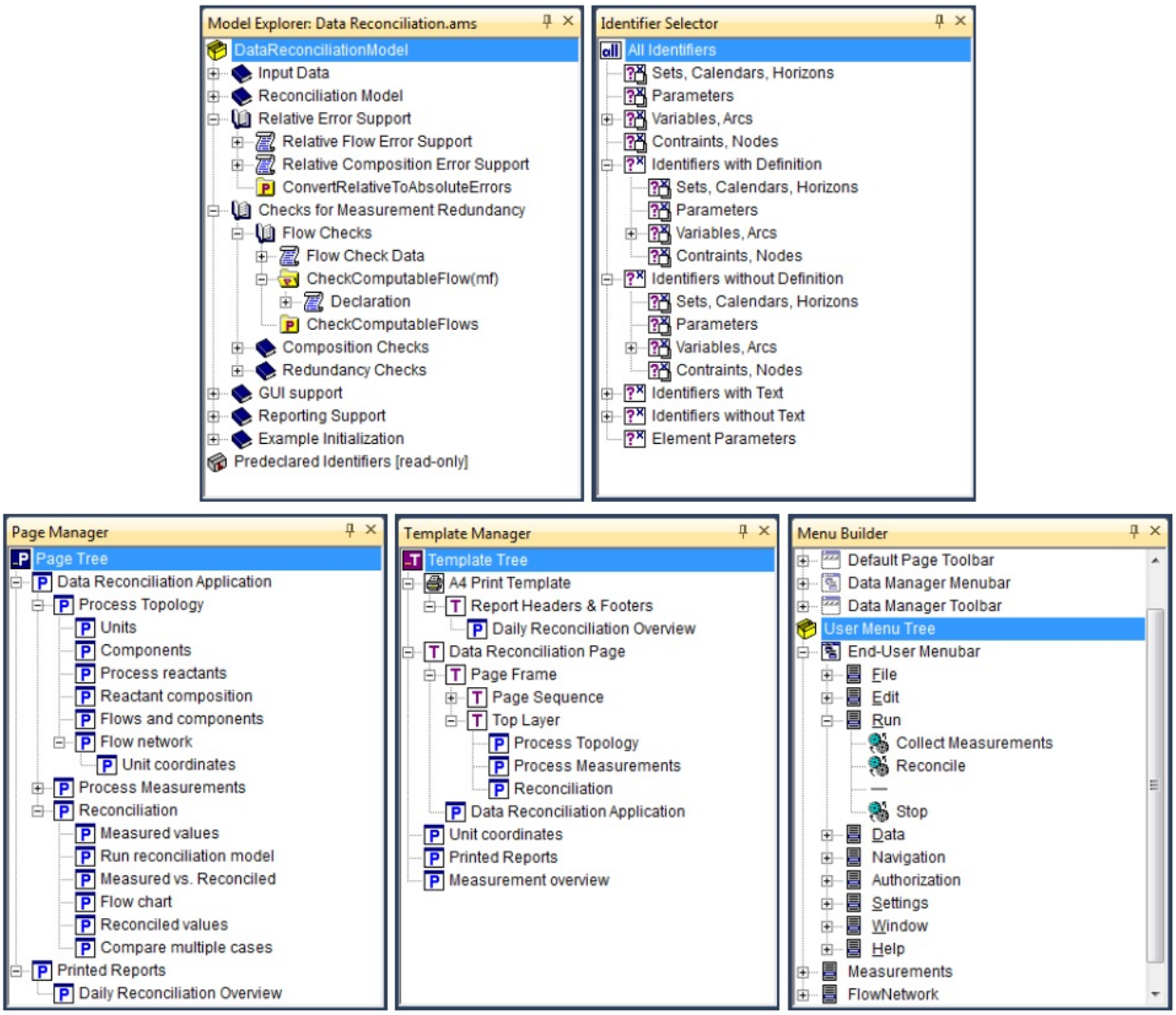


Fig. 1.3: Overview of Aimms tools

The Page Manager

> **Warning:** The AIMMS WinUI and the page manager are deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and the Page Manager.

The **Page Manager** allows you to organize all end-user windows associated with an AIMMS application (also referred to as end-user pages) in a tree-like fashion. The organization of pages in the page tree directly defines the navigational structure of the end-user interface. Relative to a particular page in the page tree, the positions of the other pages define common relationships such as *parent* page, *child* page, *next* page or *previous* page, which can used in navigational controls such as buttons and menus.

The Template Manager

> **Warning:** The AIMMS WinUI and the page manager are deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead.

Within the **Template Manager**, you can make sure that all end-user pages have the same size and possess the same look and feel. You can accomplish this by creating page templates which define the page properties and objects common to a group of end-user pages, and by subsequently placing all end-user pages into the tree of page templates.

The Menu Builder

> **Warning:** The AIMMS WinUI and the page manager are deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and the Page Menu.

With the **Menu Builder** you can create customized menu bars, pop-up menus and toolbars that can be linked to either template pages or end-user pages in your application. In the menu builder window you can define menus and toolbars in a tree-like structure similar to the other page-related tools, to indicate the hierarchical ordering of menus, submenus and menu items.

1.3 Dockable windows

Support for dockable windows

Dockable windows are an ideal means to keep frequently used tool windows in an development environment permanently visible, and are a common part of modern **I**ntegrated **D**evelopment **E**nvironments (IDE) such as *Visual Studio .NET*.

Docking states

Dockable windows can be used in a *docked*, *auto-hidden*, or *floating* state. Whether a dockable window is in a docked, auto-hidden or floating state can be changed at runtime through drag-and-drop.

Docked windows

When docked, the tool windows are attached to the left, right, top or bottom edge of the client area of the main AIMMS window. By default, all modeling tools discussed in *Modeling tools* (page 4) are docked at the left edge of the AIMMS window, as illustrated in Fig. 1.4.

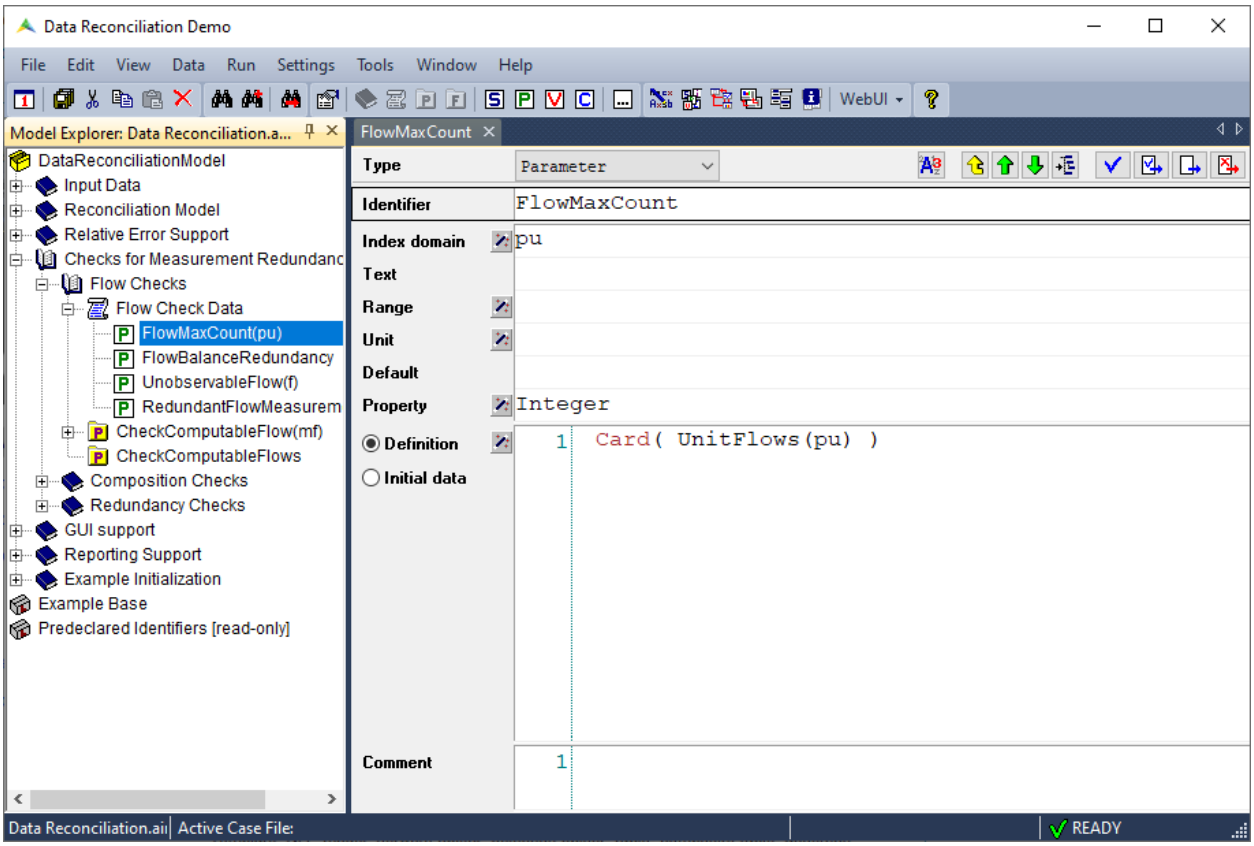


Fig. 1.4: Example of a **Model Explorer** docked at the left edge

Tool windows persistence

AIMMS automatically stores the location and size of each tool window. This information is used to restore the location and size of each tool window whenever an AIMMS session is started.

Dragging windows

By dragging the windows caption of a docked window and moving the cursor around the edges of the AIMMS window, you can move the docked window to another position. While hovering over a drop target, a blue rectangle (as illustrated in Figure Fig. 1.5) snaps into place at the appropriate location, whenever a dockable window is ready to be docked at the location corresponding to the drop target. The area of a docked window can also be split into two by dragging another dockable window into the upper, lower, left or right part of docked window. In all these cases, a blue rectangle shows how a dockable window will be docked when you release the mouse at that time.

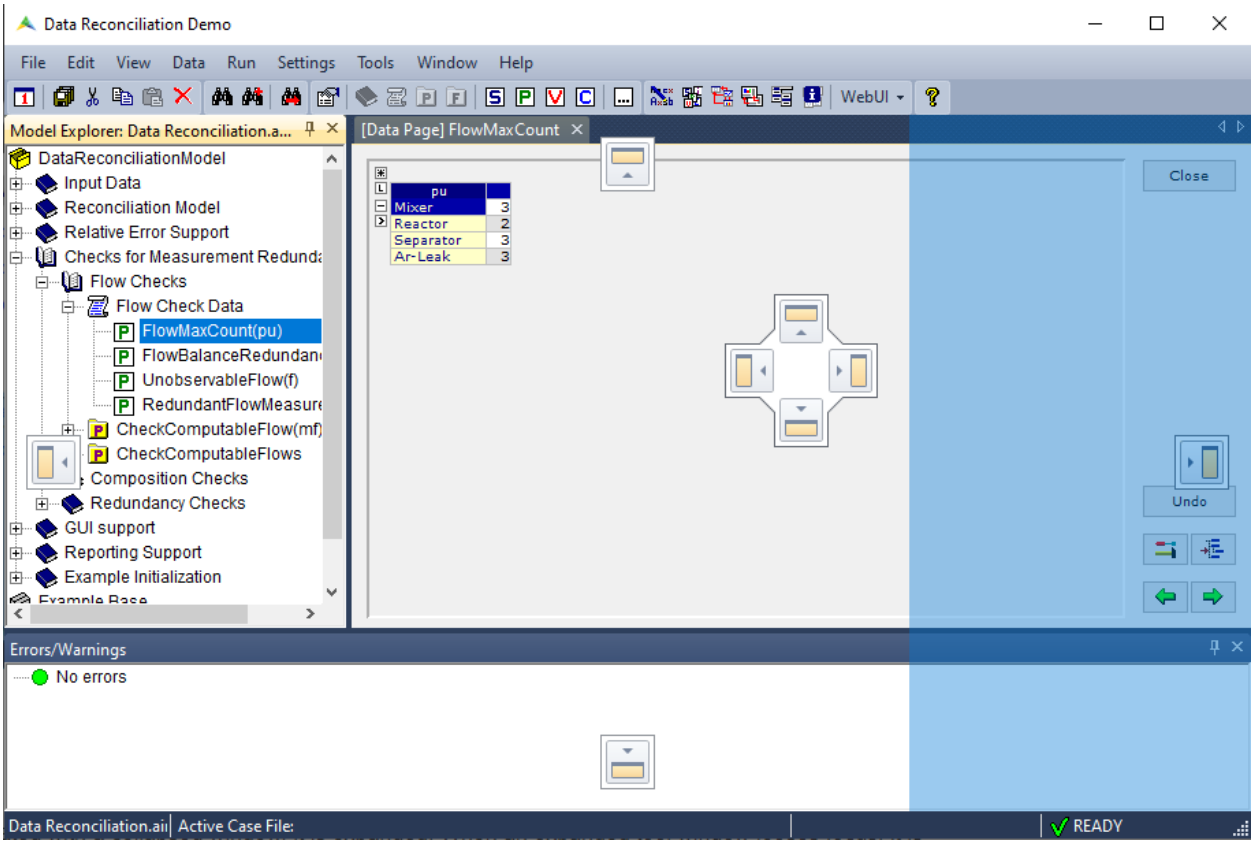


Fig. 1.5: Drag-and-drop windows

Auto-hidden windows

In auto-hidden state, a dockable window is normally collapsed as a button to the upper, lower, left or right edge of the main AIMMS window. When you push the button associated with a collapsed window, it is expanded. When an expanded tool window looses focus, it is collapsed again. By clicking the pushpin button  in the caption of a docked/collapsed window, you can change the window's state from docked to auto-hide and back.

Floating tool windows

By dragging a tool window away from an edge of the main AIMMS window, it becomes floating. When AIMMS is the active application, floating tool windows are always visible on top of all other (non-floating) AIMMS windows. Floating windows can also be shown outside the main AIMMS window frame.

Tabbed MDI mode

By default, all pages and attribute forms are shown in *tabbed MDI* mode. In Fig. 1.4 the main page of the application is displayed in tabbed MDI mode, with the attribute windows of two identifiers in the model accessible through tabs. Tabbed MDI windows occupy the remaining space of the client area of the main AIMMS window that is not occupied by docked windows. This implies that you do not have control over the size of tabbed MDI windows. Therefore, if you use tabbed MDI mode in your AIMMS application, it makes sense to make all the pages in your model resizable. However, the display mode of a page can be changed to docked (by checking the **Allow User Dockable** option on the **Page Properties** dialog box). This would for example allow you to turn a page into a floating window and display it on you second monitor.

Tab groups

When you drag the tab associated with a document window in the document window area, you can move the document window into a new or existing *tab group*, at the left, right, top or bottom of the current tab group. Tab groups effectively split the document window area into multiple parts, each of which can hold one ore more tabbed MDI windows. As with dragging dockable windows, a drag rectangle shows where the window will be positioned if you drop it at that moment. Tab groups are very convenient, for instance, if you want to view two attribute windows simultaneously.

1.4 Additional files related to an AIMMS project

Project-related files

In addition to the AIMMS project folders and files associated discussed in *Creating a new project* (page 1), using an AIMMS project either during development or in a deployment scenario may actually result in the creation of a number of files not mentioned before:

- the name change file (with a `.nch` extension),
- one or more case files (with a `.data` extension),
- a user database file (with a `.usr` extension),
- data backup files (with a `.bak` extension),
- log, error and listing files from both AIMMS and its solvers (with `.log`, `.err`, `.lis` or `.sta` extensions).

Name change file

AIMMS has the capability to keep track of the name changes you performed on identifiers in your model, and automatically replace an old identifier name by its new one whenever AIMMS encounters a renamed identifier. AIMMS keeps track of the list of such name changes in the name change file (with a `.nch` extension). Each name change is listed in this file on a separate line containing the old name, the new name and the (GMT) time stamp at which the change was actually performed. The automatic name change capabilities of AIMMS are explained in full detail in *Committing attribute changes* (page 43)

... and version control

If you are using a version control system to manage your AIMMS sources, it makes sense to also include the name change files under version control. When you change an identifier name, AIMMS will not directly refactor all pages to reflect the name change directly, but use the name change file to refactor a page when it is opened. The same is true when opening cases that contain data for the identifier the name of which has been changed. When your changes in a project are merged with another developer's changes, the merged name change file will actually contain all name changes made by both developers.

Case files

Whenever you save a case in your AIMMS project (see also *Case Management* (page 107)), this will result in the creation of a `.data` file on disk. By default these case files will be stored in the `data` subfolder of project's main folder.

Log, error and listing files

During the execution of your model, all log, error and listing information from both AIMMS and its solvers (whether visible in the AIMMS **Message** window or not) is copied to log, error and listing files, which, by default, are stored in the `Log` subdirectory of the project directory. If you are not interested in this information, you can reduce the amount of information that is copied to these log files by modifying the relevant execution options.

Data backups

Through the **AutoSave & Backups-Data** menu, you can specify that you want AIMMS to automatically create backups of the data used during a particular session of your project. The menu will pop up the **Data Backup** dialog box illustrated in Fig. 1.6.

Similarly as with the project backup files, you can indicate whether AIMMS should automatically create backup backup files of the session data at regular intervals, as well as how many data backup files should be retained. Data backup files also have the `.bak` extension and contain a reference to the date/time of the backup.

Manually creating backup files

Besides the automated backup scheme built into AIMMS, you can also create backup files of your session data manually. You can create manual backup files through the **File-Data Backups** menu. When you create a data backup file manually, AIMMS will request a name of a `.bak` file in which the backup is to be stored.

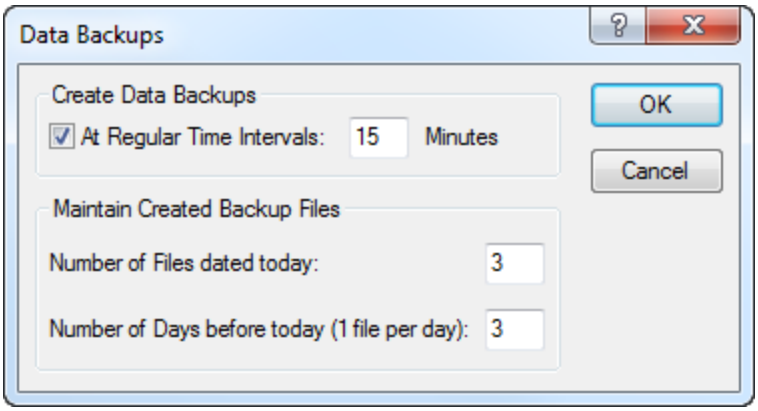


Fig. 1.6: The **Data Backup** dialog box

Restoring backup files

Through the **File-Data Backups** menu, you can restore the data in your application back to the state stored in the data backup files.

1.4.1 Project User Files

Warning: The AIMMS WinUI and Project User Files are deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead, the User Management and Create PRO User Groups

Project user files

Along with the project-related files created by AIMMS, you may need to distribute some other files with your project when deploying it to your end-users. Such files include, for instance, bitmap files displayed on buttons or in the background of your end-user pages, or files that contain project-related configuration data. Instead of having to include such files as separate files in the project directory, AIMMS also allows you to save them within the project file itself. Both within the AIMMS language as well as in the end-user interface, you can reference such *project user files* as if they were ordinary files on disk.

Why use project user files?

User project files are convenient in a number of situations. The most common reasons to store files as project user files are listed below.

- You want to reduce the number files that you have to ship to your end users. This situation commonly occurs, for instance, when the end-user interface of your project references a large number of bitmap files.
- You want to hide particular configuration data files from your end-users, which might otherwise only confuse them.
- User project cannot be modified by your end-users.

Importing project user files

You can import files into the project file through the **Tools-Project User Files** menu, which will pop up the **Project User Files** dialog box illustrated in Fig. 1.7.

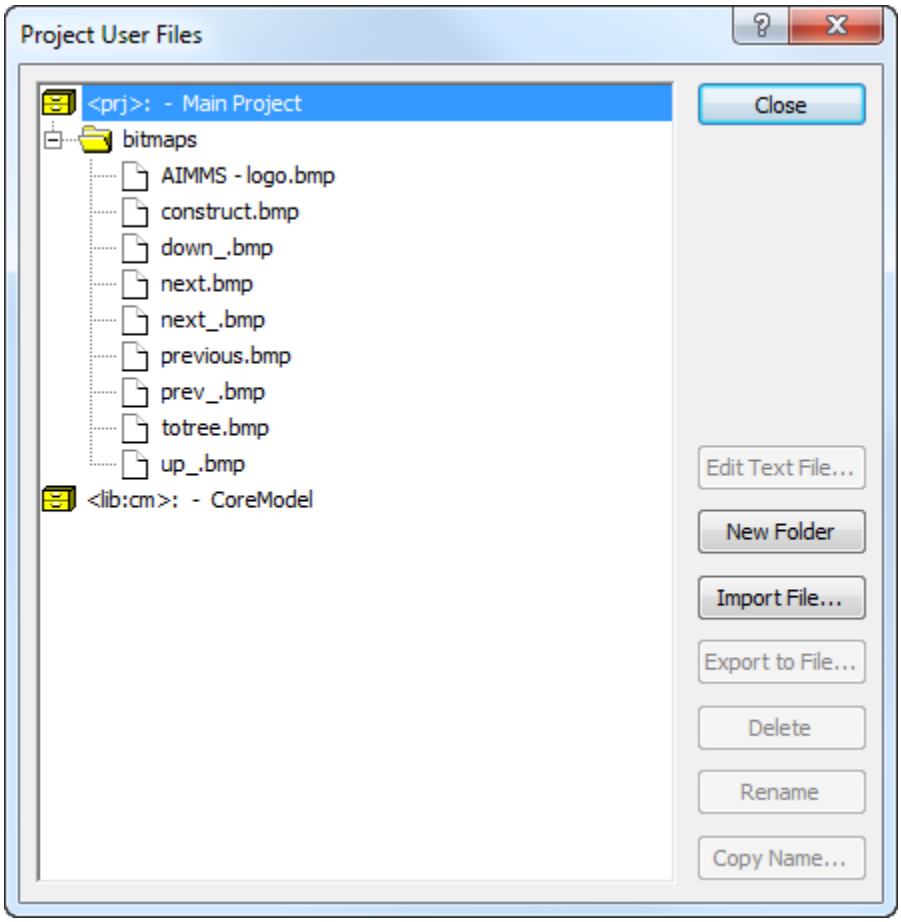


Fig. 1.7: The **Project User Files** dialog box

In this dialog box, you can create new folders to organize the files you want to import into the project file. The dialog box of Fig. 1.7 already contains a folder **bitmaps**, which is automatically added to each new AIMMS project and filled by AIMMS with the bitmaps used on AIMMS' data pages (see *Viewing and modifying identifier data* (page 45)). When you are inside a folder (or just within the main project file), you can import a file into it through the **Import File** button, which will open an ordinary file selection dialog box to select the disk file to be imported.

User files in library projects

When your project, next to the main project file, also includes a number of library project files (see *Library projects and the library manager* (page 16)), AIMMS allows you to store user files in the library project files as well. Thus, if a page defined in a library refers to a particular bitmap file, you can also store that bitmap as a user file directly into the corresponding library project file. In the dialog box of Fig. 1.7, the *CoreModel* node at the root of the tree refers to a library that is included in the project that serves as the running example throughout this book. Underneath this node you can add user files that will be stored in the library project file for the *CoreModel* library.

Referencing project user files

You can reference project user files both from within the AIMMS language and the properties of various objects with the graphical end-user interface. The basic rule is that AIMMS considers the project file as a virtual disk indicated by "<prj>". You can use this virtual drive in, for instance, READ, WRITE and PUT statements within your model. Thus, the statement

```
READ from file "<prj>:config\\english.dat";
```

reads the model data from the project user file "english.dat" contained in a (developer-created) **config** folder within the project file.

Referencing user files in library projects

You can access project files in library projects by using the virtual disk notation "<lib:*library-name*>", where *library-name* is the name of the library project. Thus, to read the same file as in the previous paragraph from the *CoreModel* library shown in Fig. 1.7, the following statement can be used.

```
READ from file "<lib:CoreModel>:config\\english.dat";
```

Use in end-user interface

Similarly, you can reference project user files on page objects in the end-user interface of your project. Fig. 1.8 illustrates the use of a bitmap file stored in the project file on a bitmap button.

For all object properties expecting a file name (such as the **File Name** property of the bitmap button illustrated in Fig. 1.8), you can easily select a project user file by pressing the wizard button  , and selecting the **Select Project File** menu item. This will pop up a project user file selection dialog box similar to the dialog box shown in Fig. 1.7.

1.5 Collaborative Project Development

This chapter discusses the options you have in AIMMS to organize a project in such a way that it becomes possible to effectively work on the project with multiple developers. While it is very natural to start working on a project with a single developer, at some time during the development of an AIMMS application, the operational requirements of the problem you are modeling may become so demanding that it requires multiple developers to accomplish the task.

From prototyping phase...

During the initial prototyping phase of a model, an AIMMS project is usually still quite small, allowing a single developer to take care of the complete development of the prototype. The productivity tools of AIMMS allow you to quickly implement different formulations and analyze their results, while you are avoiding the overhead of having to synchronize the efforts of multiple people working on the prototype.

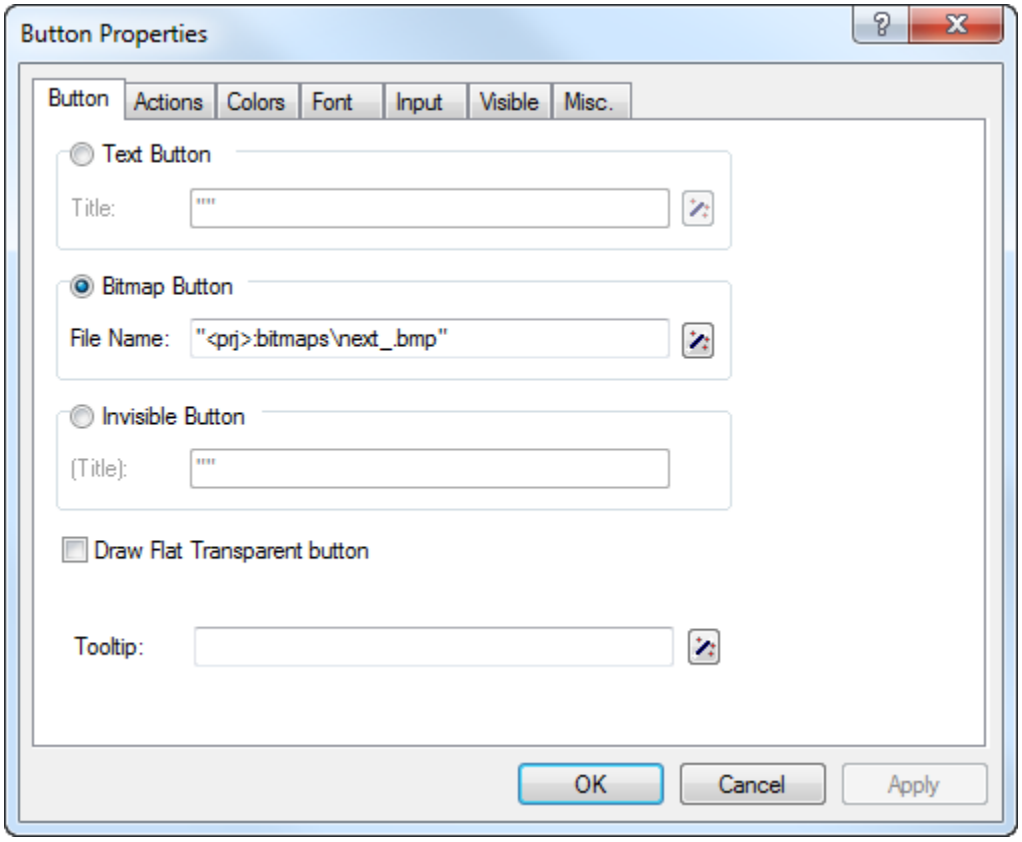


Fig. 1.8: Bitmap button referencing a project user file

. . . to operational phase

During the development of an operational AIMMS application this situation may change drastically. When an AIMMS application is intended to be used on a daily basis, it usually involves one or more of the following tasks:

- retrieving the input data from one or more data sources,
- validation and transformation of input data,
- extending the core optimization application with various, computationally possibly demanding, operational requirements,
- preparing and writing output data to one or more data sources,
- building a professionally looking end-user GUI, and/or
- integrating the application into the existing business environment.

Depending on the size of the application, implementing all of these tasks may become too demanding for a single developer.

Dividing a project into sub-projects

One possible approach to allow multiple developers to work on a single AIMMS application is to divide the project into several logical sub-projects, either based on the tasks listed in the previous paragraph, or more closely related to the logic of your application. AIMMS supports sub-projects in the form of model libraries. Using libraries especially makes sense, if the functionality developed in a library can be re-used by multiple AIMMS applications. If a library is small enough, indivual developers may take on the development of the library.

Managing project source using a VCS

In the software development world teams commonly use a *V*ersion *C*ontrol *S*ystem, such as git, subversion, or TFS, to share and merge their coding work to a common repository. As all development sources of an AIMMS application are stored as readable text files (a.o. `.aimms`, `.ams` and `.xml`), AIMMS projects can be easily managed using the version control system of your choice. Using a version control system will make it straightforward to work together on a single code base in parallel by merging the contributions of the various team members, and to use branches to differentiate between development and production code, or to work on multiple developments independently. Using version control for your AIMMS projects will usually result in higher productivity and more control.

No VCS integration

Although AIMMS effectively supports the use of a version control system for the development of your AIMMS applications, the AIMMS IDE does not offer integration with any specific version control system. All version control systems come with commandline and/or graphical tools for regular version control tasks such as committing, showing logs, diffing two versions, merging, creating branches and tags, and so on. You should use these tools, whenever you want to commit your changes to an AIMMS project under version control.

1.5.1 Library projects and the library manager

AIMMS library projects

AIMMS *library projects* allow you to divide a large AIMMS project into a number of smaller sub-projects. Library projects are especially useful if you intend to share parts of an application between multiple projects. Each library project in AIMMS provides

- a tree of model declarations,
- a page tree,
- a template tree, and
- a menu tree.

In addition, a library project may provide its own collection of user project files, user colors and fonts.

Shared templates

Besides enabling multiple developers to work in a single project, library projects can also be used to define a common collection of templates that define the look-and-feel of multiple projects. In this manner, you change the look-and-feel of multiple applications just by changing the templates in the shared library project.

> **Warning:** The AIMMS WinUI is deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and share the WebUI JSON with your fellow developers

Adding libraries to a project

By adding a library project to the main AIMMS project, the objects defined by the library, such as identifiers, pages, templates, etc., become available for use in the main project. In addition, if a library project is writable, the contents of the library can also be edited through an AIMMS project in which it is included.

The library manager

You can add libraries to an AIMMS project in the **AIMMS Library Manager** dialog box illustrated in Fig. 1.9. You can open the library manager through the **File-Library Manager...** menu.

Using the library manager AIMMS allows you to

- create new libraries,
- add existing libraries to your project,
- add system libraries (contained within the AIMMS version installed on your computer)
- add a library from the Library Repository

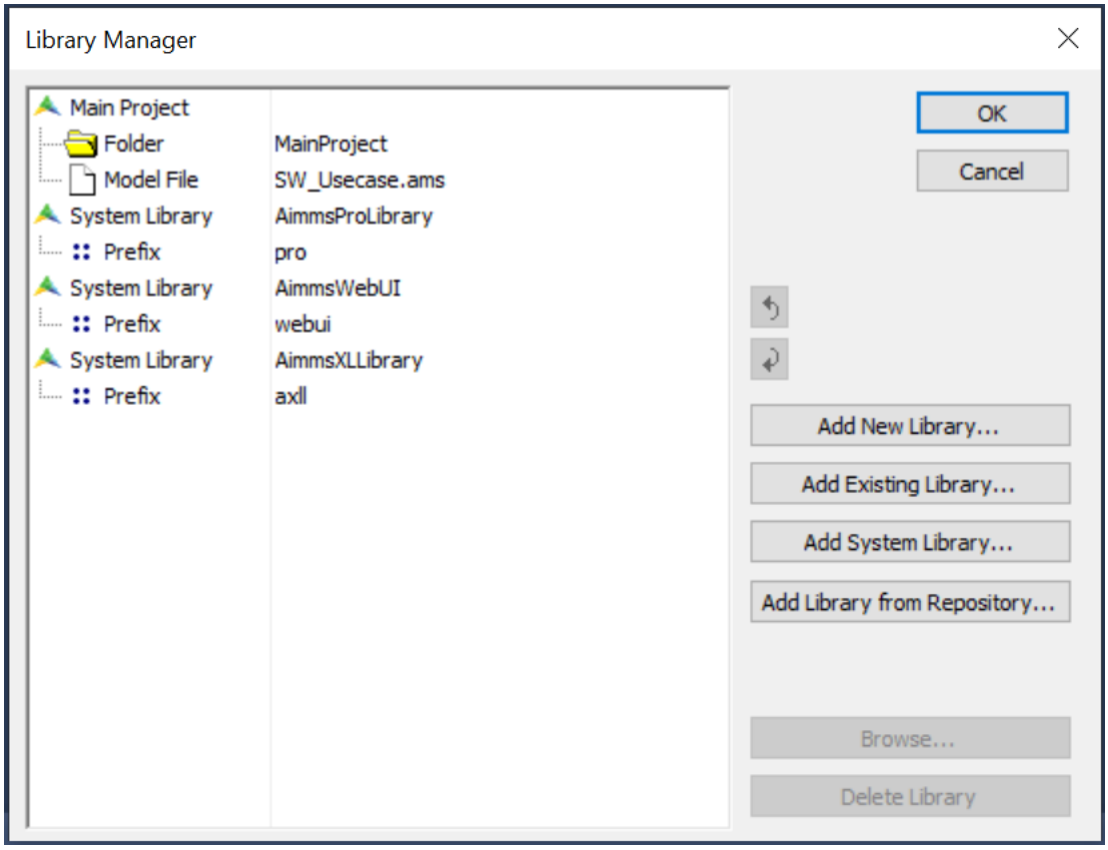


Fig. 1.9: The **AIMMS Library Manager**

Library storage

Each library project in AIMMS will be stored in a separate directory, containing the following files and folders:

- the `Project.xml` file holding a reference to the project's main model source file (with an `.ams` extension), as well as all additional model source files included in the main model source file, together containing all identifier declarations, procedures and functions that are relevant to the project,
- `PageManager.xml`, `TemplateManager.xml` and `MenuBuilder.xml` files describing the page, template and menu tree defined in the project, with all individual pages and templates being stored in the `Pages` and `Templates` subfolders,
- `Settings` and `Tools` subfolders containing the saved settings for user colors, fonts, and the various tools in the AIMMS IDE, and
- the `User Files` folder for storing all user files that you store within the project.

These files will be automatically created by AIMMS when you create a new library project. To add an existing library to an AIMMS project, you just need to select its library project file.

Library prefix

To avoid name clashes between objects in the library and the main project or other libraries, all the object names in a library are stored in a separate namespace. Outside of the library, a global prefix associated with the library has to be used to access the library objects. When you create a new library project, AIMMS will come up with a default library prefix based on the library name you entered. For an existing library project, you can view and edit its associated library prefix in the library manager.

Using library projects

After you have added one or more library projects to your main AIMMS project, AIMMS will extend

- the model tree in the **Model Explorer**,
- the page tree in the **Page Manager**,
- the template tree in the **Template Manager**, and
- the menu tree in the **Menu Builder**

with additional root nodes for every library project added to your project. In general, within any of these tools, you are free to move information from the main project tree to any of the library trees and vice versa. In addition, the AIMMS dialog boxes for user project files, user colors and fonts allow you to select and manage objects from the main project or any of the libraries. The precise details for working with library projects in each of these tools are discussed in full detail in the respective chapters discussing each of the tools.

1.5.2 Guidelines for working with library projects

Identifying independent tasks

Unless you started using library projects from scratch, you need to convert the contents of your AIMMS project as soon as you decide to divide the project into multiple library projects. The first step in this process is to decide which logical tasks in your application you can discern that meet the following criteria:

- the task represents a logical unit within your application that is rather self-contained and can possible be shared with other AIMMS projects,

- the task can be comfortably and independently worked on by a separate developer or developer team, and
- the task provides a limited interface to the main application and/or the other tasks you have identified.

Good examples of generic tasks that meet these criteria are the tasks listed on *Collaborative Project Development* (page 13). Once your team of developers agrees on the specific tasks that are relevant for your application, you can set up a library project for each of them.

Library interface...

The idea behind library projects is to be able to minimize the interaction between the library, the main project and other library projects. At the language level AIMMS supports this by letting you define an *interface* to the library, i.e. the set of public identifiers and procedures through which the outside world is allowed to connect to the library. Library identifiers not in the interface are strictly private and cannot be referenced outside of the library. The precise semantics of the interface of a library module is discussed in LibraryModule Declaration and Attributes of the Language Reference.

... used in model and GUI

This notion of public and private identifiers of a library module does not only apply to the model source itself, but also propagates to the end-user interface. Pages defined in a library can access the library's private identifiers, while paged defined outside of the library only have access to identifiers in the interface of the library.

Minimal dependency

The concept of an interface allows you to work independently on a library. As long as you do not change the declaration of the identifiers and procedures in its interface, you have complete freedom to change their implementation without disturbing any other project that uses identifiers from the library interface. Similarly, as long as a page or a tree of pages defined in a library project is internally consistent, any other project can add a reference to such pages in its own page tree. Pages outside of the library can only refer to identifiers in the library interface, and hence are not influenced by changes you make to the library's internal implementation.

Conversion to library projects

If your application already contains model source and pages associated with the tasks you have identified in the previous step, the next step is to move the relevant parts of your AIMMS project to the appropriate libraries. You can accomplish this by simply dragging the relevant nodes or subtrees from any of the trees tree in the main project to associate tree in a library project. What should remain in the global project are the those parts of the application that define the overall behavior of your application and that glue together the functionality provided by the separate library projects.

CHAPTER

TWO

CREATING AND MANAGING A MODEL

2.1 The Model Explorer

This chapter introduces the interactive **Model Explorer** that is part of the AIMMS system. With the **Model Explorer** you have easy access to every component of the source of your model. In this chapter, you are introduced to the model tree, and you are shown which model information can be added to the model tree. In addition, the basic principles of working with the **Model Explorer** are explained.

2.1.1 What is the Model Explorer?

Support for large models

Decision making commonly requires access to massive amounts of information on which to base the decision making process. As a result, professional decision support systems are usually very complex programs with hundreds of (indexed) identifiers to store all the data that are relevant to the decision making process. In such systems, finding your way through the source code is therefore a cumbersome task. To support you in this process, AIMMS makes all model declarations and procedures available in a special tool called the **Model Explorer**.

Structured model representation

The AIMMS **Model Explorer** provides you with a simple graphical model representation. All relevant information is stored in the form of a *model tree*, an example of which is shown in Fig. 2.1.

As you can see in this example, AIMMS does not prescribe a fixed declaration order, but leaves it up to you to structure all the information in the model in any way that you find useful.

Different node types

As illustrated in Fig. 2.1, the model tree lets you store information of different types, such as identifier declarations, procedures, functions, and model sections. Each piece of information is stored as a separate node in the model tree, where each node has its own type-dependent icon. In this section, the main node types in the model tree will be briefly introduced. In subsequent chapters, the details of all model-related node types such as identifiers, procedures and functions will be discussed in further detail.

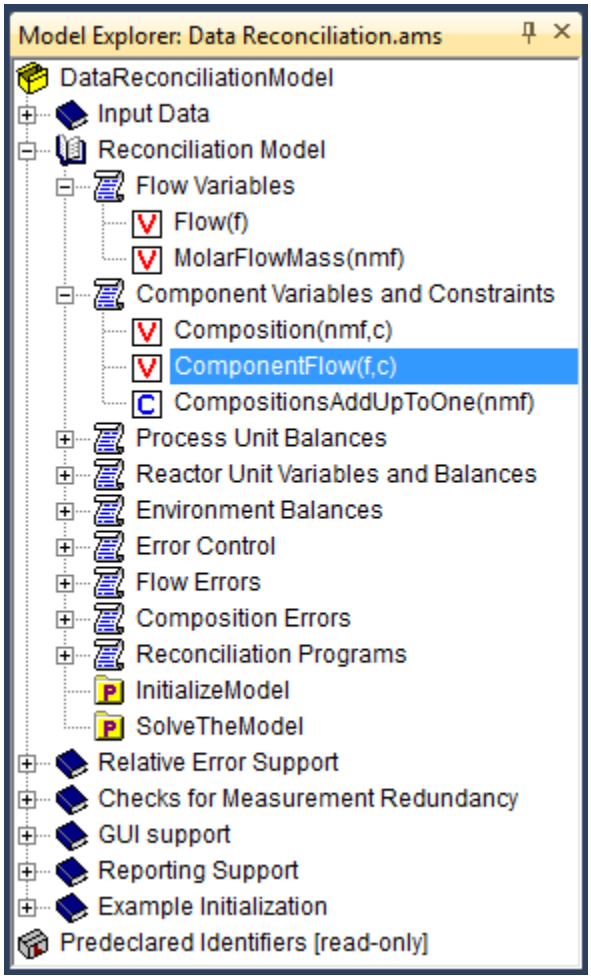


Fig. 2.1: Example of a model tree

Structuring nodes

There are three basic node types available for structuring the model tree. You can branch further from these nodes to provide more depth to the model tree. These basic types are:

- The *main model* node which forms the root of the model tree. The main model is represented by a box icon which opens when the model tree is expanded, and can contain book sections, declaration sections, procedures and functions.
- *Book section* nodes are used to subdivide a model into logical parts with clear and descriptive names. Book sections are represented by a book icon which opens when the section is expanded. A book section can contain other book sections, declaration sections, procedures and functions.
- *Declaration section* nodes are used to group identifier declarations of your model. Declaration sections are represented by a scroll icon, and can only contain identifier declaration nodes.

Advantages

The structuring nodes allow you to subdivide the information in your model into a logical framework of sections with clear and descriptive names. This is one of the major advantages of the AIMMS model tree over a straightforward text model representation, as imposing such a logical subdivision makes it much easier to locate the relevant information when needed later on. This helps to reduce the maintenance cost of AIMMS applications drastically.

Module and library nodes

In addition to the basic structuring nodes discussed above, AIMMS supports two additional structuring node types, which are aimed at re-use of parts of a model and working on a single AIMMS project with multiple developers.

- The *module* node offers the same functionality as a book section, but stores the identifiers it defines in a separate namespace. This allows a module to be included in multiple models without the risk of name clashes. Module nodes are represented by the icon .
- The *library module* node is the source module associated with a library project (see *Library projects and the library manager* (page 16)). Library modules can only be added to or deleted from a model through the **Library Manager**, and are always displayed as a separate root in the model tree. Library module nodes are represented by the icon .

Modules, library modules and the difference between them are discussed in full detail in Model Structure and Modules of the Language Reference.

AIMMS library

For your convenience, AIMMS always includes a single, read-only library module called `Predeclared Identifiers` (displayed in Fig. 2.1), containing all the identifiers that are predeclared by AIMMS, categorized by function.

Non-structuring nodes

All remaining nodes in the tree refer to actual declarations of identifiers, procedures and functions. These nodes form the actual contents of your modeling application, as they represent the set, parameter and variable declarations that are necessary to represent your application, together with the actions that you want to perform on these identifiers.

Identifier nodes

The most frequent type of node in the model tree is the identifier declaration node. All identifiers in your model are visible in the model explorer as leaf nodes in the declaration sections. Identifier declarations are not allowed outside of declaration sections. AIMMS supports several identifier types which are all represented by a different icon. The most common identifier types (i.e. sets, parameters, variables and constraints) can be added to the model tree by pressing one of the buttons  (the last button opens a selection list of all available identifier types). Identifier declarations are explained in full detail in *Identifier Declarations* (page 34).

Independent order

Identifiers can be used independently of the order in which they have been declared in the model tree. As a matter of fact, you may use an identifier in an expression near the beginning of the tree, while its declaration is placed further down the tree. This order independence makes it possible to store identifiers where you think they should be stored logically, which adds to the overall maintainability of your model. This is different from most other systems where the order of identifiers is dictated by the order in which they are used inside the model description.

Procedure and function nodes

Another frequently occurring node type is the declaration of a procedure or a function. Such a procedure or function node contains the data retrieval statements, computations, and algorithms that make up the procedural execution of your modeling application. Procedures and functions are represented by folder icons,  and  , which open when the procedure or function node is expanded. They can be inserted in the model tree in the root node or in any book section. The fine details of procedure and function declarations are explained in *Procedures and Functions* (page 47).

Procedure and function subnodes

Procedures and functions may contain their own declaration sections for their arguments and local identifiers. In addition, a procedure or function can be subdivided into logical components which are inserted into the body of that procedure or function, and are stored as execution subnodes. Such execution subnodes allow you to follow a top-down approach in implementing an algorithm without the need to introduce separate procedures to perform every single step. The complete list of permitted subnodes is discussed in *Procedures and Functions* (page 47).

Attributes

For every node in the model tree you can specify additional information in the form of *attributes*. AIMMS lets you view and change the values of these attributes in an *attribute form* that can be opened for every node in the tree. An example of an attribute form of an identifier node is shown in Fig. 2.2.

Such an attribute form shows all the attributes that are possible for a particular node type. For instance, the attribute form of a parameter declaration will show its domain of definition and value range, while the form for a procedure will show the argument list and procedure body. In the attribute form you can enter values that are relevant for your model.

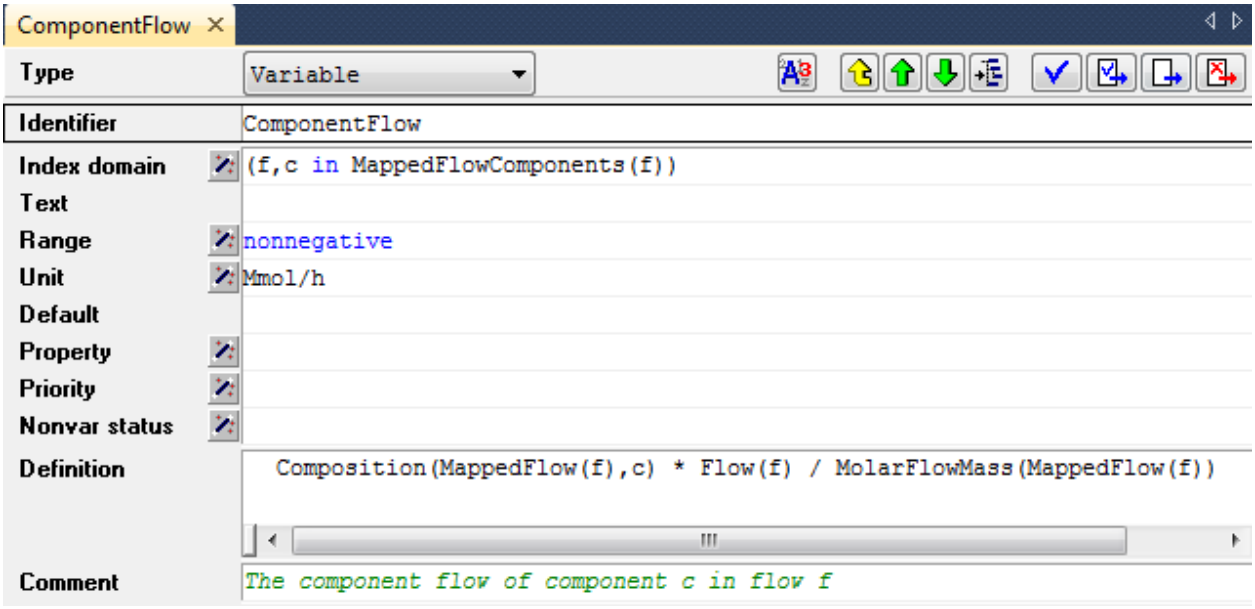


Fig. 2.2: Example of an attribute form

Wizards

For most attributes in an attribute form AIMMS provides wizards which help you complete the attributes with which you are not familiar. Attribute wizards can be invoked by pressing the small buttons in front of the attribute fields as shown in Fig. 2.2. The wizard dialog boxes may range from presenting a fixed selection of properties, to presenting a relevant subselection of data from your model which can be used to complete the attribute.

Reduce syntax knowledge

By providing attribute forms and their associated wizards for the declaration of all identifiers, the amount of syntax knowledge required to set up the model source is drastically reduced. The attribute window of each identifier provides you with a complete overview of all the available attributes for that particular type of identifier. The wizards, in most cases, guide you through one or more dialog boxes in which you can choose from a number of possible options. After selecting the options relevant to your model, AIMMS will subsequently enter these in the attribute form using the correct syntax.

Local compilation

Once your complete model has been compiled successfully, attribute changes to a single identifier usually require only the recompilation of that identifier before the model can be executed again. This local compilation feature of AIMMS allows you to quickly observe the effect of particular attribute changes.

. . . versus global compilation

However, when you make changes to some attributes that have global implications for the rest of your model, local compilation will no longer be sufficient. In such a case, AIMMS will automatically recompile the entire model before you can execute it again. Global recompilation is necessary, for instance, when you change the dimension of a particular identifier. In this case global re- compilation is required, since the identifier could be referenced elsewhere in your model.

Attributes of structuring nodes

The attributes of structuring nodes allow you to specify documentation regarding the contents of that node. You can also provide directives to AIMMS to store a section node and all its offshoots in a separate file which is to be included when the model is compiled. Storing parts of your model in separate model files is discussed in more detail in *Creating and managing models* (page 27).

2.1.2 Creating and managing models

Creating new models

When you begin a new model, AIMMS will automatically create a skeleton model tree suitable for small applications and student assignments. Such a skeleton contains the following nodes:

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

The model tree also displays the predefined and read-only library module `Predeclared Identifiers` (see also *What is the Model Explorer?* (page 21)), which contains all the identifiers predeclared by AIMMS, categorized by function.

Changing the skeleton

Whenever the number of declarations in your model becomes too large to be easily managed within a single declaration section, or whenever you want to divide the execution associated with your application into several procedures, you are free (and advised) to change the skeleton model tree created by AIMMS. You can group particular declarations into separate declaration sections with meaningful names, and introduce your own procedures and functions. You may even decide to remove one or more of the skeleton nodes that are not of use in your application.

Additional structuring of your model

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

Storage on disk

The contents of a model are stored in one or more files with the "`.ams`" (**m**odel **s**ource) extension. By default the entire model is stored as a single file, but for each book section node  or module node  in the tree you can indicate that you want to store the subtree below it in a separate source file. This is especially useful when particular parts of your application are shared with other AIMMS applications, or are developed by other persons. Library modules  associated with a library project that you have included in your project, are always stored in a separate `.ams` file.

Separate storage

To store a module or section of your model in a separate source file, open the attribute form of that section node by double-clicking on it in the model explorer. The attribute form of a section is illustrated in Fig. 2.3.

By selecting the **Write...** command of the `SourceFile` attribute wizard on this form, you can select a file where you want all information under the section node to be stored. AIMMS will export the contents of the book section to the indicated file, and enter that file name in the `SourceFile` attribute of the book section. As a consequence, AIMMS will automatically read the contents of the book section from that file during every subsequent session.

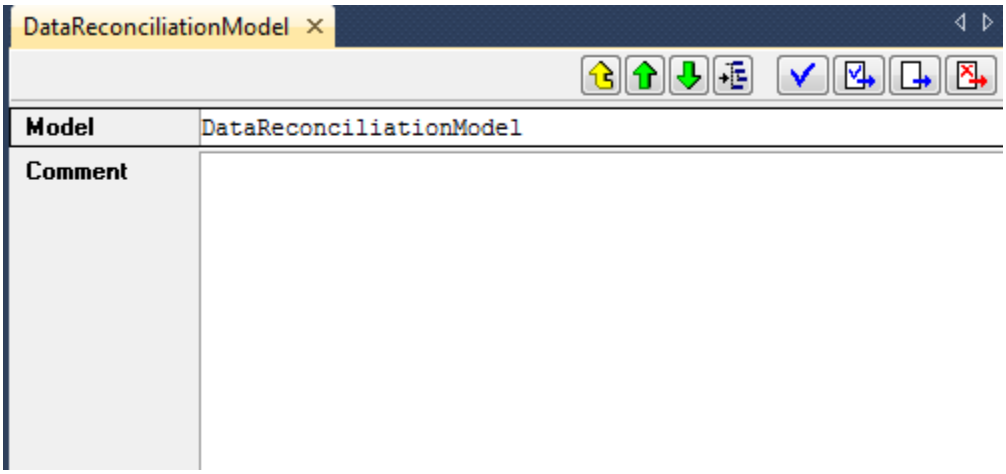


Fig. 2.3: Attribute form of a section node

Exporting a book section

Alternatively, when you are in the **Model Explorer** on the book section node that you want to store in a separate file, you can use the **Edit-Export** menu, to export the contents of the selected section to a separate `.ams` file. In the latter case, AIMMS will only export a *copy* of the contents of the selected section to the specified `.ams` file, while the original contents is still stored in the main `.ams` model file.

Adding a book section reference

Likewise, if you want a book section to hold the contents of a section stored in a separate `.ams` file, you can use the **Read...** command of the **SourceFile** wizard. This will let you select an `.ams` file which will be entered in the `SourceFile` attribute. As a consequence, the contents of this file will be included into the section during this and any subsequent sessions. Note that any previous contents of a section at the time of entering the `SourceFile` attribute will be lost completely. By specifying a `SourceFile` attribute, any changes that you make to the contents of the section after adding a `SourceFile` attribute will be automatically saved in the corresponding `.ams`, whenever you save your model.

Importing a book section

Alternatively, you can import a copy of the contents of a separate `.ams` file into your model, by executing the **Edit-Import** menu command on a selected section node in the **Model Explorer**. This will completely replace the current contents of the section with the contents of the `.ams` file. In this case, however, any changes that you make to the section after importing the `.ams` file will not be stored in that file, but only in your main model file.

Working with Modules and Libraries

Name clashes

When you import the contents of a book section node into your model, you may find that particular identifier names in that book section already have been declared in the remainder of your model. If such a name clash occurs, AIMMS will refuse to import the specified `.ams` file into your model, and present a dialog box indicating which identifiers would cause a name clash when imported.

Avoid name clashes using modules

You can avoid name clashes by using *modules*, which provide their own namespace. Modules allow you to share sections of model source between multiple models, without the risk of running into name clashes. The precise semantics of modules are discussed in full detail in Model Structure and Modules of the Language Reference.

Creating modules

You can create a module anywhere in your model tree by inserting a *Module* node into your tree, as discussed in *Working with trees* (page 31). For each module you must specify a module prefix through which you can access the identifiers stored in the module. Fig. 2.4 illustrates the attributes of a module.

If this module contains a parameter `GlobalSettings`, then outside of the module it can be referenced as `shared::GlobalSettings`.

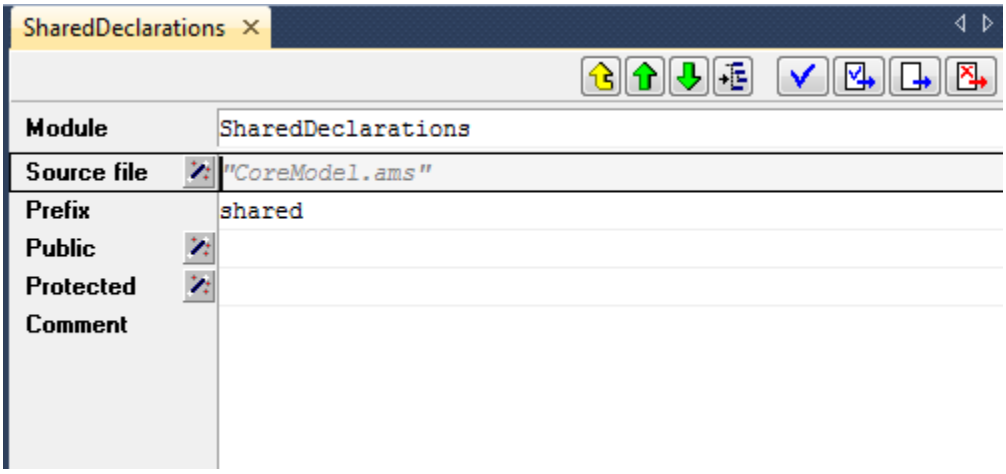


Fig. 2.4: The attributes of a Module node

AIMMS system modules

AIMMS uses modules to implement those parts of its functionality that can be best expressed in the AIMMS language itself. The available AIMMS system modules include

- a (customizable) implementation of the outer approximation algorithm,
- a scenario generation module for stochastic programming, and
- sets of constants used in the graphical 2D- and 3D-chart objects.

You can include these system modules into your model through the **Settings-Install System Module...** menu.

Library projects ...

If your model becomes too large for a single developer to maintain and develop, you may use *library projects* to create a division of your existing project into sub-projects. The procedure for creating such library projects is discussed in *Library projects and the library manager* (page 16). For each library included in your project, AIMMS creates a separate library module node at the root the **Model Explorer**, as illustrated in Fig. 2.5.

When creating a new library the associated library module will initially be empty. In the library module of Fig. 2.5, one section from the original model tree in Fig. 2.1 has already been moved into the newly created library.

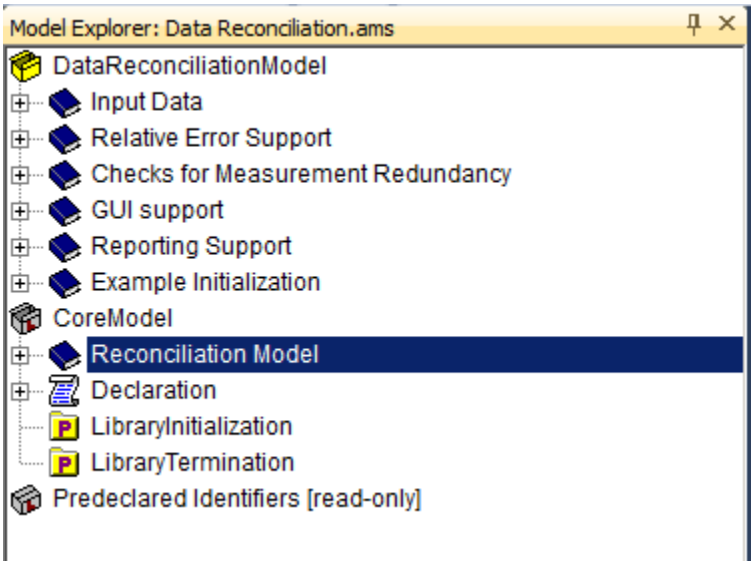


Fig. 2.5: A library module containing the core model formulation

...for modular development

Contrary to modules, whose principle aim is to let you share a common set of identifier and procedure declarations among multiple models, library projects allow you to truly divide an AIMMS project into subprojects. With every library project you cannot only associate a module in the model tree, but AIMMS lets you also develop pages and menus for the graphical user interface within a library project. Within an AIMMS project that includes such a library project, you can use the model, pages and menus to compose the entire application in a modular way.

Moving identifiers to modules and libraries

When you move identifiers from the main model to a module or a library module, references to such identifiers in the main model may become invalid because because they become part of a different namespace. In accordance with the automatic name change support described in *Navigation Features* (page 42), AIMMS will automatically change all references to the identifier in the model source, project pages, and case files to include the module prefix, unless the reference is included in the module or library itself. In occasional situations, however, the automatic name change support of AIMMS may fail to detect such references, for instance, when an identifier name is included in a data initialization statement of a subset of `AllIdentifiers`.

Library initialization and termination

Each library may provide four procedures *LibraryInitialization*, *PostLibraryInitialization*, *PreLibraryTermination* and *LibraryTermination*. If you specify these procedures, they should contain all statements necessary to properly initialize the data associated with a library prior to it first use, and provide the library with a possibility to save its internal state prior to closing a project. The exact initialization and termination sequence of AIMMS models is discussed in Model Initialization and Termination of the Language Reference.

2.1.3 Working with trees

Working with trees

The trees used in the various developer tools inside AIMMS offer very similar functionality to the directory tree in the Windows Explorer. Therefore, if you are used to working with the Windows Explorer, you should have little difficulty understanding the basic functionality offered by the trees in the AIMMS tools. For novice users, as well as for advanced users who want to understand the differences to the Windows Explorer, this section explains the fine details of working with trees in AIMMS, using the context of the model tree.

Expanding and collapsing branches

Branches in a tree (i.e. intermediate nodes with subnodes) have a small expansion box in front of them containing either a plus or a minus sign. Collapsed branches have a plus sign +,, and can be expanded one level by a single click on the plus sign (to show *more* information). Expanded branches have a minus sign -, and can be collapsed by a single click on the minus sign (to show *less* information). Alternatively, a node can be expanded or collapsed by double clicking on its icon. Leaf nodes have no associated expansion box.

Double-clicking a node

When you double-click (or press **Enter**) on the name of any node in a tree, AIMMS will invoke the most commonly used menu command that is specific for each tree.

- In the **Model Explorer**, the double-click is identical to the **Edit-Attributes** menu, which opens the attribute window for the selected node.
- In the **Identifier Selector**, the double-click is identical to the **Edit-Open With** menu, which opens a view window to simultaneously display the contents of the selection.
- In the **Page** and **Template Manager**, the double-click is identical to the **Edit-Open** menu, which opens the page or template.
- In the **Menu Builder**, the double-click is identical to the **Edit-Properties** menu, which opens the appropriate **Properties** dialog box.

Alternatively, you can open the attribute form or **Properties** dialog box of any node type using the **Properties** button  on the toolbar.

Creating new nodes

To create a new node in the model tree you must position the cursor at the node in the tree *after* which you want to insert a new node. You can create a new node here:

- by clicking on one of the node creation icons  or  on the toolbar
- by selecting the item **Insert...** from the right-mouse menu, or
- by pressing the **Ins** key on the keyboard.

The toolbar contains creation icons for the most common node types. You can select the **New...** icon  to select further node types.

Selecting a node type

Once you have clicked the **New...** icon  on the toolbar, or selected the **Insert...** menu from the right-mouse menu, or have pressed the **Ins** key, a dialog box as shown in Fig. 2.6

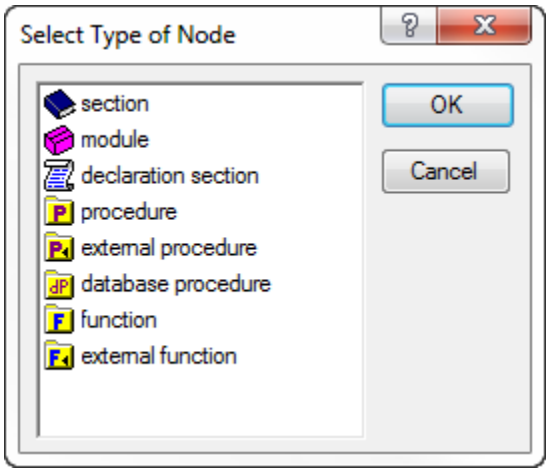


Fig. 2.6: Dialog box for selecting a node type

appears from which you have to select a node type. The dialog box shows only those node types that are allowed at the particular position in the tree. You can select a node type by a single mouse click, or by typing in the first letter of the node type that you want to insert. When there are more node types that begin with the same letter (as in Fig. 2.6), re-type that letter to alternate over all possibilities.

Naming the node

After you have selected a node type, it is inserted in the model tree, and you have to enter a name for the new node. In the model tree, all node names must consist only of alphanumeric characters and underscores, and must start with a letter. In addition, the names of structuring nodes may contain spaces. For most node types their node names have to be unique throughout the model. The only, quite natural, exception are declaration sections which accept either the predefined name *Declaration* or a name unique throughout the model.

Expanding branches without subnodes

When you want to add subnodes to a branch, you must first expand the branch. If you do not do this, a new node will be inserted directly after the branch, and not as a subnode. Expanding an empty branch will result in an empty subtree being displayed. After expansion you can insert a new node in the usual manner.

Renaming existing nodes

You can rename a selected node by pressing the **F2** button, or single clicking on the node name. After changing the name, press the **Enter** key to action the change, or the **Esc** key to cancel. When the node is an identifier declaration, a procedure, or a function which is used elsewhere in the model (or displayed on a page in the graphical user interface), AIMMS will, if asked, automatically update such references to reflect the name change.

Multiple selections

Unlike the Windows Explorer, AIMMS lets you make multiple selections within a tree which you can delete, cut, copy and paste, or drag and drop. The nodes in a selection do not even have to be within the same branch. By left-clicking in combination with the **Ctrl** key you can add or delete single nodes from the selection. By left-clicking in combination with the **Shift** key you can add all nodes between the current node and the last selected node.

Deleting nodes and branches

You can delete all nodes in a selection by selecting **Delete** from the right-mouse menu, or by pressing the **Del** key. When the selection contains branch nodes, AIMMS will also delete all child nodes contained in that branch.

Cut, copy, paste and duplicate

With the **Cut**, and **Copy** and **Paste** items from the **Edit** menu, or right-mouse menu, you can cut or copy the current selection from the tree, and paste it elsewhere. In addition to the usual way of pasting, which copies information from one position to another, AIMMS also supports the **Paste as Duplicate** operation in the **Identifier Selector**, the **Template Manager** and the **Menu Builder**. This form of pasting makes no copy of the node but only stores a reference to it. In this way changes in one node are also reflected in the other.

Drag and drop support

In addition to the cut, and copy and paste types of operation, you can drag a node selection and drop it onto another position in the model tree, or in any of the other tools offered by AIMMS. Thus you can, for instance, easily move a declaration section to another position in the model tree, or to an existing selection in the selection manager.

Copying or moving with drag and drop

By pressing the **Shift** or **Ctrl** keys during a drag-and-drop action, you can alter its default action. In combination with the **Shift** key, AIMMS will *move* the selection to the new position, while the **Ctrl** key will *copy* the selection to the new position. With the **Shift** and **Control** key pressed simultaneously, you activate the special *find* function explained in the next paragraph. AIMMS will show the type of action that is performed when you drop the selection by modifying the mouse pointer, or by displaying a stop sign when a particular operation is not permitted.

Searching for identifiers

AIMMS offers several tools for finding model-related information quickly and easily.

- When the attribute of an identifier, or the body of a procedure or function, contains a reference to another identifier within your application, you can pop up the attribute form of that identifier by simply clicking on the reference and selecting the **Attributes...** item from the right-mouse menu.
- With the **Find...** item from the **Edit** menu (or the **Find** button  on the toolbar) you can search for all occurrences of an identifier in your entire model or in a particular branch. The **Find** function also offers the possibility of restricting the search to only particular node attributes.
- The **Identifier Selector** offers an advanced tool for creating identifier selections on the basis of one or more dynamic criteria. You can subsequently select a view from the **View Manager** to display and/or change a subset of attributes of all identifiers in the selection simultaneously. Selections and views are discussed in full detail in *Viewing Identifier Selections* (page 54).
- By dragging a selection of identifiers onto any other tree while pressing the **Ctrl** and **Shift** key simultaneously, AIMMS will highlight those nodes in the tree onto which the selection is dropped, in which the identifiers in the selection play a role. This form of drag and drop support does not only work with identifier selections, but can be used with selections from any other tree as well. Thus, for instance, you can easily find the pages in which a particular identifier is used, or find all pages that use a particular end-user menu or toolbar.

2.2 Identifier Declarations

This chapter shows you how to add new identifier declarations using the **Model Explorer** and how to modify existing identifier declarations. The chapter also explains how any changes you make to either the name or the domain of an identifier are propagated throughout the remainder of your model.

2.2.1 Adding identifier declarations

Identifiers

Identifiers form the heart of your model. All data are stored in identifiers, and the bodies of all functions and procedures consist of statements which compute the values of one identifier based on the data associated with other identifiers.

Adding identifiers

Adding an identifier declaration to your model is as simple as adding a node of the desired type to a global declaration section (or to a declaration section local to a particular procedure or function), as explained in *Working with trees* (page 31). AIMMS will only allow you to add identifier declarations inside declaration sections.

Identifier types

There are many different types of identifiers. Each identifier type corresponds to a leaf node in the model tree and has its own icon, consisting of a white box containing one or more letters representing the identifier type. When you add an identifier to a declaration section of your model in the model tree, you must first select its identifier type from the dialog box as presented in Fig. 2.7.

Identifier name

After you have selected the identifier type, AIMMS adds a node of the specified type to the model tree. Initially, the node name is left empty, and you have to enter a unique identifier name. If you enter a name that is an AIMMS keyword, an identifier predeclared by AIMMS itself, or an existing identifier in your model, AIMMS will warn you of this fact. By pressing the **Esc** key while you are entering the identifier name, the newly created node is removed from the tree.

Meaningful names are preferable

There is no strict limit to the length of an identifier name. Therefore, you are advised to use clear and meaningful names, and not to worry about either word length or the intermingling of small and capital letters. AIMMS offers special features for name completion such as **Ctrl-Spacebar** (see *Identifier attributes* (page 38)), which allow you to write subsequent statements without having to retype the complete identifier names. Name completion in AIMMS is also case consistent.

Index domain

In addition, when an identifier is multidimensional, you can immediately add the index domain to the identifier name as a parenthesized list of indices that have already been declared in the model tree. Alternatively, you can provide the index domain as a separate attribute of the identifier in its attribute form. Fig. 2.8 illustrates the two ways in which you can enter the index domain of an identifier.

In both cases the resulting list of indices will appear in the model tree as well as in the `IndexDomain` attribute of the attribute form of that identifier. In the `IndexDomain` attribute it is possible, however, to provide a further restriction to the domain of definition of the identifier by providing one or more domain conditions (as explained in full detail in the Language Reference). Such conditions will not appear in the model tree.

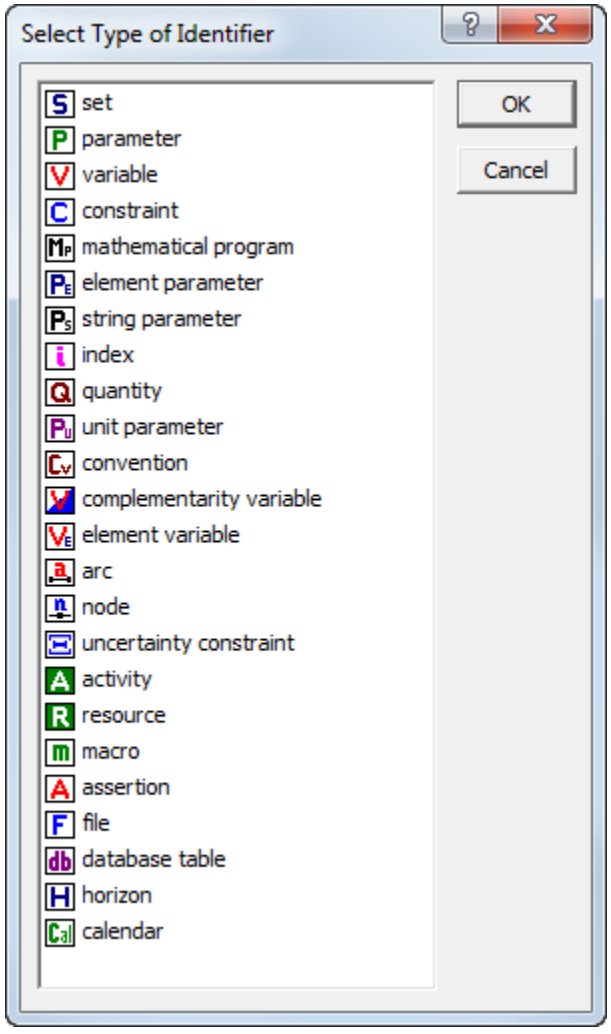
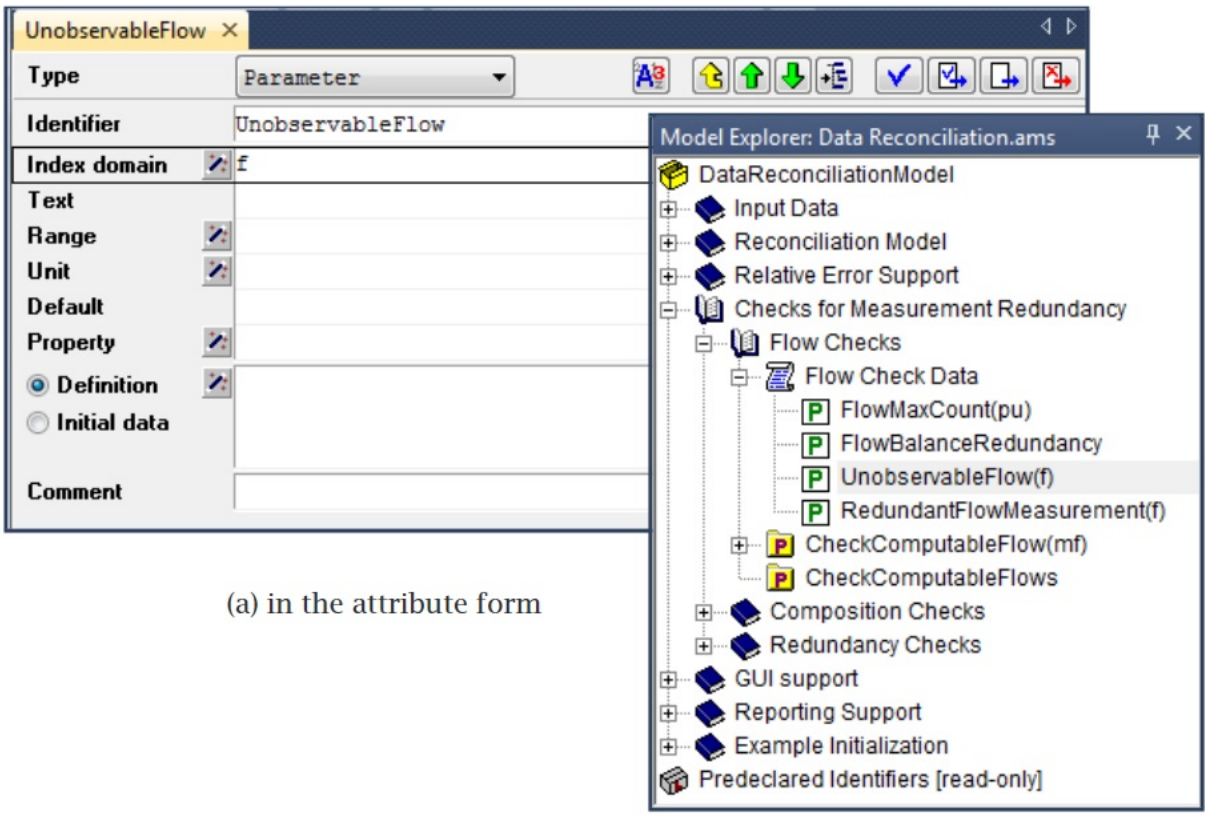


Fig. 2.7: Choosing an identifier type



(a) in the attribute form

(b) in the model explorer

Fig. 2.8: Specifying an index domain

Unrestricted order of declarations

The identifier declarations in the model tree can be used independently of the order in which they have been declared. This allows you to use an identifier anywhere in the tree. This order independence makes it possible to store identifiers where you think they should be stored logically. This is different to most other systems where the order of identifier declarations is dictated by the order in which they are used inside the model description.

Identifier scope

In general, all identifiers in an AIMMS model are known globally, unless they have been declared inside a local declaration section of a procedure or function. Such identifiers are only known inside the procedure or function in which they have been declared. When you declare a local identifier with the same name as a global identifier, references to such identifiers in the procedure or function will evaluate using the local rather than the global identifier.

Local declarations

Local identifiers declared in procedures and functions are restricted to particular types of identifier. For example, AIMMS does not allow you to declare constraints as local identifiers in a procedure or function, as these identifier types are always global. Therefore, when you try to add declarations to a declaration section somewhere in the model tree, AIMMS only lists those types of nodes that can be inserted at that position in the model tree.

Declarations via attributes

As an alternative to explicitly adding identifier nodes to the model tree, it is sometimes possible that AIMMS will implicitly define one or more identifiers on the basis of attribute values of other identifiers. The most notable examples are indices and (scalar) element parameters, which are most naturally declared along with the declaration of an index set. These identifiers can, therefore, be specified implicitly via the `Index` and `Parameter` attributes in the attribute form of a set. Implicitly declared identifiers do not appear as separate nodes in the model tree.

2.2.2 Identifier attributes

Identifier attributes

The attributes of identifier declarations specify various aspects of the identifier which are relevant during the further execution of the model. Examples are the index domain over which the identifier is declared, its range, or a definition which expresses how the identifier can be uniquely computed from other identifiers. For the precise interpretation of particular attributes of each identifier type, you are referred to the AIMMS Language Reference, which discusses all identifier types in detail.

Attribute window

The attributes of an identifier are presented in a standard form. This form lists all the relevant attributes together with the current values of these attributes. The attribute values are always presented in a textual representation, consisting of either a single line or multiple lines depending on the attribute. Fig. 2.9 illustrates the attribute form of a variable `ComponentFlow(f,c)`.

The attributes specify, for instance, that the variable is measured in `Mmol/h`, and provide a definition in terms of other parameters and variables.

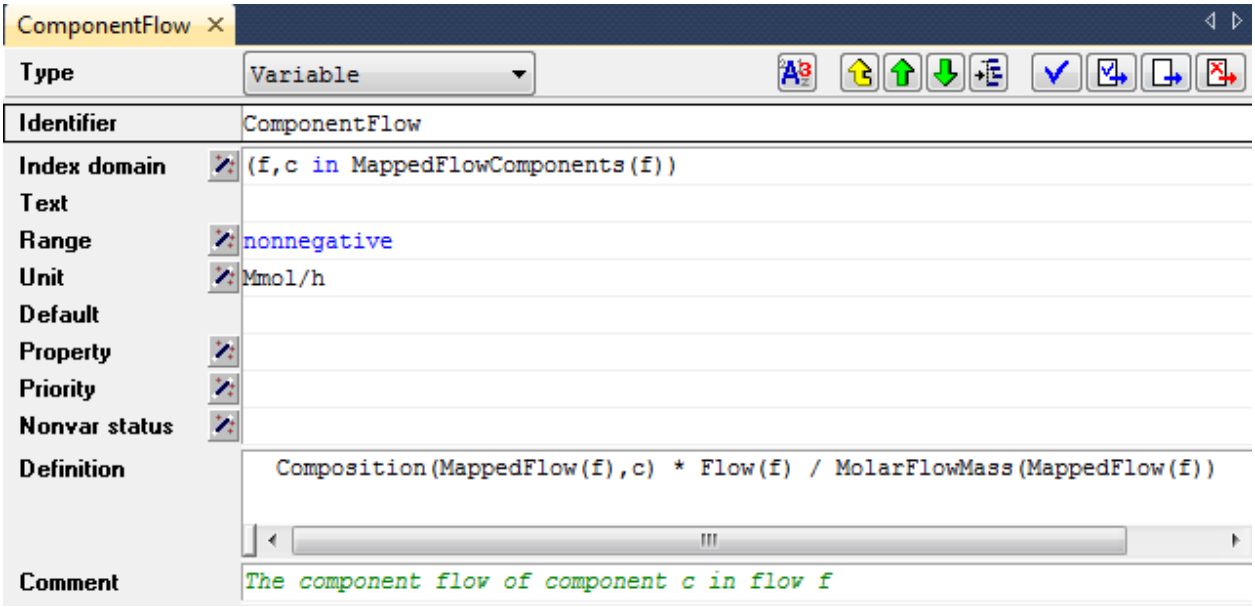


Fig. 2.9: Identifier attributes

Default values

You do not need to enter values for all the attributes in an attribute window. In fact, most of the attributes are optional, or have a default value (which is not shown). You only have to enter an attribute value when you want to alter the behavior of the identifier, or when you want to provide a value that is different to the default.

Entering attribute text ...

You can freely edit the text of almost every attribute field, using the mechanisms common to any text editor. Of course, you will then need to know the syntax for each attribute. The precise syntax required for each attribute is described in the AIMMS Language Reference book.

...or using attribute wizards

To help you when filling in attributes, AIMMS offers specialized wizards for most of them. These wizards consists of (a sequence of) dialog boxes, which help you make specific choices, or pick identifier names relevant for specifying the attribute. An example of an attribute wizard is shown is Fig. 2.10.

In this wizard, the numerical range of a particular parameter or variable is specified as the user-defined interval `[0, MaxFlowErrorBound]`. After completing the dialog box, the result of filling in the wizard is copied to the attribute window with the correct syntax.

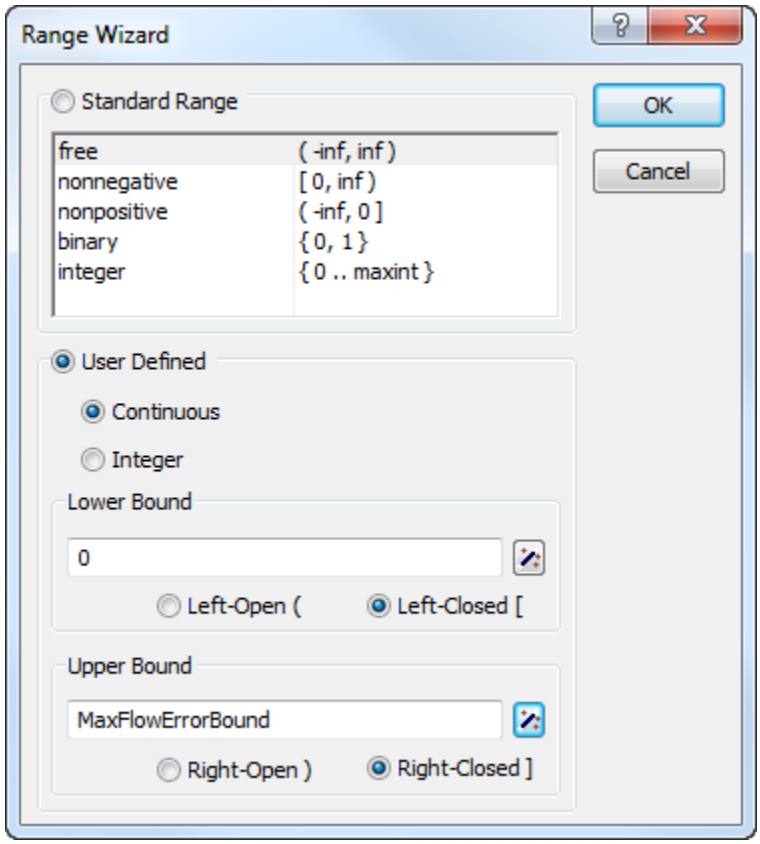


Fig. 2.10: Example of an attribute wizard

Mandatory use of wizards

Some of the attribute fields are not editable by hand, but require you to always use the associated wizard. AIMMS requires the use of wizards, whenever this is necessary to keep the model in a consistent state. Examples are (non-empty) `Index` and `Parameter` attributes of sets, as well ass the `BaseUnit` attribute of quantities.

Identifier reference support

Even when you decide to enter an attribute into a field manually, AIMMS still offers support to help you enter such a field quickly and easily. If your application contains a large number of identifiers and/or if the names of these identifiers are long, then it may be difficult to remember all the exact names. There are two ways to let AIMMS help you in filling in the appropriate names in an attribute field:

- you can drag and drop the names from the model tree into the field, or
- with the name completion feature you can let AIMMS fill in the remainder of the name based on only the first few characters typed.

Dragging identifiers

When filling in an attribute field, you can drag any identifier node in the model tree to a particular location in the attribute field. As a result, AIMMS will copy the identifier name, with its index domain, at the location where you dropped the identifier.

Name completion ...

When you use the **Ctrl-Spacebar** combination anywhere in an attribute field, AIMMS will complete any incomplete identifier name at the current cursor position wherever possible. With the **Ctrl-Shift-Spacebar** combination AIMMS will also complete keywords and predefined procedure and function names. When there are more than one possibilities, a menu of choices is presented as in Fig. 2.11. In this menu the first possible extension will be selected and the selection will be updated as you type. When an identifier name is complete, applying name completion will cause AIMMS to extend the identifier by its index domain as specified in its declaration.

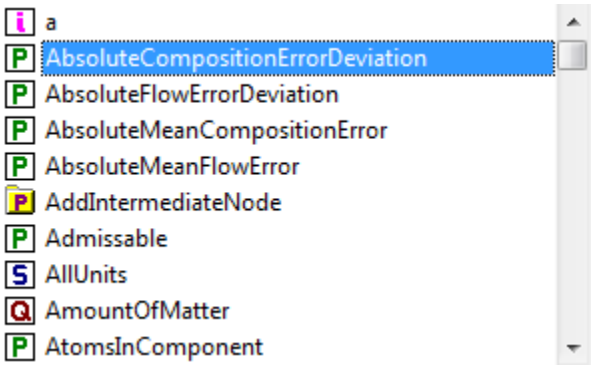


Fig. 2.11: Name completion

... applied to the :: and . characters

By pressing **Ctrl-Spacebar** in a string that contains the `::` or `.` characters, AIMMS will restrict the list of possible choices as follows.

- If the name in front of the `::` character is a module or library module prefix, AIMMS will show all the identifiers contained in the module, or all identifiers contained in the interface of the library module, respectively.
- If the string to complete refers to a property in a `PROPERTY` statement, and the name in front of the `.` character is an identifier, AIMMS will show all properties available for the identifier (based on its type).
- If the string to complete refers to an option in an `OPTION` statement, and the string in front of the `.` character refers to an element of the set `AllSolvers`, AIMMS will show all options available for that solver.
- In all other cases, if the name in front of the `.` character is an identifier, AIMMS will show all the suffices available for the identifier (based on its declaration).

Navigation Features

Navigation features

From within an attribute window, there are several menus and buttons available to quickly access related information, such as the position in the model tree, identifier attributes and data, and context help on identifier types, attributes and keywords.

Browsing the model tree

From within an attribute window you can navigate further through the model tree by using the navigation buttons displayed at the top of the window.

- The **Parent**  , **Previous**  and **Next Attribute Window**  buttons will close the current attribute window, and open the attribute window of the parent, previous or next node in the model, respectively.
- The **Location in Model Tree**  button will display the model tree and highlight the position of the node associated with the current attribute window.

Viewing identifier details

When an identifier attribute contains a reference to a particular identifier in your model, you may want to review (or maybe even modify) the attributes or current data of that identifier. AIMMS provides various ways to help you find such identifier details:

- by clicking on a particular identifier reference in an identifier attribute, you can open its attributes window through the **Attributes** item in the right-mouse pop-up menu,
- you can locate the identifier declaration in the model tree through the **Location in Model Tree** item in the right-mouse pop-up menu, and
- you can view (or modify) the identifier's data through the **Data** item in the right-mouse pop-up menu (see *Viewing and modifying identifier data* (page 45)).

Context help

Context sensitive help

Through either the **Context Help** button  on the toolbar, or the **Help on** item in the right-mouse pop-up menu, you can get online help for the identifier type, its attributes and keywords used in the attribute fields. It will open the section in one of the AIMMS books or help files, which provides further explanation about the topic for which you requested help.

2.2.3 Committing attribute changes

Syntax checking

The modifications that you make to the attributes of a declaration are initially only stored locally within the form. Once you take further action, the changes in your model will be checked syntactically and committed to the model. There are three ways to do this.

Saving the model

In addition to committing the changes in a single attribute form manually as above, the changes that you have made in any attribute form are also committed when you save the model (through the **File-Save** menu), or recompile it in its entirety (through the **Run-Compile All** menu).

Renaming identifiers

It is quite common to rename an existing identifier in a modeling application because you consider that a new name would better express its intention. In such cases, you should be aware of the possible consequences for your application. The following questions are relevant.

- Are there references to the (old) identifier name in other parts of the model?
- Are there case files that contain data with respect to the (old) identifier name?
- Are there pages in the end-user interface that display data with respect to the (old) identifier name?

If the answer to any of these questions is yes, then changing the identifier name could create problems.

Automatic name changes

AIMMS helps you in dealing with the possible consequences of name changes by offering the following support:

- AIMMS updates all references to the identifier throughout the model text, and in addition,
- AIMMS keeps a log of the name change (see also *Additional files related to an AIMMS project* (page 9)), so that when AIMMS encounters any reference to the old name in either a page or in a case file, the new name will be substituted.

Beware of structural changes

Problems arise when you want to change the index domain of an identifier, or remove an identifier, while it is still referenced somewhere in your application. Such changes are called *structural*, and are likely to cause errors in pages and cases. In general, these errors cannot be recovered automatically. To help you locate possible problem areas, AIMMS will mark all pages and cases that contain references to changed or deleted identifiers. To check how a change really affects these pages and cases, you should open them, make any required adaptations to deal with the errors, and resave them.

Modifying identifier type

You can modify the type of a particular identifier in the model tree via the identifier type drop-down list Element Parameter in the attribute window of the identifier. The drop-down list lets you select from all identifier types that are compatible with the current identifier type. Alternatively, you can change the identifier type via the **Edit-Change Type** menu.

Incompatible attributes

Before a change of identifier type is actually committed, AIMMS displays the dialog box illustrated in Fig. 2.12, which lists all the attributes of the identifier that are not compatible with the newly selected type.

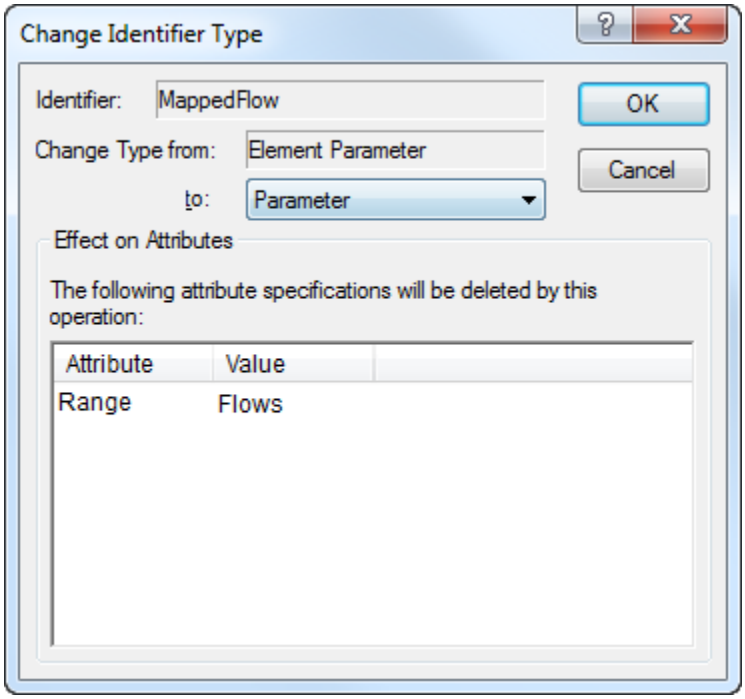


Fig. 2.12: The **Change Identifier Type** dialog box

If you do not want such attributes to be deleted, you should cancel the operation at this point. When you allow AIMMS to actually perform the type change, the incompatible attributes will be deleted.

2.2.4 Viewing and modifying identifier data

Viewing identifier data

When you are developing your model (or are reviewing certain aspects of it later on), AIMMS offers facilities to directly view (and modify) the data associated with a particular identifier. This feature is very convenient when you want to enter data for an identifier during the development of your model, or when you are debugging your model (see also *The AIMMS debugger* (page 61)) and want to look at the results of executing a particular procedure or evaluating a particular identifier definition.

The Data button

Via the **Data** button  available in the attribute window of every global identifier (see, for instance, Fig. 2.9), AIMMS will pop up one of the *data pages* as illustrated in Fig. 2.13.

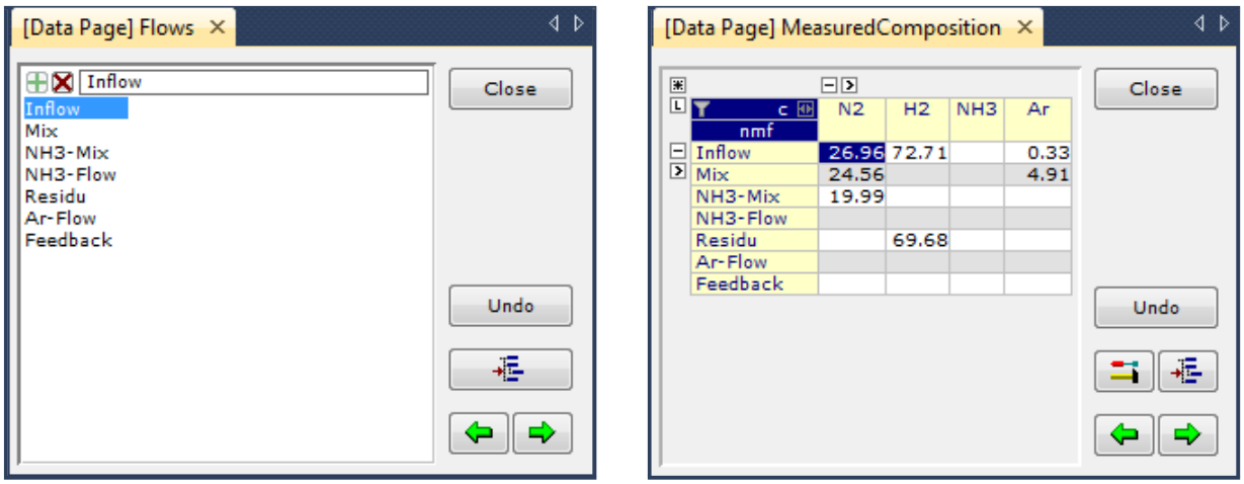


Fig. 2.13: Data pages of a set and a 2-dimensional parameter

Data pages provide a view of the *current contents* of the selected identifier. Which type of data page is shown by AIMMS depends on the type of the identifier. The data page on the left is particular to one-dimensional root sets, while the data page on the right is appropriate for a two-dimensional parameter.

Data pages for variables and constraints

For variables (and similarly for constraints), AIMMS will display a pivot table containing all the indices from the index domain of the variable plus one additional dimension containing all the suffices of the variable that contain relevant information regarding the solution of the variable. Depending on the properties set for the variable, this dimension may contain a varying number of suffices containing sensitivity data related to the variable.

Viewing data in the Model Explorer

Data pages can also be opened directly for a selected identifier node in the model tree using either the **Edit-Data** menu, or the **Data** command in the right-mouse pop-up menu. Additionally, you can open a data page for any identifier referenced in an attribute window by selecting the identifier in the text, and applying the **Data** command from the right-mouse pop-up menu.

Multidimensional identifiers

For multidimensional identifiers, AIMMS displays data using a default view which depends on the identifier dimension. Using the  button on the data page you can modify this default view. As a result, AIMMS will display the dialog box illustrated in Fig. 2.14.

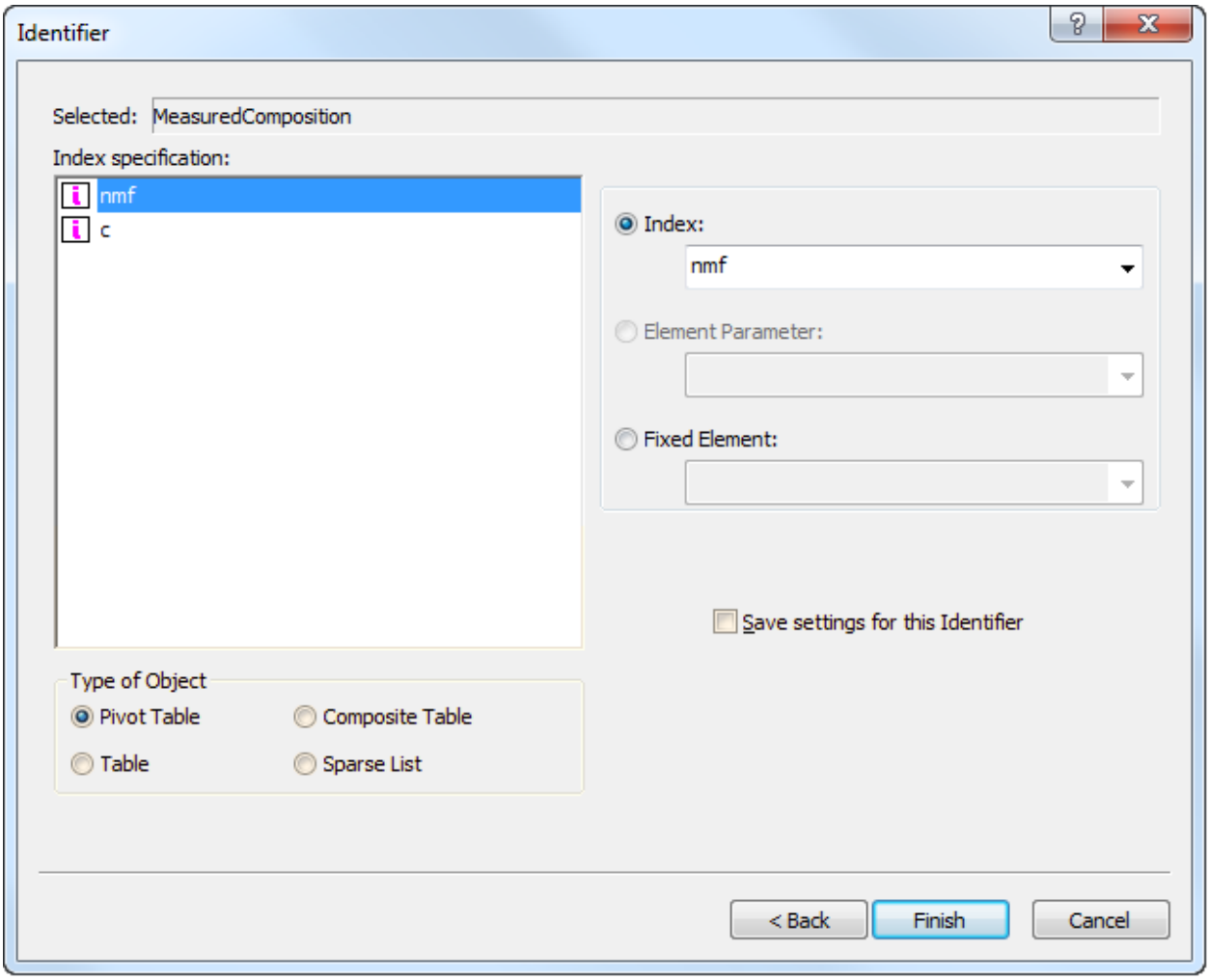


Fig. 2.14: Selecting a data page type

In this dialog box, you can select whether you want to view the data in a sparse list object, a composite table object, a pivot table object or in the form of a (rectangular) table. Additionally, you can indicate that you want the view to be *sliced*, by selecting fixed elements for one or more dimensions. For every sliced dimension, AIMMS will automatically add a floating index to the data page, allowing you to view the data for every element in the sliced dimension.

Saving your choice

If you want to always use the same data page settings for a particular identifier, you can save the choices you made in Fig. 2.14. As a result, AIMMS will save the data page as an ordinary end-user page in the special **All Data Pages** section of the **Page Manager**. If you so desire, you can further edit this page, and, for instance, add additional related identifiers to it which will subsequently become visible when you view the identifier data in the **Model Explorer**.

End-user page as data page

Whenever there is a page in the **All Data Pages** section of the page manager with the fixed name format `[Data Page]` followed by the name of an identifier of your model, AIMMS will use this page as the data page for that identifier. This enables you to copy a custom end-user page, that you want to use as a data page for one or more identifiers, to the **All Data Pages** section of the page manager, and rename it in the prescribed name format. When you remove a page from the **All Data Pages** section, AIMMS will again open a default data page for that identifier. If you hold down the **Shift** key while opening a data page, AIMMS will always use the default data page.

Global and local identifiers

Normally, AIMMS will only allow you to open data pages of global identifiers of your model. However, within the AIMMS debugger (see also *The AIMMS debugger* (page 61)), AIMMS also supports data pages for local identifiers within a (debugged) procedure, enabling you to examine the contents of local identifiers during a debug session.

2.3 Procedures and Functions

This chapter describes how you can add procedures and functions to a model. It also shows how you can add arguments and local identifiers to procedures and functions. In addition, it illustrates how the body of a procedure or function can be broken down into smaller pieces of execution code, allowing you to implement procedures and functions in a top-down approach.

2.3.1 Creating procedures and functions

Procedures and functions

Procedures and functions are the main means of executing the sequential tasks in your model that cannot be expressed by simple functional relationships in the form of identifier definitions. Such tasks include importing or exporting your model's data from or to an external data source, the execution of data assignments, and giving AIMMS instructions to optimize a system of simultaneous equations.

Creating procedures and functions

Procedures and functions are added as a special type of node to the model tree, and must be placed in the main model, or in any book section. They cannot be part of declaration sections, which are exclusively used for model identifiers. Procedures and functions are represented by folder icons, which open up when the node is expanded. Fig. 2.15 illustrates an example of a procedure node in the model tree.

Naming and arguments

After you have inserted a procedure or function node into the tree, you have to enter its name. If you want to add a procedure or function with arguments, you can add the argument list here as well. Alternatively, you can specify the argument list in the attribute window of the procedure or function. The full details for adding arguments and their declaration as identifiers, local to the procedure or function, are discussed in *Declaration of arguments and local identifiers* (page 49). Whether or not the arguments are fully visible in the tree is configurable using the **View** menu.

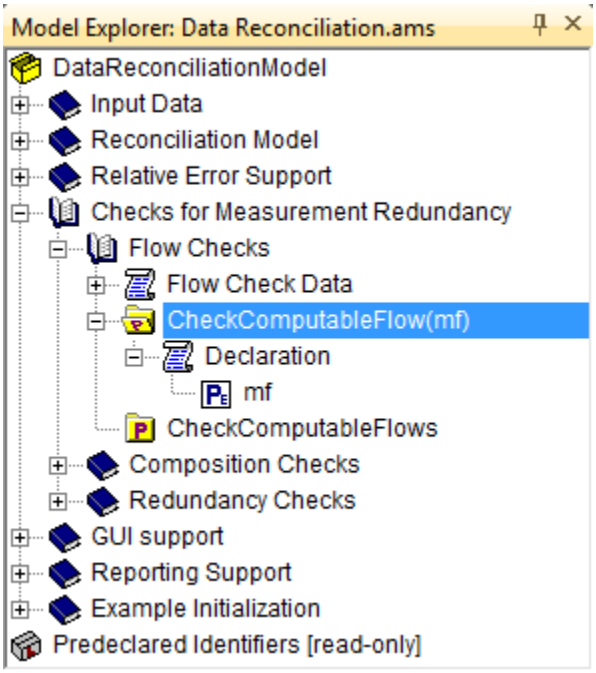


Fig. 2.15: Example of a procedure node

Example of a procedure node

The attribute window of a procedure or function lets you specify or view aspects such as its list of arguments, the index domain of its result, or the actual body. The body may merely consists of a `solve` statement to solve an optimization model, but can also consist of a sequence of execution and flow control statements.

The contents of the `Body` attribute is application-specific, and is irrelevant to a further understanding of the material in this section.

Specifying function domain and range

When the resulting value of a function is multidimensional, you can specify the index domain and range of the result in the attribute form of the function using the `IndexDomain` and `Range` attributes. Inside the function body you can make assignments to the function name as if it were a local (indexed) parameter, with the same dimension as specified in the `IndexDomain` attribute. The most recently assigned values are the values that are returned by the function.

2.3.2 Declaration of arguments and local identifiers

Specifying arguments

All (formal) arguments of a procedure or function must be specified as a parenthesized, comma-separated, list of non-indexed identifier names. All formal arguments must also be declared as local identifiers in a declaration section local to the procedure or function. These local declarations then specify the further domain and range information of the arguments. If an argument has not been declared when you create (or modify) a procedure or function, AIMMS will open the dialog box illustrated in Fig. 2.16 which helps you add the appropriate declaration quickly and easily.

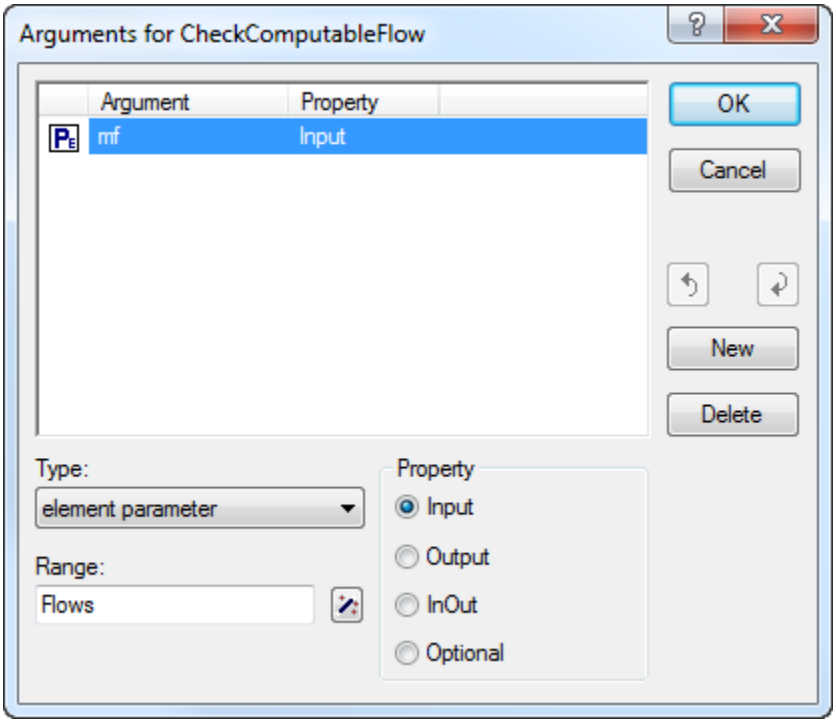


Fig. 2.16: **Argument Declaration** dialog box

After completing the dialog box, AIMMS will automatically add a declaration section to the procedure or function, and add the arguments displayed in the dialog box to it, as illustrated in Fig. 2.15.

Input-output type

An important aspect of any argument is its input-output type, which can be specified by selecting one of the `Input`, `InOut`, `Output` or `Optional` properties in the **Argument Declaration** dialog box. The input- output type determines whether any data changes you make to the formal arguments are passed back to the actual arguments on leaving the procedure. The precise semantic meaning of each of these properties is discussed in the AIMMS Language Reference book.

Argument attributes

The choices made in the **Argument Declaration** dialog box are directly reflected in the attribute form of the local identifier added to the model tree by AIMMS. As an example, Fig. 2.17 shows the attribute form of the single argument `mf` of the procedure `CheckComputableFlow` added in Fig. 2.16.

In the dialog box of Fig. 2.16 it is not possible to modify the dimension of a procedure or function argument directly. If your procedure or function has a multidimensional argument, you can specify this with the `IndexDomain` attribute of the argument after the argument has been added as a local identifier to the model tree.

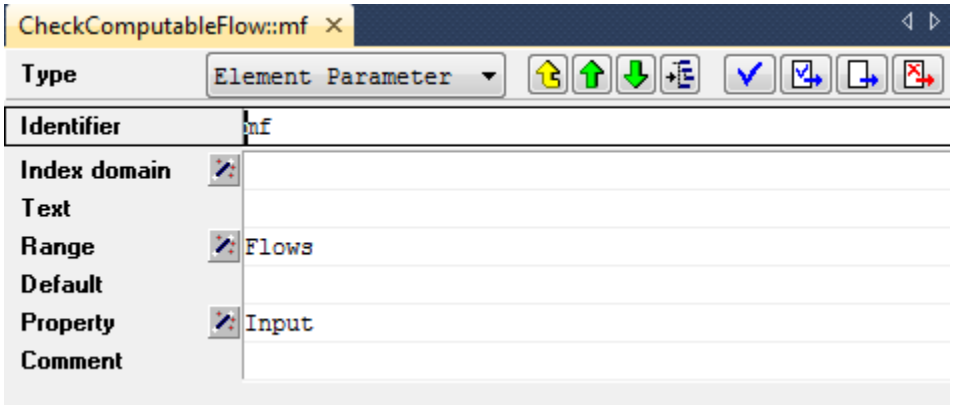


Fig. 2.17: Declaration form of a procedure argument

Prototype checking

For every call to the procedure or function, AIMMS will verify whether the types of all the actual arguments match the prototypes supplied for the formal arguments, including the supplied index domain and range. For full details about argument declaration refer to the AIMMS Language Reference book.

Local declarations

In addition to arguments, you can also add other local identifiers to declaration sections within procedures and functions. Such local identifiers are only known inside the function or procedure. They are convenient for storing temporary data that is only useful within the context of the procedure or function, and have no global meaning or interpretation.

Not all types supported

Not all identifier types can be declared as local identifiers of a procedure or function, because of the global implications they may have for the AIMMS execution engine. When you try to add a local identifier to a procedure or function, AIMMS will only offer those identifier types that are actually supported within a procedure or function. An example of an identifier type that cannot be declared locally is a constraint.

Not all attributes supported

In addition, for local identifiers, AIMMS may only support a subset of the attributes that are supported for global identifiers of the same type. For instance, AIMMS does not allow you to specify a `Definition` attribute for local sets and parameters. In the attribute window of local identifiers such non-supported attributes are automatically removed when you open the associated attribute form.

2.3.3 Specifying the body

Statements

In the **Body** attribute of a procedure or function you can specify the

- assignments,
- execution statements such as `solve` or `read`/ `write`
- calls to other procedures or functions in your model, and
- flow control statements such as `for`, `while` or `if-then-else`

which perform the actual task or computation for which the procedure or function is intended. The precise syntax of all execution statements is discussed in detail in the AIMMS Language Reference book.

Automatic outlining

When you are constructing a procedure or function whose execution consists of a large number of (nested) statements, it may not always be easy or natural to break up the procedure or function into a number of separate procedures. To help you maintain an overview of such large pieces of execution code, AIMMS will automatically add outlining support for common flow control statement and multiline comments. The minus button ⊟ that appears in front of the statement allows you to collapse the statement to a single line block and the and plus button ⊞ allows you to expand the statement to its full extent again.

Execution blocks

In addition, you can break down the body of a procedure or function into manageable pieces using one or more execution blocks. Any number of AIMMS statements enclosed between `block` and `endblock` keywords can be graphically collapsed into a single block. The text in the single line comment following the `block` keyword is used as display text for the collapsed block.

An example of a procedure body containing two collapsed execution blocks is given in Fig. 2.18.

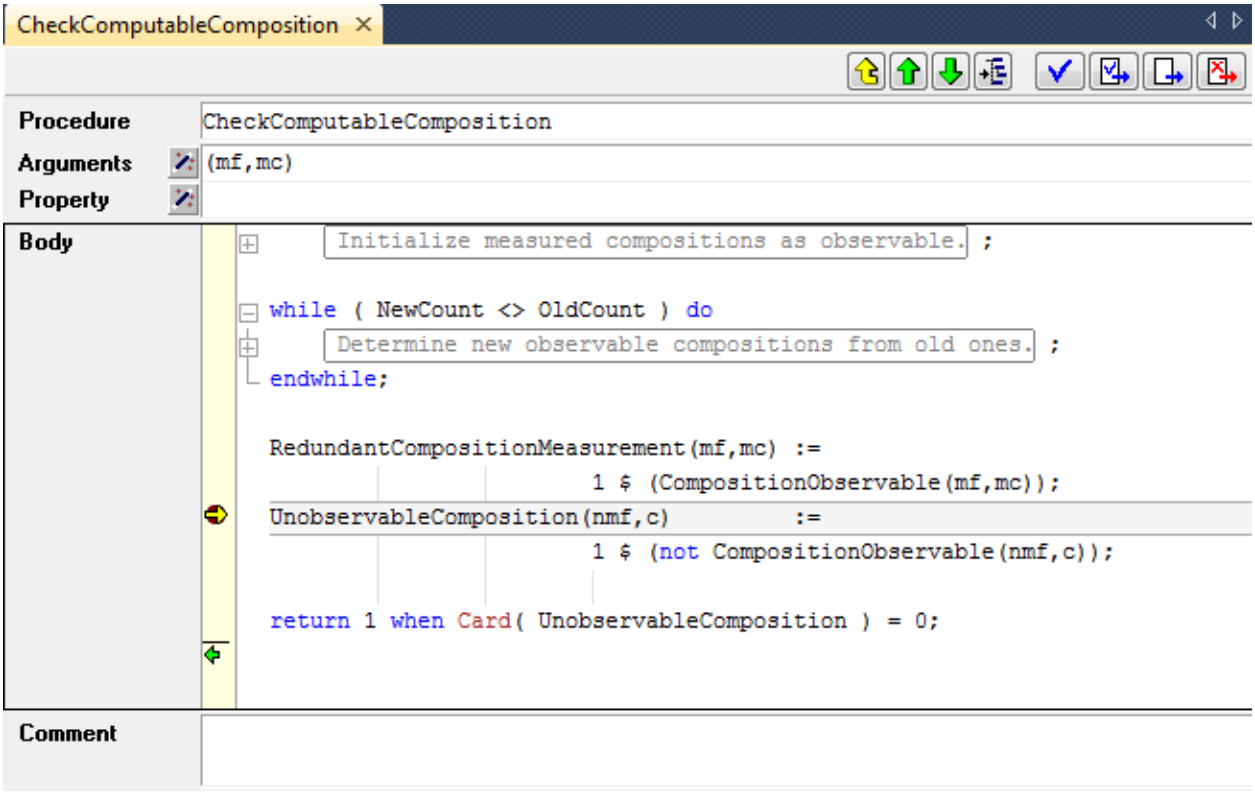


Fig. 2.18: Example of a procedure body with execution blocks

Identifier references

When you are entering statements into a body of a procedure or function, AIMMS can help you to add identifier references to the body quickly and easily:

- you can drag and drop the names from the model tree into text
- with the name completion feature you can let AIMMS complete the remainder of the name based on only the first characters typed.

The precise details of drag-and-drop support and name completion of identifiers are discussed in *Working with trees* (page 31) and *Identifier attributes* (page 38).

Viewing identifier details

When you are entering the body of a procedure or function, you may want to review the attributes or current data of a particular identifier referenced in the body. AIMMS offers various ways to help you find such identifier details:

- through a text based search in the model tree, you can locate the specific identifier node and open its attribute form (see *Working with trees* (page 31)),
- by clicking on a particular identifier reference in the body, you can open its attributes form through the **Attributes** item in the right-mouse pop-up menu,
- you can locate the identifier declaration in the model tree through the **Location in Model Tree** item in the right-mouse pop-up menu, and

- you can view (or modify) the identifier's data through the **Data** item in the right-mouse pop-up menu (see *Viewing and modifying identifier data* (page 45)).

Viewing procedure details

Similarly, while you are referencing a procedure or function inside the body of another procedure or function, AIMMS will provide prototype information of such a procedure or function as soon as you enter the opening bracket (or when you hover with the mouse pointer over the procedure or function name). This will pop up a window as illustrated in Fig. 2.19.

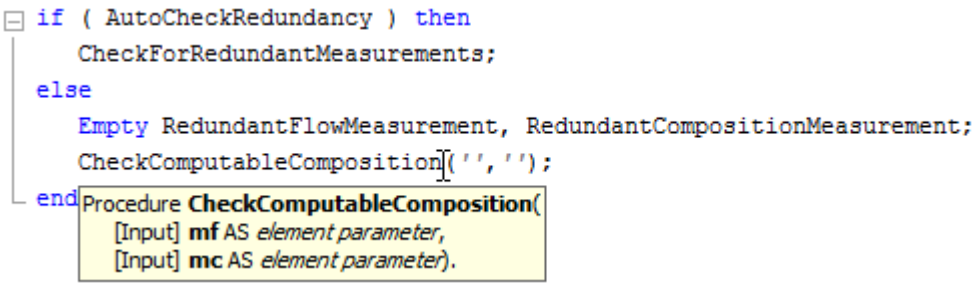


Fig. 2.19: Prototype info of a procedure

This tooltip window displays all arguments of the selected procedure or function, their respective data types, as well as their *Input-Output* status. The latter enables you to assess the (global) effect on the actual arguments of a call to the procedure.

2.3.4 Syntax checking, compilation and execution

Performing a syntax check

Using either **Check and commit** or **Check, commit and close** as discussed in *Committing attribute changes* (page 43) AIMMS will compile the procedure or function in hand, and point out any syntax error in its body. If you do not want to compile a procedure or function, but still want to commit the changes, you should use the **Commit and close** button. All edits are ignored when you close the window using the **Discard** button.

Partial recompilation

Before executing any procedure in your model, AIMMS will automatically verify whether your model needs to be recompiled, either partially or fully. In most cases, there is no need for AIMMS to recompile the entire model after a modification or addition of a new identifier, a procedure or a function. For instance, when you have only changed the body of a procedure, AIMMS needs only to recompile that particular procedure.

Complete recompilation

However, if you change the index domain of an identifier or the number of arguments of a procedure or function, each reference to such an identifier, procedure or function needs to be verified for correctness and possibly changed. In such cases, AIMMS will (automatically) recompile the entire model before any further execution can take place. Depending on the size of your model, complete recompilation may take some time. Note that either partial or complete recompilation will only retain the data of all identifiers present prior to compilation, to the extent possible (data cannot be retained when, for instance, the dimension of an identifier has changed).

Running a procedure

AIMMS supports several methods to initiate procedural model execution. More specifically, you can run procedures

- from within another procedure of your model,
- from within the graphical user interface by pressing a button, or when changing a particular identifier value, or
- by selecting the **Run procedure** item from the right-mouse menu for any procedure selected in the **Model Explorer**.

The first two methods of running a procedure are applicable to both developers and end-users. Running a procedure from within the **Model Explorer** a useful method for testing the correct operation of a newly added or modified procedure.

2.4 Viewing Identifier Selections

Identifier overviews

Although the **Model Explorer** is a very convenient tool to organize all the information in your model, it does not allow you to obtain a simultaneous overview of a group of identifiers that share certain aspects of your model. By mutual comparison of important attributes (such as the definition), such overviews may help you to further structure and edit the contents of your model, or to discover oversights in a formulation.

To assist you in creating overviews that can help you analyze the interrelationships between identifiers in your model, AIMMS offers the **Identifier Selector** tool and **View** windows. This chapter helps you understand how to create meaningful identifier selections with the **Identifier Selector**, and how to display such selections using different views.

2.4.1 Creating identifier selections

Select by similarity

When you are developing or managing a large and complicated model, you sometimes may need an overview of all identifiers that have some sort of similarity. For example, it may be important to have a simultaneous view of

- all the constraints in a model,
- all variables with a definition,
- all parameters using a certain domain index, or
- all identifiers that cover a specific part of your model.

Identifier selections

In AIMMS, you can create a list of such identifiers using the configurable **Identifier Selector** tool. This tool helps you to create a selection of identifiers according to a set of one or more criteria of varying natures. You can let AIMMS create a once only selection directly in the **Model Explorer**, or create a compound selection in the **Identifier Selector**, which allows you to intersect or unite multiple selections.

Creating once only selections

If you need a selection only once, then you can create it directly in the **Model Explorer** by

- either manually selecting one or more nodes in the tree, or
- using the **View-Selection** menu to create a custom selection based on one or more of the conditional selection criteria offered by AIMMS (explained below).

In both cases, the resulting list of selected identifiers will be highlighted in the model tree. If you like, you can narrow down or extend the selection by applying one or more subsequent conditional selections to the existing selection.

The Identifier Selector

If you need a specific selection more than once, then you can create it in the **Identifier Selector** tool. The **Identifier Selector** consists of a tree in which each node contains one of the three types of identifier selectors described below. Fig. 2.20 illustrates an example selector tree.

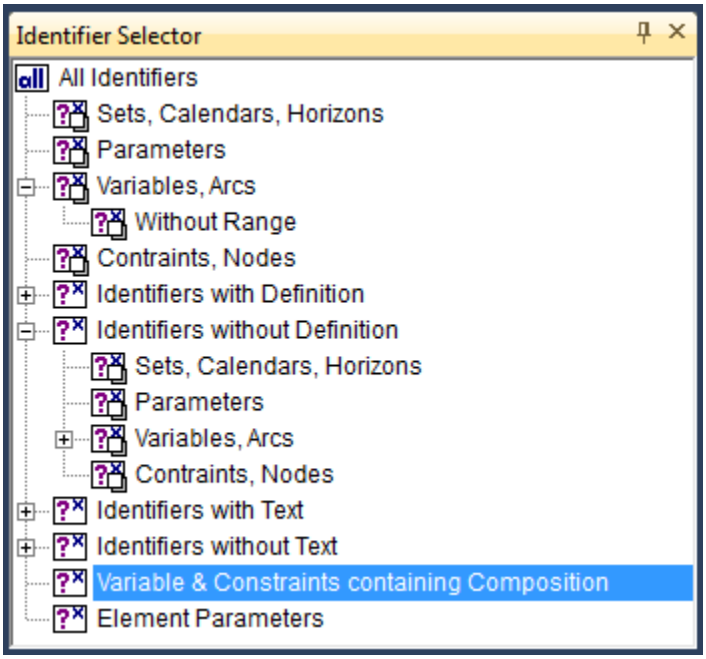


Fig. 2.20: The selector tree

Selector types

In the **Identifier Selector** tool, you can add nodes corresponding to several types of identifier selectors:

- a *node-based selector* , where all the identifiers below one or more user-selected nodes in the model tree are added to the selection,
- a *conditional selector* , where the list of identifiers is created dynamically on identifier type and/or the contents of one of their respective attributes,
- a *set-dependent selector* , where the list of identifiers is created dynamically based on a specific set in either the domain or range of identifiers, or
- a *type-based selector* , where the list of identifiers consists of all variables of a certain type (e.g. free, nonnegative, binary) or all constraints of a certain type ($\leq$, $=$ or $\geq$). This selector can only be used in combination with the Math Program Inspector.

While the above four selectors allow you to define selections based on a symbolic criteria, the four types of identifier selectors below allow you to specify selections based on individual criteria. The main purpose of these selectors is to define selections that can be used in the Math Program Inspector (see *The Math Program Inspector* (page 78)).

- an *element-dependent selector* , where the list of individual identifiers is created dynamically based of the occurrence of one or more specific elements in the domain,
- a *scale-based selector* , where the list of identifiers is built up from all variables and constraints for which the ratio between the largest absolute value and the smallest absolute value in the corresponding row or column of the matrix exceeds a given value,
- a *status-based selector* , where the list of identifiers is built up from all variables and constraints for which the solution satisfies some property (e.g. feasible, basic, at bound), or
- a *value-based selector* , where the list of identifiers is built up from all variables and constraints for which the level, bound, marginal, or bound violation value satisfy satisfy some property.

Through the **View-Selection** menu in the **Model Explorer** you can only create a new, or refine an existing, selection using a *conditional selector*.

Selection dialog box

To create a selector, AIMMS offers special dialog boxes which let you specify the criteria on which to select. As an example the dialog box for creating a conditional selector is illustrated in Fig. 2.21.

In it, you can select (by double clicking) one or more identifier types that you want to be part of the selection and filter on specific attributes that should be either empty, nonempty, or should contain a particular string.

Compound selections

The tree structure in the **Identifier Selector** defines combinations of selectors by applying one of the set operators *union*, *difference* or *intersection* with respect to the identifier selection represented by the parent node. The root of the tree always consists of the fixed selection of all model identifiers. For each subsequent child node you have to indicate whether the node should add identifiers to the parent selection, should remove identifiers from the parent selection, or should consider the intersection of the identifiers associated with the current and the parent selection. Thus, you can quickly compose identifier selections that satisfy multiple selection criteria. The type of set operation applied is indicated by the icon of each node in the identifier selector.

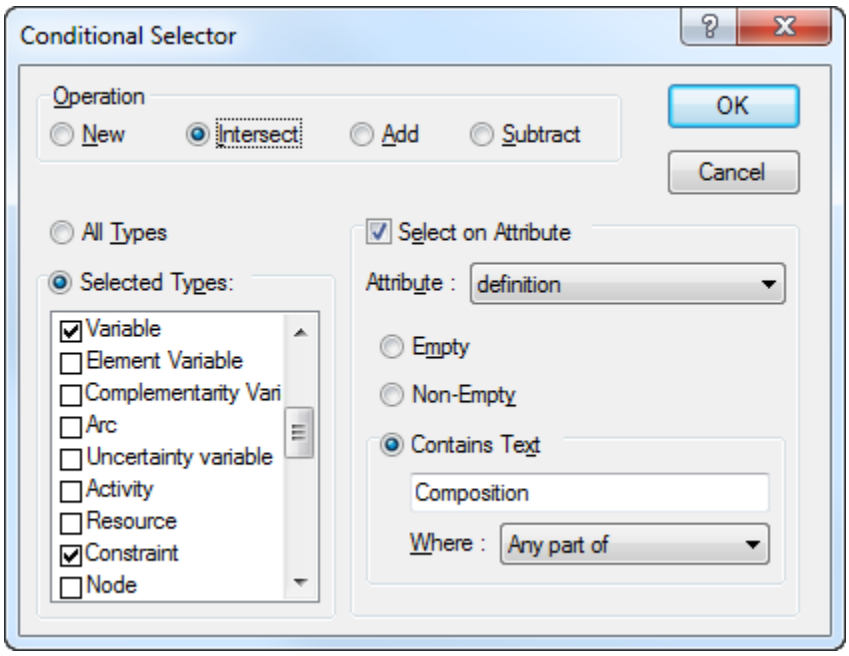


Fig. 2.21: The **Conditional Selector** dialog box

Refining model tree selections

In the **Model Explorer**, the *union*, *difference* and *intersection* operations apply to the identifier selection that is currently highlighted in the model tree. You can use them to add identifiers to the current selection, to remove identifiers from the current selection, or filter the current selection by means of an additional criterion.

Using selections

The list of identifiers that results from a (compound) identifier selector can be used in one of the following ways:

- you can display the identifiers in a **View** window of your choice (explained in the next section),
- you can restrict the set of variables and constraints initially displayed in the **Math Program Inspector** (see *The Math Program Inspector* (page 78)), or
- by dragging and dropping a selector into the **Model Explorer**, the corresponding identifiers will be highlighted in the model tree.

Advanced drag and drop

The drag-and-drop features of AIMMS make it very easy to fill a **View** window with identifiers from either the model tree, the **Identifier Selector** or other **View** windows. If you drag-and-drop a selection into any other AIMMS window, AIMMS will interpret this as a special search action to highlight all occurrences of the selected identifiers as follows:

- in the *model tree* all identifiers in the selection will be highlighted,
- in the *page* or *template tree* all pages that contain reference to the identifiers in the selection will be highlighted,
- in an end-user *page*, in edit mode, all objects that contain references to the identifiers will be selected, and
- in the *menu builder tree*, AIMMS will highlight all menu items that reference one or more identifiers in the selection.

In addition, AIMMS also supports the 'drag-and-drop-search' action in a **View** window by pressing both the **Shift** and **Control** key during the drop operation.

2.4.2 Viewing identifier selections

Overview of attributes

After you have created an identifier selection, in either the **Model Explorer** or in the **Identifier Selector**, you may want to compare or simultaneously edit multiple attributes of the identifiers in the selection. In general, sequential or simultaneous, opening of all the corresponding single attribute forms is impractical or unacceptable for such a task. To assist, AIMMS offers special identifier **View** windows.

Identifier views

A **View** window allows you to view one or more attributes simultaneously for a number of identifiers. Such a **View** window is presented in the form of a table, where each row represents a single identifier and each column corresponds to a specific attribute. The first column is always reserved for the identifier name. An example of an identifier **View** window is given in Fig. 2.22.

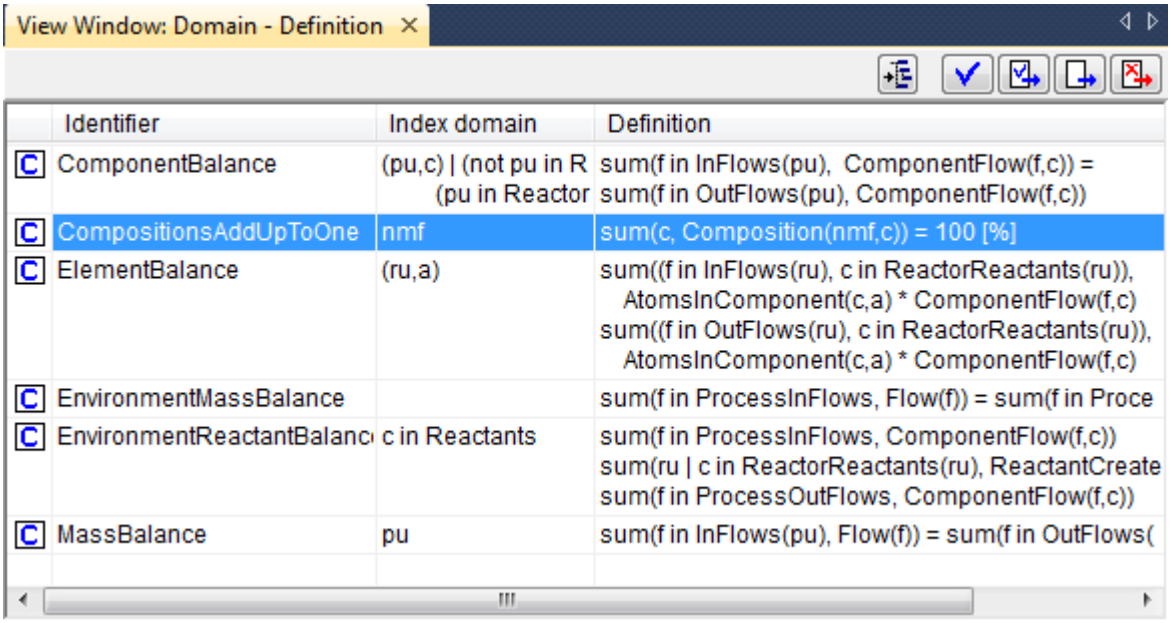


Fig. 2.22: Example of a View window

Editing in a View window

In addition to simply viewing the identifier content in a **View** window, you can also use it to edit individual entries. To edit a particular attribute of an identifier you can just click on the relevant position in the **View** window and modify the attribute value. This can be convenient, for instance, when you want to add descriptive text to all identifiers for which no text has yet been provided, or when you want to make consistent changes to units for a particular selection of identifiers. As in a single attribute form, the changes that you make are not committed in the model source until you use one of the special compile buttons at the top right of the window (see also *Committing attribute changes* (page 43)).

Opening a View window

Using the **Edit-Open with** menu, or the **Open with** item in the right- mouse pop-up menu, you can open a particular **View** window for any identifier selection in the model explorer or in the identifier selector. Selecting the **Open with** menu will open the **View Manager** dialog box as displayed in Fig. 2.23.

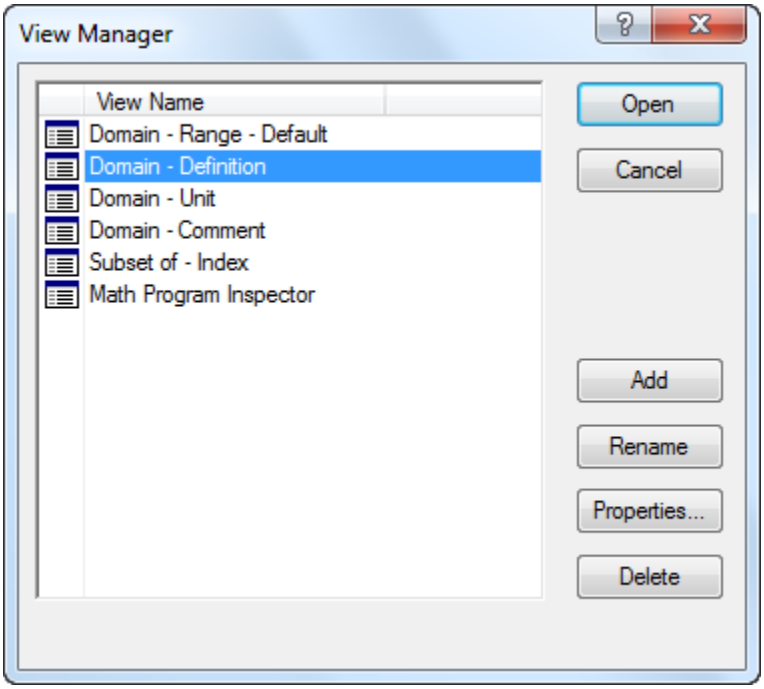


Fig. 2.23: The **View Manager** dialog box

In the **View Manager** you must select one of the available *view window definitions*, with which to view the given identifier selection. For every new project, the **View Manager** will automatically contain a number of basic view window definitions that can be used to display the most common combinations of identifier attributes.

Creating a view window definition

Using the **Add**, **Delete** and **Properties** buttons in the **View Manager**, you can add or delete view window definitions to the list of available definitions, or modify the contents of existing definitions. For every view window definition that you add to the list or want to modify, AIMMS will open the **View Definition Properties** dialog box as illustrated in Fig. 2.24.

With this dialog box you can add or remove attributes from the list of attributes that will be shown in the **View** window, or change the order in which the particular attributes are shown.

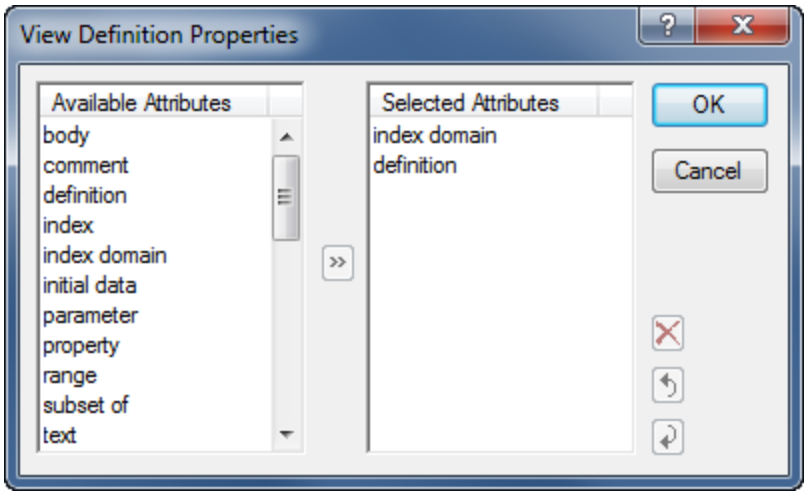


Fig. 2.24: **View Definition Properties** dialog box

Changing the View window contents

After opening a **View** window, with the contents of a particular identifier selection, you can add new identifiers to it by dragging and dropping other identifier selections from either the **Model Explorer** or the **Identifier Selector**. Using the **Edit-Delete** menu or the **Del** key, on the other hand, you can delete any subselection of identifiers from the **View** window. At any time you can save the modified identifier selection as a new node in the identifier selector tree through the **View-Selection-Save** menu.

Selecting identifier groups

Besides selecting individual identifiers from the model tree, you can also select whole groups of identifiers by selecting their parent node. For example, if you drag-and-drop an entire declaration section into a **View** window, all the identifiers contained in that section will be added to the view.

Specifying a default view

As can be seen at the bottom of the **View Manager** dialog box in Fig. 2.23, it is possible to associate a default view definition with every selector in the **Identifier Selector**. As a consequence, whenever you double-click on such an identifier selector node, AIMMS will immediately open a default **View** window with the current contents of that selection.

2.5 Debugging and Profiling an AIMMS Model

After you have developed an (optimization) model in AIMMS, it will most probably contain some unnoticed logical and/or programming errors. These errors can cause infeasible solutions or results that are not entirely what you expected. Also, you may find that the execution times of some procedures in your model are unacceptably high for their intended purpose, quite often as the result of only a few inefficiently formulated statements. To help you isolate and resolve such problems, AIMMS offers a number of diagnostic tools, such as a debugger and a profiler, which will be discussed in this chapter.

2.5.1 The AIMMS debugger

Tracking modeling errors

When your model contains logical errors or programming errors, finding the exact location of the offending identifier declarations and/or statements may not be easy. In general, incorrect results might be caused by:

- incorrectly specified attributes for one or more identifiers declared in your model (most notably in the `IndexDomain` and `Definition` attributes),
- logical oversights or programming errors in the formulation of one or more (assignment) statements in the procedures of your model,
- logical oversights or programming errors in the declaration of the variables and constraints comprising a mathematical program, and
- data errors in the parametric data used in the formulation of a mathematical program.

Errors in mathematical programs

If the error is in the formulation or input data of a mathematical program, the main route for tracking down such problems is the use of the **Math Program Inspector** discussed in *The Math Program Inspector* (page 78). Using the **Math Program Inspector** you can inspect the properties of custom selections of individual constraints and/or variables of a mathematical program.

The AIMMS debugger

To help you track down errors that are the result of misformulations in assignment statements or in the definitions of defined parameters in your model, AIMMS provides a *source debugger*. You can activate the AIMMS debugger through the **Tools-Diagnostic Tools-Debugger** menu. This will add a **Debugger** menu to the system menu bar, and, in addition, add the **Debugger** toolbar illustrated in Fig. 2.25 to the toolbar area.

Fig. 2.25: The **Debugger** toolbar

You can stop the AIMMS debugger through the **Debugger-Exit Debugger** menu.

Debugger functionality

Using the AIMMS debugger, you can

- set conditional and unconditional breakpoints on a statement within the body of any procedure or function of your model, as well as on the evaluation of set and parameter definitions,
- step through the execution of procedures, functions and definitions, and
- observe the effect of single statements and definitions on the data within your model, either through tooltips within the observed definitions and procedure bodies, or through separate data pages (see also *Viewing and modifying identifier data* (page 45))

Setting breakpoints in the body

Within the AIMMS debugger you can set breakpoints on any statement in a procedure or function body (or on the definition of a defined set, parameter or variable) by selecting the corresponding source line in the body of the procedure or function, and choosing the **Debugger-Breakpoints-Insert/Remove** menu (or the **Insert/Remove Breakpoint** button  on the **Debugger** toolbar). After you have set a breakpoint, this is made visible by means of red dot in the left margin of selected source line, as illustrated in Fig. 2.26.

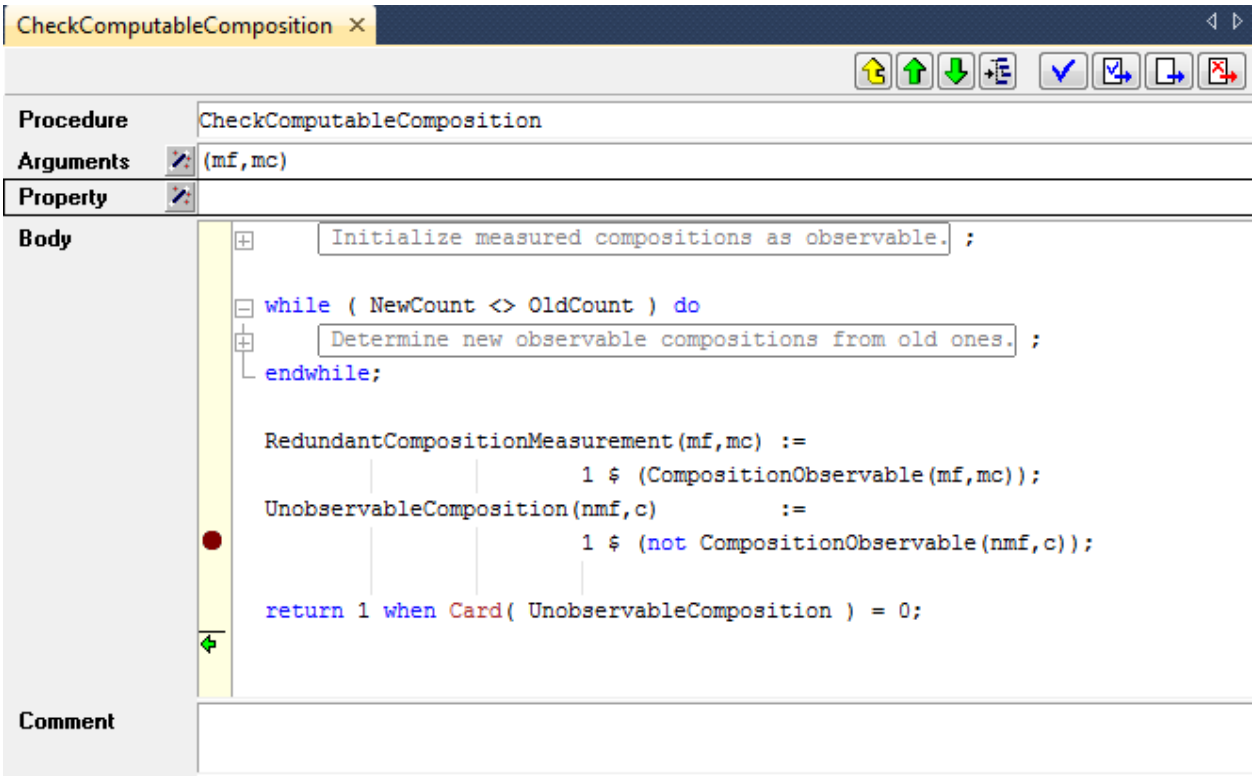


Fig. 2.26: Setting a breakpoint in a procedure body

Setting breakpoints in the model tree

Alternatively, you can set a breakpoint on a procedure, function or on a defined set, parameter or variable by selecting the corresponding node in the **Model Explorer**, and choosing the **Debugger-Breakpoints-Insert/Remove** menu. As a result, AIMMS will add a breakpoint to the first statement contained in the body of the selected procedure or function. The name of a node of any procedure, function or defined set, parameter or variable with a breakpoint is displayed in red in the model tree, as illustrated in Fig. 2.27.

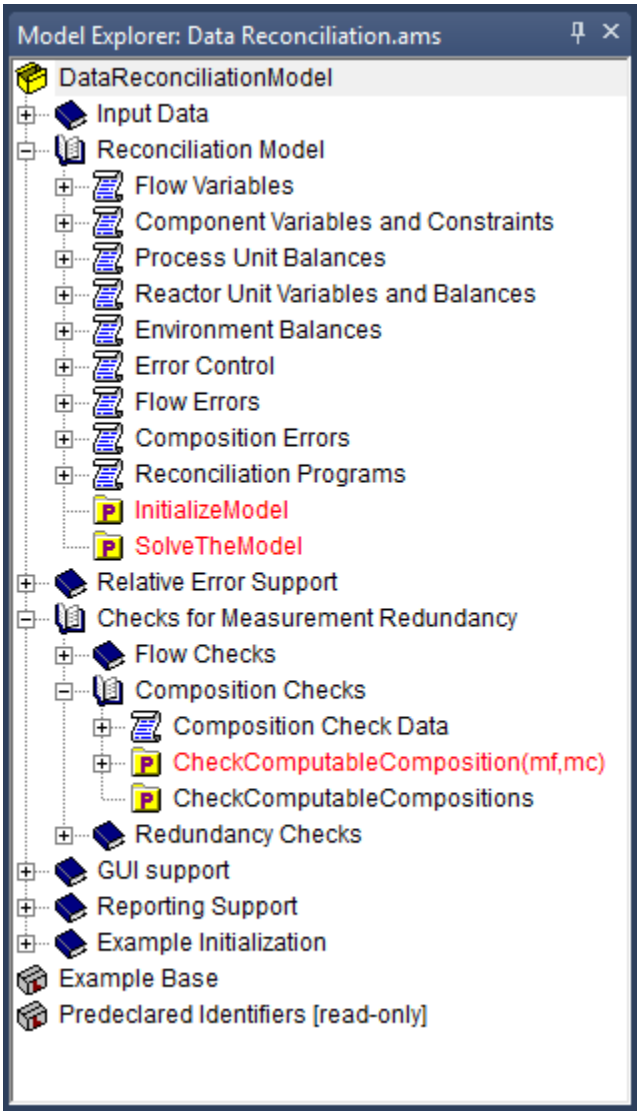


Fig. 2.27: Viewing procedures with breakpoints in the **Model Explorer**

Entering the debugger

Once you have set a breakpoint in your model, AIMMS will automatically stop at this breakpoint whenever a line of execution arrives at the corresponding statement. This can be the result of

- explicitly running a procedure within the **Model Explorer**,
- pushing a button on an end-user page which results in the execution of one or more procedures, or
- opening an end-user (or data) page, which requires the evaluation of a defined set or parameter.

Whenever the execution stops at a breakpoint, AIMMS will open the corresponding procedure body (or the declaration form of the defined set, parameter or variable), and show the current line of execution through the breakpoint pointer, as illustrated in Fig. 2.28.

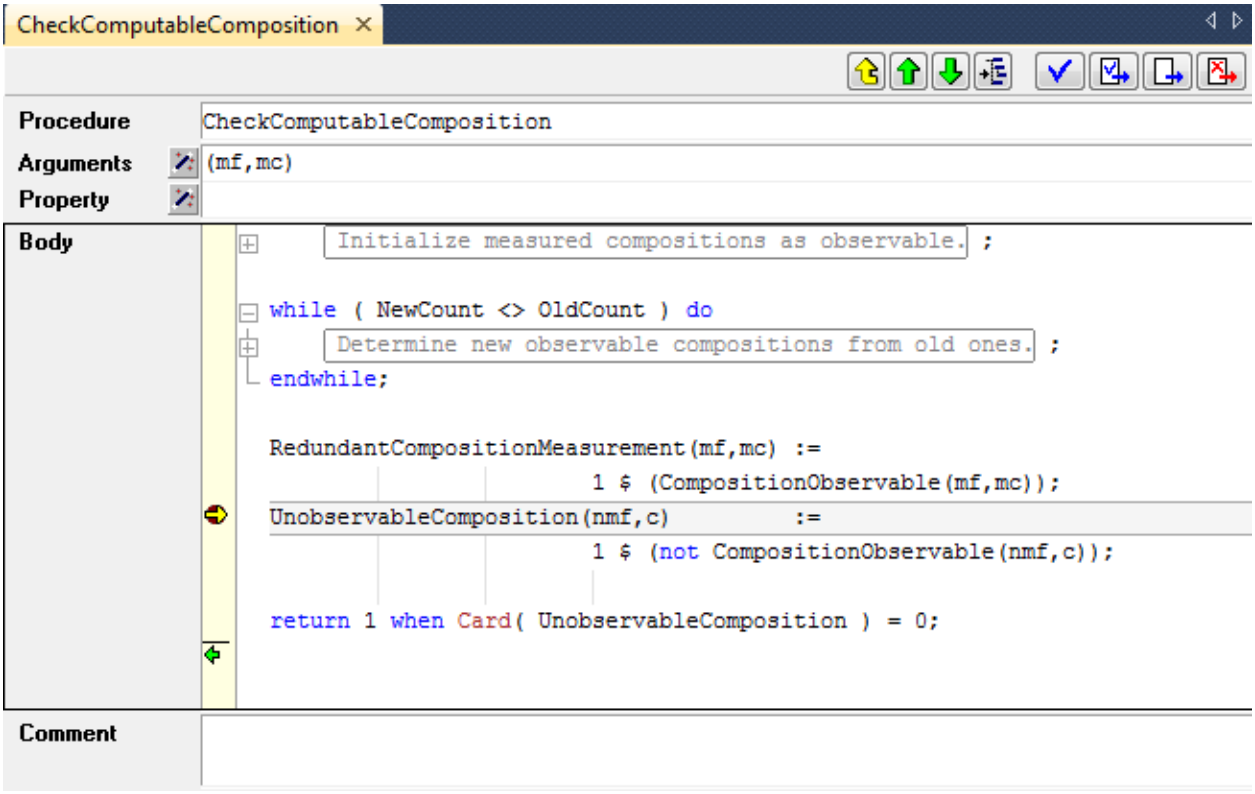


Fig. 2.28: Arriving at a breakpoint

Interrupting execution

Even when you have not set breakpoints, you can still enter the debugger by explicitly *interrupting* the current line of execution through the **Run-Stop** menu (or through the **Ctrl-Shift-S** shortcut key). It will pop up the **Stop Run** dialog box illustrated in Fig. 2.29

When you have activated the AIMMS debugger prior to execution, the **Debug** button on it will be enabled, and AIMMS will enter the debugger when you push it. By pushing the **OK** or **Cancel** button, AIMMS will completely stop or just continue executing, respectively.

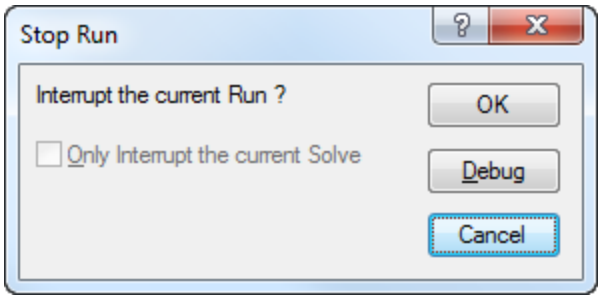


Fig. 2.29: The **Stop Run** dialog box

Interrupting slow statements

The above method of interrupting AIMMS will not work when AIMMS is executing a statement or definition that takes a very long time. In that case you can interrupt AIMMS via the `AimmsInterrupt` tool. This tool is available from the Windows start All Programs menu. Upon startup, it will place itself in the system tray. By right-clicking the AIMMS system tray icon, you'll obtain a menu of running AIMMS instances that can be interrupted. In developer mode, the interrupted AIMMS will also popup a debugger showing where it has been interrupted. With that debugger, you can't continue execution, however; as the consistency of the values of the identifier(s) being computed during the interrupt can't be guaranteed. On the other hand, you can start new procedures. In end-user mode, the interrupted AIMMS will just issue an error message, indicating the interrupted statement, definition or constraint.

Stepping through statements

Once AIMMS has interrupted a line of execution and entered the debugger, you can step through individual statements by using the various step buttons on the **Debugger** toolbar and follow the further flow of execution, or observe the effect on the data of your model. AIMMS offers several methods to step through your code:

- the **Step Over** method runs a single statement, and, when this statement is a procedure call, executes this in its entirety,
- the **Step Into** method runs a single statement, but, when this statement is a procedure call, sets the breakpoint pointer to the first statement in this procedure,
- the **Step Out** method runs to the end of the current procedure and sets the breakpoint pointer to the statement directly following the procedure call in the calling context, and
- the **Run To Cursor** method runs in a single step from the current position of the breakpoint pointer to the current location of the cursor, which should be *within the current procedure*.

In addition, AIMMS offers some methods to continue or halt the execution:

- the **Continue Execution** method continues execution, but will stop at any breakpoint it will encounter during this execution,
- the **Finish Execution** method finishes the current line of execution, ignoring any breakpoints encountered,
- the **Halt** method immediately halts the current line of execution.

Examining identifier data

Whenever you are in the debugger, AIMMS allows you to interactively examine the data associated with the identifiers in your model, and observe the effect of statements in your source code. The most straightforward method is by simply moving the mouse pointer over a reference to an identifier (or identifier *slice*) within the source code of your model. As a result, AIMMS will provide an overview of the data contained in that identifier (slice) in the form of a tooltip, as illustrated in Fig. 2.30.

The tooltip will provide global information about the identifier slice at hand, such as

- its name and indices,
- the number of elements or non-default data values (in brackets), and
- the first few elements or non-default data value in the form of a list consisting of tuples and their corresponding values.

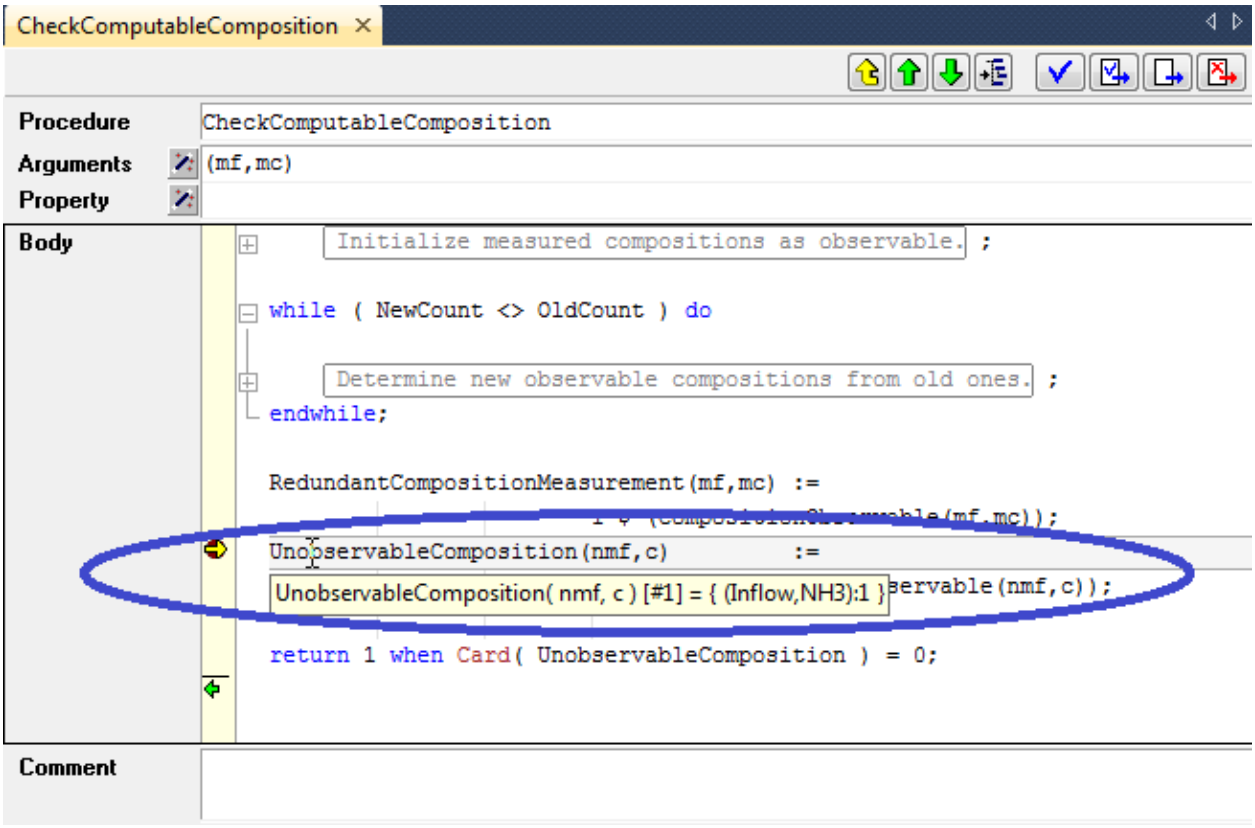


Fig. 2.30: Observing the current data of an identifier through a tooltip

Detailed identifier data

If you need to examine the effect of a statement on the data of a particular identifier in more detail, you can simply open a **Data Page**, as described in *Viewing and modifying identifier data* (page 45), or observe the effect on ordinary end-user pages. Within a debugger session, AIMMS supports data pages for both global and local identifiers, thereby allowing you to examine the contents of local identifiers as well. After each step in the debugger AIMMS will automatically update the data on any open end-user or data page.

Breakpoint on data change

If you are not sure which statement in your model is responsible for changing the data of a (non-defined) set or parameter, you can set a breakpoint on such a set or parameter. Whenever a statement in your model changes the set or parameter data at hand, AIMMS will break on that statement. Notice, however, that breakpoint on data change will not pick up data changes that are due to set or parameter data becoming inactive because of changes to sets or parameters included in the domain or domain condition.

Viewing the call stack

Whenever you are in the debugger, the **Call Stack** button  on the **Debugger** toolbar will display the **Call Stack** dialog box illustrated in Fig. 2.31.

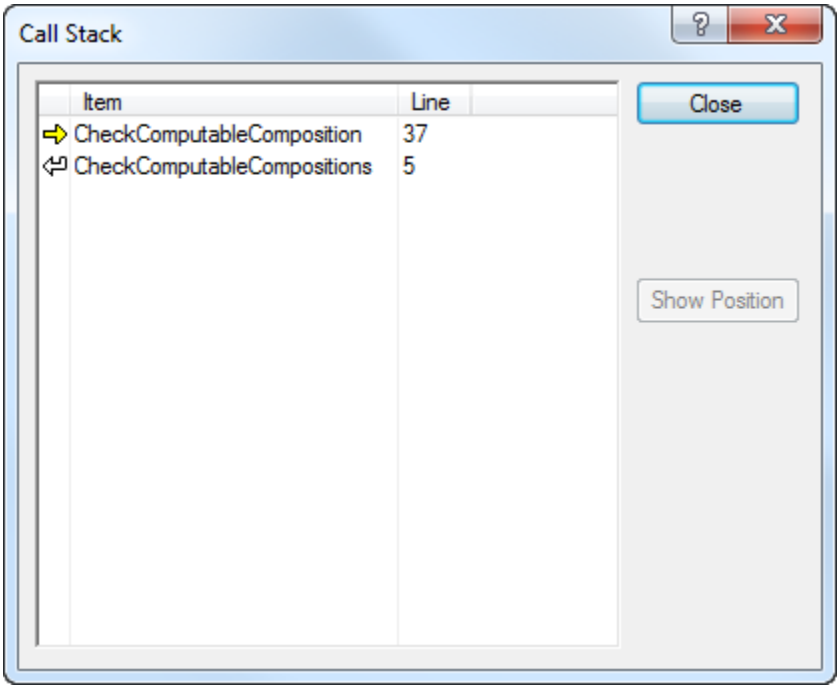


Fig. 2.31: The **Call Stack** dialog box

With it you get a detailed overview of the stack of procedure calls associated with the current line of execution. It enables you to observe the flow of execution at the level of procedures associated with the current position of the breakpoint pointer. After selecting a procedure or definition in the **Call Stack** dialog box, the **Show Position** button will open its attribute window at the indicated line.

Viewing and modifying breakpoints

After you have inserted a number of breakpoints into your model, you can get an overview of all breakpoints through the **Show All Breakpoints** button  . This button will invoke the **List of Breakpoints** dialog box illustrated in Fig. 2.32.

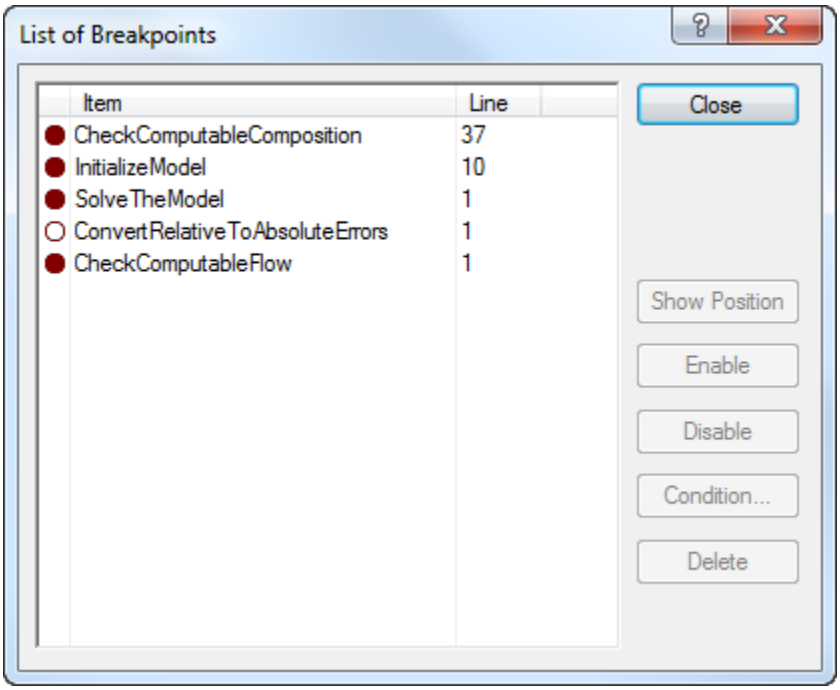


Fig. 2.32: The **List of Breakpoints** dialog box

For each breakpoint, AIMMS will indicate whether it is enabled or disabled (i.e. to be ignored by the AIMMS debugger). Through the buttons on the right hand side of the dialog box you can

- disable breakpoints,
- enable previously disabled breakpoints,
- delete breakpoints, and
- create new breakpoints.

Alternatively, you can disable or remove all breakpoints simultaneously using the **Disable All Breakpoints** button  and the **Remove All Breakpoints** button  .

Conditional breakpoints

In addition, by pushing the **Condition** button on the **List of Breakpoints** dialog box, you can add a condition to an existing breakpoint. It will open the **Breakpoint Condition** dialog box illustrated in Fig. 2.33.

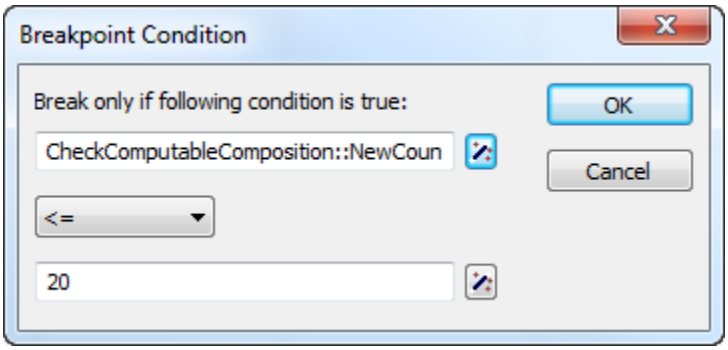


Fig. 2.33: The **Breakpoint Condition** dialog box

The condition must consist of a simple numerical, element or string comparison. This simple comparison can only involve scalar identifiers, identifier slices or constants. Free indices in an identifier slice are only allowed when they are fixed within the breakpoint context (e.g. through a for loop). AIMMS will only stop at a conditional breakpoint, when the condition that you have specified is met during a particular call. Conditional breakpoints are very convenient when, for instance, a procedure is called very frequently, but only appears to contain an error in one particular situation which can be detected through a simple comparison.

2.5.2 The AIMMS profiler

Meeting time requirements with solvers

Once your model is functionally complete, you may find that the overall computational time requirement set for the application is not met. If your application contains optimization, and most of the time is spent by the solver, finding a remedy for the observed long solution times may not be easy. In general, it involves finding a reformulation of the mathematical program which is more suitable to the selected solver. Finding such a reformulation may require a considerable amount of expertise in the area of optimization.

Meeting time requirements with data execution

It could also be, however, that optimization (if any) only consumes a small part of the total execution time. In that case, the time required for executing the application is caused by data manipulation statements. If total execution time is unacceptably high, it could be caused by inefficiently formulated statements. Such statements force AIMMS to fall back to *dense* instead of *sparse* execution. The AIMMS Sparse Execution Engine and Execution Efficiency Cookbook of the Language Reference discuss the principles of the sparse execution engine used by AIMMS, and describe several common pitfalls together with reformulations to remedy them.

The AIMMS profiler

AIMMS offers a profiler to help you resolve computational time related issues. The AIMMS profiler enables you to locate the most time-consuming evaluations of

- procedures and functions,
- individual statements within procedures and functions,
- defined sets and parameters, and
- constraints and defined variables during matrix generation.

Activating the profiler

You can activate the AIMMS profiler by selecting the **Tools-Diagnostic Tools-Profiler** menu, which will add a **Profiler** menu to the default system menu bar. If the debugger is still active at this time, it will be automatically deactivated, as both tools cannot be used simultaneously.

Gathering timing information

As soon as you have activated the profiler, AIMMS will start gathering timing information during every subsequent procedure run or definition evaluation, regardless whether these are initiated by pushing a button on an end-user page, by executing a procedure from within the **Model Explorer**, or even by means of a call to the AIMMS API from within an external DLL.

Viewing profiler results

After you have gathered timing information about your modeling application by executing the relevant parts of your application at least once, you can get an overview of the timing results through the **Profiler-Results Overview** menu. This will open the **Profiler Results Overview** dialog box illustrated in Fig. 2.34.

In it, you will find a list of all procedures that have been executed and identifier definitions that have been evaluated since the profiler was activated.

Detailed timing information

For each procedure, function, or defined identifier listed in the **Profiler Results Overview** dialog box, AIMMS will provide the following information:

- the number of hits (i.e. the number of times a procedure has been executed or a definition has been evaluated),
- the total gross time (explained below) spent during all hits,
- the total net time (explained below) spent during all hits,
- the average gross time spent during each separate hit, and
- the average net time spent during each separate hit.

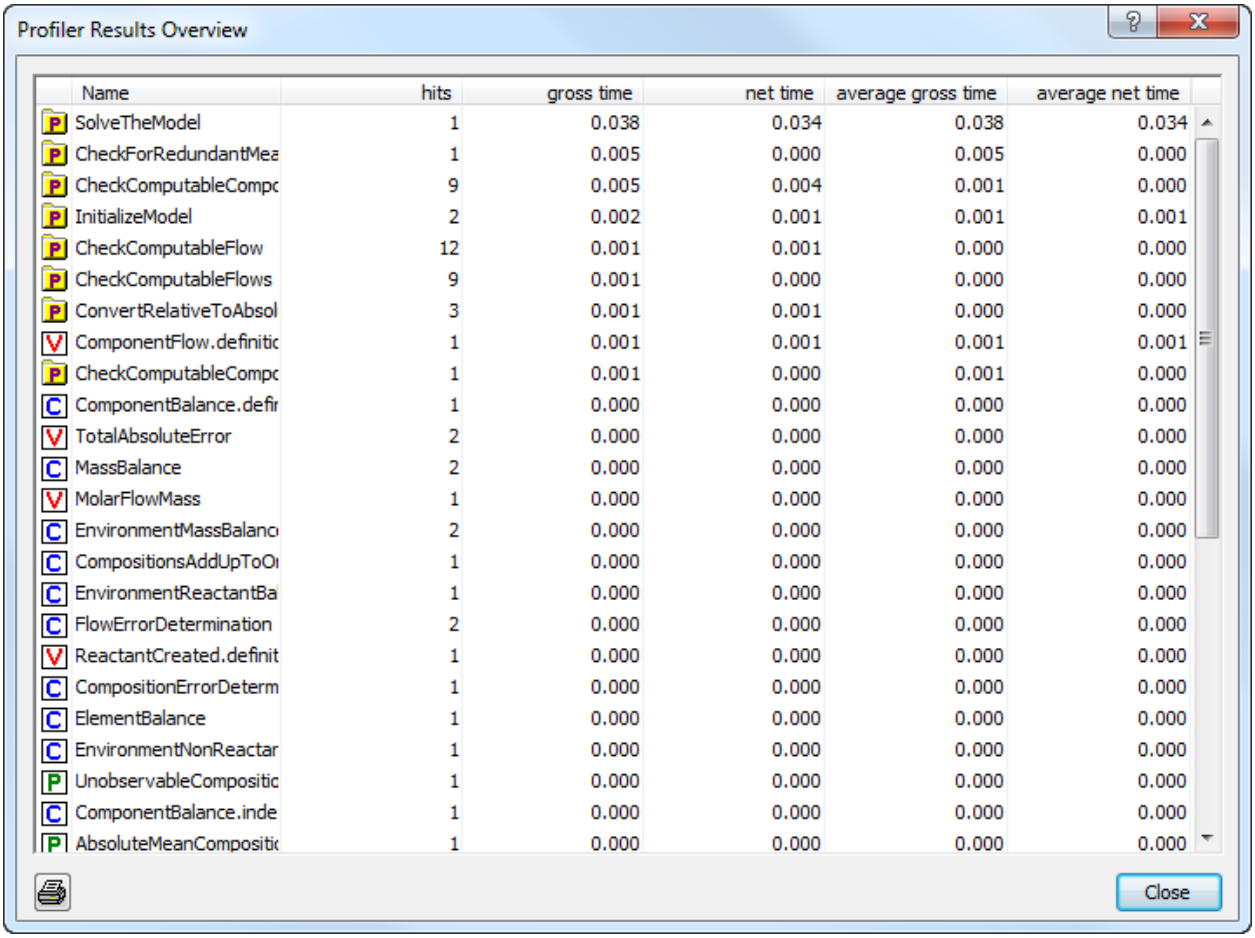


Fig. 2.34: The **Profiler Results Overview** dialog box

Gross versus net time

The term *gross time* refers to the total time spent in a procedure *including* the time spent in procedure calls or definition evaluations within the profiled procedure. The term *net time* refers to the total time spent *excluding* the time spent in procedure calls or definition evaluations within the profiled procedure.

Locating time-consuming procedures

With this timing information you can try to locate the procedures and identifier definitions which are most likely to benefit from a reformulation to improve efficiency. To help you locate these procedures and definitions, the list of procedures and definitions in the **Profiler Results Overview** dialog box can be sorted with respect to all its columns. The most likely criterion for this is to sort by decreasing net time or average net time, which will identify those procedures and identifier definitions which take up the most time by themselves, either in total or for each individual call. You can open the attribute form of any identifier in the **Profiler Results Overview** dialog box by simply double-clicking on the corresponding line.

Locating offending statements

When you have located a time-consuming procedure, you can can open its attribute form and try to locate the offending statement(s). Whenever the profiler has been activated, AIMMS will add additional profiling columns to the body of a procedure, as illustrated in Fig. 2.35.

Similarly, AIMMS will add these profiling columns to the definition attributes of defined identifiers.

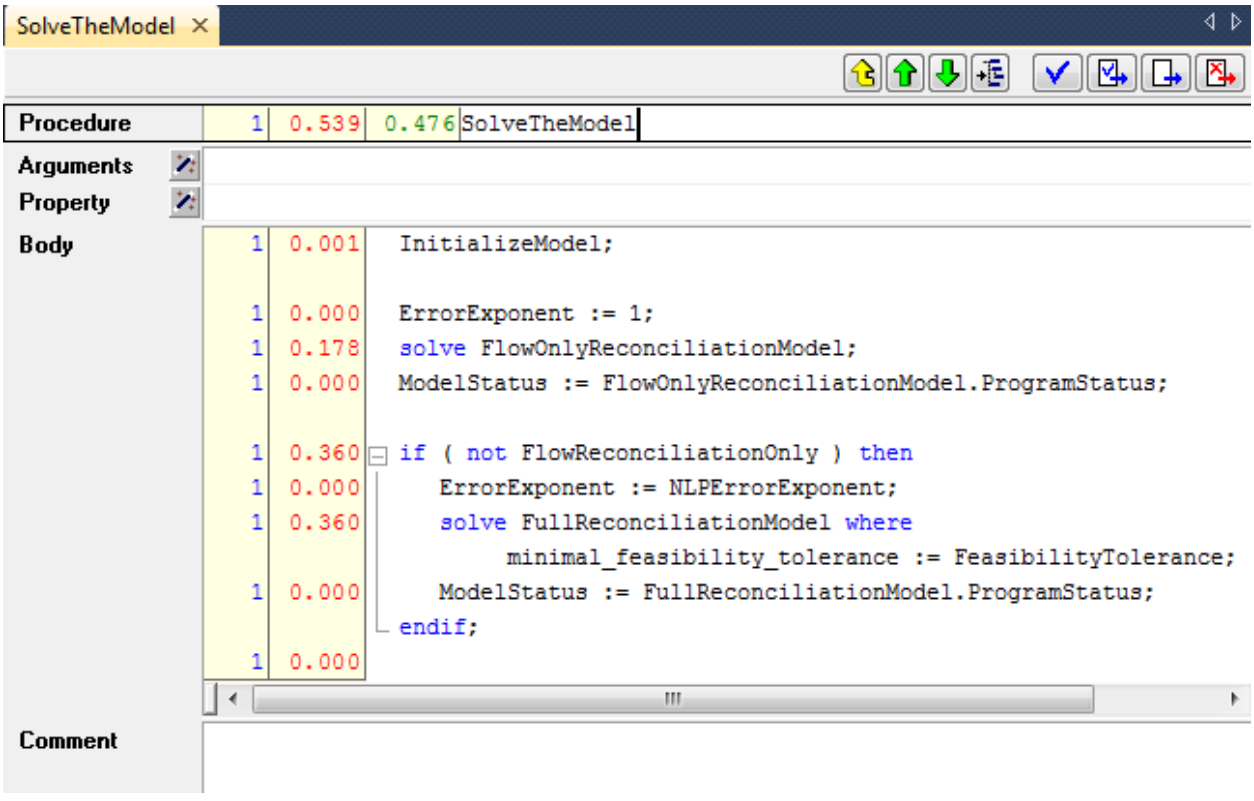


Fig. 2.35: Profiling information in an attribute form

Profiling column information

For each statement in the body of a procedure, AIMMS can display various types of profiling data in the profiling columns of an attribute form. As you can see next, this information is even more extensive than for procedures as a whole. The following information is available:

- the number of hits (i.e. the number of times a particular statement has been executed),
- the total gross time spent during all hits,
- the total net time spent during all hits,
- the average gross time spent during each separate hit, and
- the average net time spent during each separate hit.

Gross versus net time for particular statements

In the context of a procedure body, the difference between gross and net time need not always refer only to the time spent in other procedures (as in the **Profiler Results Overview** dialog box). For selected statements both numbers may have a somewhat different, yet meaningful, interpretation. Some exceptions:

- in flow control statements such as the `IF`, `WHILE` and `FOR` statement (see also Flow Control Statements of the Language Reference), the net time refers to the time required to evaluate the statement itself (for instance, its condition) whereas the gross time refers to the time required to execute the entire statement,
- in the `SOLVE` statement (see also The SOLVE Statement of the Language Reference), the net time refers to the time spent in the solver, while the gross time refers to the time spent in the solver plus the time required to generate the model.
- in a procedure call itself the net time refers to the time spent in argument passing.

Profiler tooltips

In addition to observing the profiling times in the **Profiler Results Overview** and through the profiling columns in attribute windows, AIMMS also provides profiling tooltips in both the **Model Explorer** and in the attribute windows of procedures and defined identifiers, as long as the profiler is active. An example of a profiling tooltip is given in Fig. 2.36.

Profiling tooltips can provide a convenient method to quickly observe the profiling information without requiring any further interaction with AIMMS. If you do not want AIMMS to display profiling tooltips while moving your mouse through either the **Model Explorer** or procedure bodies, you can disable them through the **Profiler Setup** dialog box described below, by unchecking the **Show Profiler Values** check mark (see also Fig. 2.37).

Profiler listing

If you are interested in a profiling overview comprising your entire modeling application, you can get this through the **Profiler-Create Listing File** menu. This will create a common source listing file of your model text extended with profiling information wherever this is available. Through the **Profiler Setup** dialog box described below you can determine which profiler information will be added to the profiler listing.

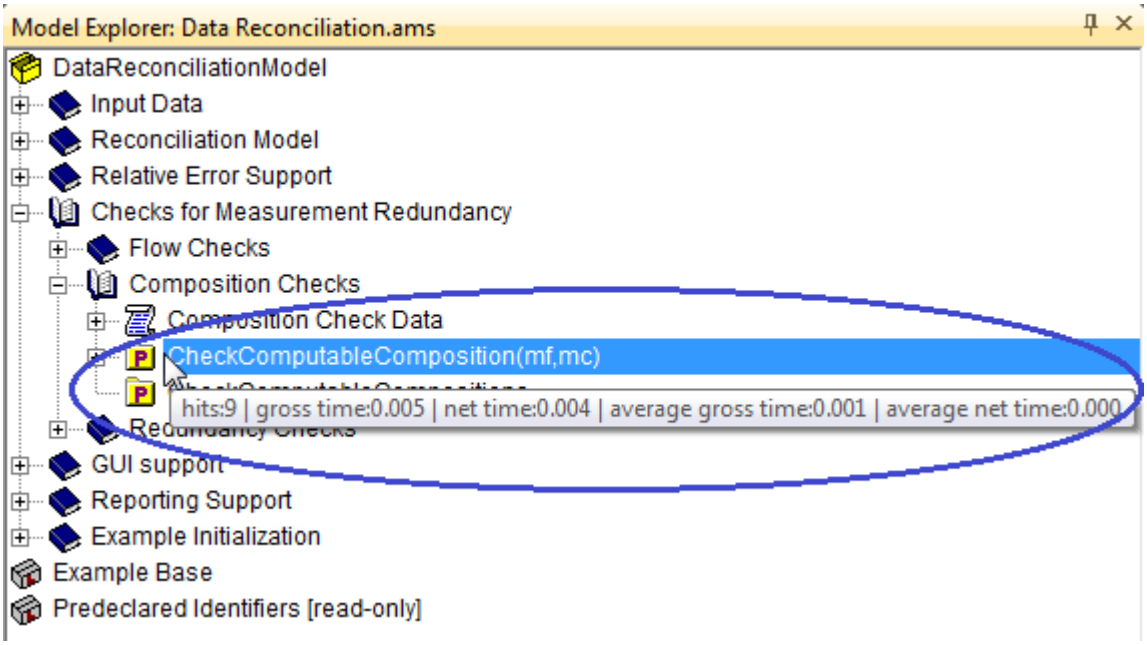


Fig. 2.36: Observing profiling information through a tooltip

Setting up profiling columns

For every new project, AIMMS uses a set of default settings to determine which profiling information is displayed in the various available methods to display profiling information. You can modify these settings through the **Profiler-Setup** menu, which will open the **Profiler Setup** dialog box illustrated in Fig. 2.37.

In this dialog box you can, on a project-wide basis, determine

- which of the profiling display methods described are enabled (through the **Show Profiler Values** check mark), and
- per such display method, which profiling information is to be displayed, their order, and their corresponding display format.

The settings selected in the **Profiler Setup** dialog box are saved along within the project file, and will be restored when you reopen the project in another AIMMS session.

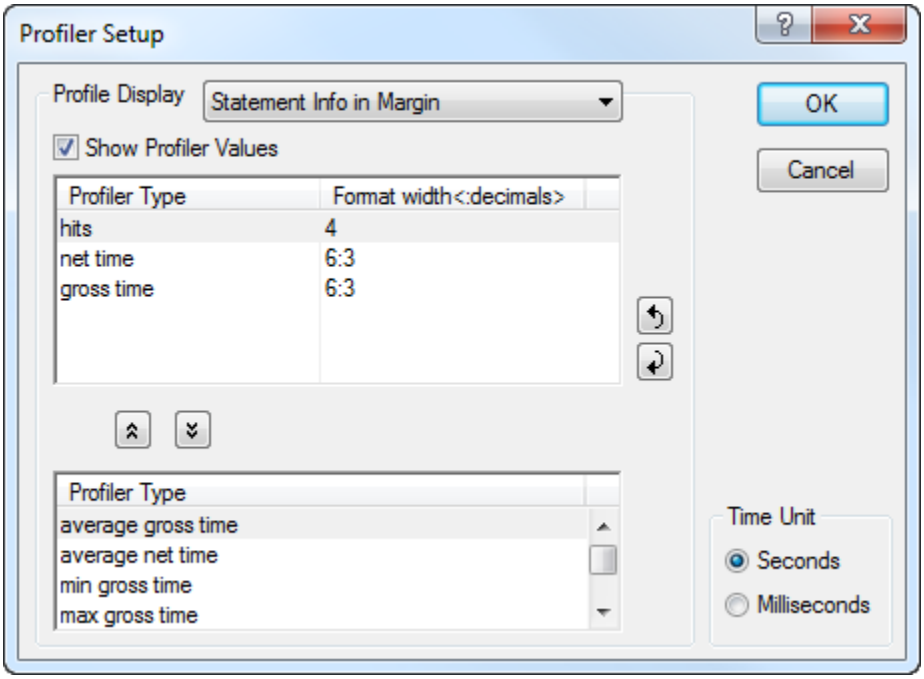


Fig. 2.37: The **Profiler Setup** dialog box

Pausing and continuing the profiler

Through the **Profiler-Pause** menu, you can temporarily halt the gathering of profiling information by AIMMS, while the existing profiling information will be retained. You can use this menu, for example, when you only want to profile the core computational procedures of your modeling application and do not want the profiling information to be cluttered with profiling information that is the result of procedure calls and definition evaluations in the end-user interface. You can resume the gathering of profiling information through the **Profiler- Continue** menu.

Resetting the profiler

With the **Profiler-Reset** menu, you can completely reset all profiling counters to zero. You can use this menu, if the profiling information of your application has become cluttered. For instance, some procedures may have been executed multiple times and, thus, disturb the typical profiling times required by your entire application. After resetting the profiling counters, you can continue to gather new profiling information which will then be displayed in the various profiling displays.

Exiting the profiler

You can completely disable the AIMMS profiler through the **Profiler-Exit Profiler** menu. As a result, the gathering of profiling information will be completely discontinued, and all profiling counters will be reset to zero. Thus, when you restart the profiler, all profiling information of a previous session will be lost.

2.5.3 Observing identifier cardinalities

Observing identifier cardinalities

Another possible cause of performance problems is when one or more multi- dimensional identifiers in your model have missing or incorrectly specified domain conditions. As a result, AIMMS could store far too much data for such identifiers. In addition, computations with these identifiers may consume a disproportional amount of time.

Locating cardinality problems

To help you locate identifiers with missing or incorrectly specified domain conditions, AIMMS offers the **Identifier Cardinalities** dialog box illustrated in Fig. 2.38. You can invoke it through the **Tools-Diagnostic Tools-Identifier Cardinalities** menu.

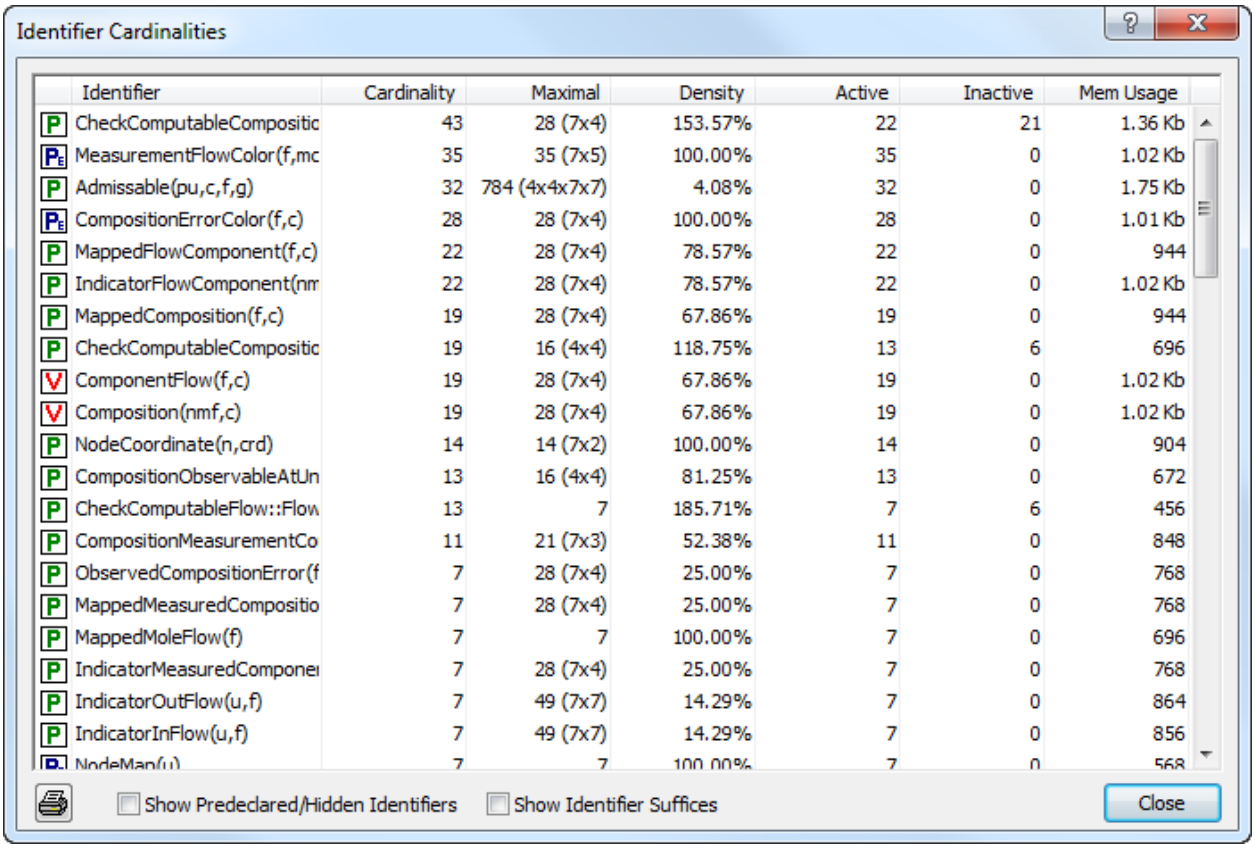


Fig. 2.38: The **Identifier Cardinalities** dialog box

Available information

The **Identifier Cardinalities** dialog box displays the following information for each identifier in your model:

- the *cardinality* of the identifier, i.e. the total number of non-default values currently stored,
- the *maximal cardinality*, i.e. the cardinality if all values would assume a non-default value,
- the *density*, i.e. the cardinality as a percentage of the maximal cardinality,
- the number of *active* values, i.e. of elements that lie within the domain of the identifier,
- the number of *inactive* values, i.e. of elements that lie outside of the domain of the identifier, and
- the *memory usage* of the identifier, i.e. the amount of memory needed to store the identifier data.

The list of identifier cardinalities can be sorted with respect to any of these values.

Locating dense data storage

You can locate potential dense data storage problems by sorting all identifiers by their cardinality. Identifiers with a very high cardinality and a high density can indicate a missing or incorrectly specified domain condition. In most real-world applications, the higher-dimensional identifiers usually have relatively few tuples, as only a very small number of combinations have a meaningful interpretation.

Resolving dense storage

If your model contains one or more identifiers that appear to demonstrate dense data storage, it is often possible to symbolicly describe the appropriate domain of allowed tuple combinations. Adding such a condition can be helpful to increase the performance of your model by reducing both the memory usage and execution times.

Locating inactive data

Another type of problem that you can locate with the **Identifier Cardinalities** dialog box, is the occurrence of *inactive* data in your model. Inactive data can be caused by

- the removal of elements in one or more domain sets, or
- data modifications in the identifier(s) involved in the domain restriction of the identifier.

Problems with inactive data

In principle, inactive data does not directly influence the behavior of your model, as the AIMMS execution engine itself will never consider inactive data. Inactive data, however, can cause unexpected problems in some specific situations.

- One of the areas where you have to be aware about inactive data is in the AIMMS API (see also The AIMMS Programming Interface of the Language Reference), where you have to decide whether or not you want the AIMMS API to pass inactive data to an external application or DLL.
- Also, when inactive data becomes active again, the previous values are retained, which may or may not be what you intended.

As a result, the occurrence of inactive data in the **Identifier Cardinalities** dialog box may make you rethink its consequences, and may cause you to add statements to your model to remove the inactive data explicitly (see also Data Control of the Language Reference).

2.6 The Math Program Inspector

In this chapter you will find the description of an extensive facility to analyze both the input and output of a generated optimization model. This mathematical program inspector allows you to make custom selections of constraints and variables. For each such selection, you can inspect statistics of the corresponding matrix and the corresponding solution. The main purpose of the math program inspector, however, is to assist you in finding causes of infeasibility, unboundedness and unrealistic solutions.

Acknowledgement

The design and contents of the math program inspector is strongly influenced by the papers and contributions of Bruce A. McCarl on misbehaving mathematical programs. The example in the last section of this chapter is a direct reference to his work.

2.6.1 Introduction and motivation

Unforeseen surprises ...

Even though you have taken the utmost care in constructing your linear optimization model, there are often unforeseen surprises that force you to take a further look at the particular generated model at hand. Why is the model infeasible or unbounded? Why is the objective function value so much different from what you were expecting? Are the individual constraints generated as intended? Why is the number of individual constraints so large? Why are the observed shadow prices so unrealistically high (or low)? These and several other related questions about the matrix and the solution need further investigation.

... are not easily explained

The answer to many model validation questions is not easily discovered, especially when the underlying optimization model has a large number of individual constraints and variables. The amount of information to be examined is often daunting, and an answer to a question usually requires extensive analysis involving several steps. The functionality of the math program inspector is designed to facilitate such analysis.

Some of the causes

There are many causes of unforeseen surprises that have been observed in practice. Several are related to the values in the matrix. Matrix input coefficients may be incorrect due to a wrong sign, a typing error, an incorrect unit of measurement, or a calculation flaw. Bounds on variables may have been omitted unintentionally. Other causes are related to structural information. The direction of a constraint may be accidentally misspecified. The subsets of constraints and variables may contain incorrect elements causing either missing blocks of constraints and variables, or unwanted blocks. Even if the blocks are the correct ones, their index domain restrictions may be missing or incorrect. As a result, the model may contain unwanted and/or missing constraints and/or variables.

Purpose

The purpose of the mathematical program inspector included in AIMMS is to provide you with

- insight into the (structure of the) generated model and its solution (if present), and
- a collection of tools to help you discover errors in your model ,

2.6.2 Functional overview

This section

In this section you will find a description of the functionality available in the mathematical program inspector. Successively, you will learn

- the basics of the trees and windows available in the mathematical program inspector,
- how you can manipulate the contents of the variable and constraint trees through variable and constraint properties, but also using the **Identifier Selector** tool,
- how to inspect the contents and properties of the matrix and solution corresponding to your mathematical program, and
- which analysis you can perform using the mathematical program inspector when your mathematical program is infeasible.

Tree View Basics

Viewing generated variables and constraints

The math program inspector window displays the set of all generated variables and all generated constraints, each in a separate tree (see the left portion of Fig. 2.39).

In these trees, the symbolic identifiers are the first-level nodes and on every subsequent level in the tree, one or more indices are fixed. As a result, the individual variables and constraints in your model appear as leaf nodes in the two tree view windows.

Tree view selections

The math program inspector contains several tabs (see the right portion of Fig. 2.39) that retrieve information regarding the selection that has been made in the tree views. Common Windows controls are available to select a subset of variables and constraints (mouse click possibly in combination with the or key). Whenever you select a slice (i.e. an intermediate node in the tree) all variables or constraints in that subtree are selected implicitly. You can use the **Next Leaf Node** and **Previous Leaf Node** buttons in the toolbar for navigational purposes. In Fig. 2.39 a single variable has been selected in the variable tree. In addition, the text on most tabs can by copied to the Windows clipboard using the familiar **Ctrl-C** shortcut key.

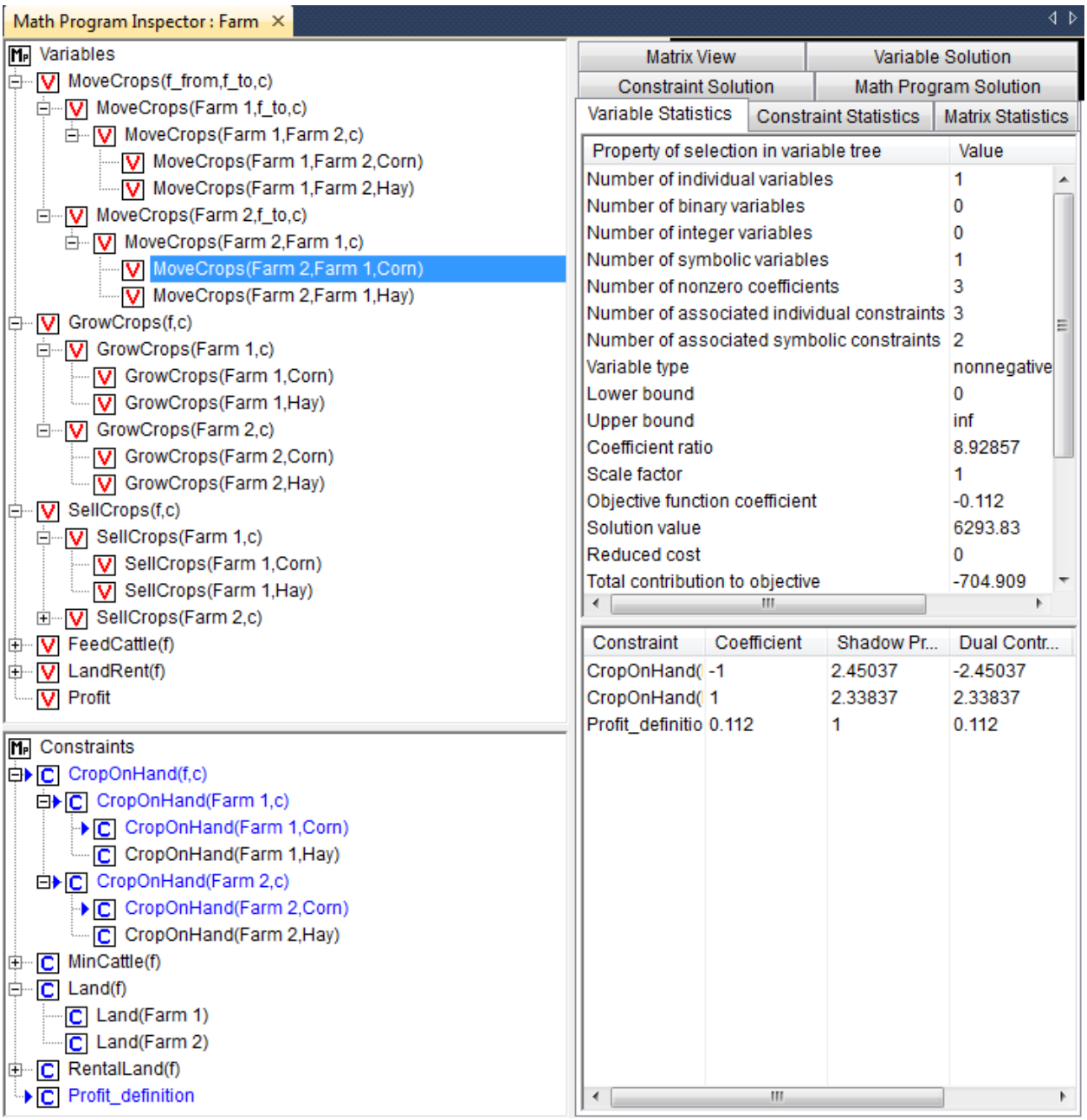


Fig. 2.39: The math program inspector window

Bookmarks

Bookmarks allow you to temporarily tag one or more variables or constraints. While navigating through the tree you will always change the current selection, while the bookmarked nodes will not be affected. Whenever you bookmark a node, all its child nodes plus parent nodes are also bookmarked. Using the **Bookmarks** menu you can easily select all bookmarked nodes or bookmark all selected nodes. Bookmarks appear in blue text. Fig. 2.39 contains a constraint tree with three bookmarked constraints plus their three parent nodes. You can use the **Next Bookmark** and **Previous Bookmark** buttons in the toolbar for navigational purposes. In case your Fig. 2.39 is not displayed in color, the light-gray print indicates the bookmarked selection.

Domain index display order

By default only one index is fixed at every level of the tree views, and the indices are fixed from the left to the right. However, you can override the default index order as well as the subtree depth by using the **Variable Property** or **Constraint Property** dialog on the first-level nodes in the tree. The subtree depth is determined by the number of distinct index groups that you have specified in the dialog.

Finding associated variables/ constraints

The linkage between variables and constraints in your model is determined through the individual matrix coefficients. To find all variables that play a role in a particular constraint selection, you can use the **Associated Variables** command to bookmark the corresponding variables. Similarly, the **Associated Constraints** command can be used to find all constraints that play a role in a particular variable selection. In Fig. 2.39, the associated constraint selection for the selected variable has been bookmarked in the constraint tree.

Advanced Tree Manipulation

Variable and constraint properties

Using the right-mouse popup menu you can access the **Variable Properties** and **Constraint Properties**. On the dialog box you can specify

- the domain index display order (already discussed above), and
- the role of the selected symbolic variable or constraint during infeasibility and unboundedness analysis.

Variable and constraint statistics

The math program inspector tool has two tabs to retrieve statistics on the current variable and constraint selection. In case the selection consists of a single variable or constraint, all coefficients in the corresponding column or row are also listed. You can easily access the variable and constraint statistics tabs by double-clicking in the variable or constraint tree. Fig. 2.39 shows the variable statistics for the selected variable.

Popup menu commands

In addition to **Variable Properties** and **Constraint Properties**, you can use the right-mouse popup menu to

- open the attribute form containing the declaration of an identifier,
- open a data page displaying the data of the selected slice,
- make a variable or constraint at the first level of the tree inactive (i.e. to exclude the variable or constraint from the generated matrix during a re-solve), and
- bookmark or remove the bookmark of nodes in the selected slice.

Interaction with identifier selector

Using the identifier selector you can make sophisticated selections in the variable and/or constraint tree. Several new selector types have been introduced to help you investigate your mathematical program. These new selector types are as follows.

element-dependency selector The element-dependency selector allows you to select all individual variables or constraints for which one of the indices has been fixed to a certain element.

scale selector The scale selector allows you to find individual rows or columns in the generated matrix that may be badly scaled. The selection coefficient for a row or column introduced for this purpose has been defined as

$$\frac{\text{largest absolute (nonzero) coefficient}}{\text{smallest absolute (nonzero) coefficient}}.$$

The **Properties** dialog associated with the scale selector offers you several possibilities to control the determination of the above selection coefficient.

status selector Using the status selector you can quickly select all variables or constraints that are either basic, feasible or at bound.

value selector The value selector allows you to select all variables or constraints for which the value (or marginal value) satisfies some simple numerical condition.

type selector With the type selector you can easily filter on variable type (e.g. continuous, binary, nonnegative) or constraint type (e.g. less-than- or-equal, equal, greater-than-or-equal). In addition, you can use the type selector to filter on nonlinear constraints.

Inspecting Matrix Information

Variable Statistics tab

Most of the statistics that are displayed on the Variable Statistics tab are self-explanatory. Only two cases need additional explanation. In case a single symbolic (first-level node) has been selected, the *Index domain density* statistic will display the number of actually generated variables or constraints as a percentage of the full domain (i.e. the domain without any domain condition applied). In case a single variable (a leaf node) has been selected, the statistics will be extended with some specific information about the variable such as bound values and solution values.

Column coefficients

In case a single variable x_j has been selected, the lower part of the information retrieved through the Variable Statistics tab will contain a list with all coefficients a_{ij} of the corresponding rows i, together with the appropriate shadow prices y_i (see Fig. 2.39). The last column of this table will contain the dual contributions $a_{ij}y_j$ that in case of a linear model together with the objective function coefficient c_j make up the reduced cost $\bar{c}_j$ according to the following formula.

$$\bar{c}_j = c_j - \sum_i a_{ij} y_i$$

Nonlinear coefficients

Coefficients of variables that appear in nonlinear terms in your model are denoted between square brackets. These numbers represent the linearized coefficients for the current solution values.

Constraint Statistics tab

The Constraints Statistics tab and the Variable Statistics tab retrieve similar statistics. Fig. 2.40 shows the constraint statistic for the selection consisting of a single constraint. Note that in this particular case the symbolic form of the constraint definition will also be displayed. In case the selected constraint is nonlinear, the individual nonlinear constraint as generated by AIMMS and communicated to the solver is also displayed.

Row coefficients

In case a single row i has been selected, the lower part of the Constraint Statistics tab will contain all coefficients a_{ij} in the corresponding columns j, together with their level values x_j. The last column of this table lists the primal contributions $a_{ij}x_j$ that together in case of a linear model with the right-hand-side make up either the slack or surplus that is associated with the constraint according to the following formula.

$$\text{slack}_i - \text{surplus}_i = \text{rhs}_i - \sum_j a_{ij} x_j$$

Nonlinear constraints

As is the case on the Variable Statistics Tab, all coefficients corresponding to nonlinear terms are denoted between square brackets. For these coefficients, the last column displays all terms that contribute to the linearized coefficient value.

Matrix Statistics tab

The Matrix Statistics tabs retrieves information that reflects both the selection in the variable tree and the selection in the constraint tree. Among these statistics are several statistical moments that might help you to locate data outliers (in terms of size) in a particular part of the matrix.

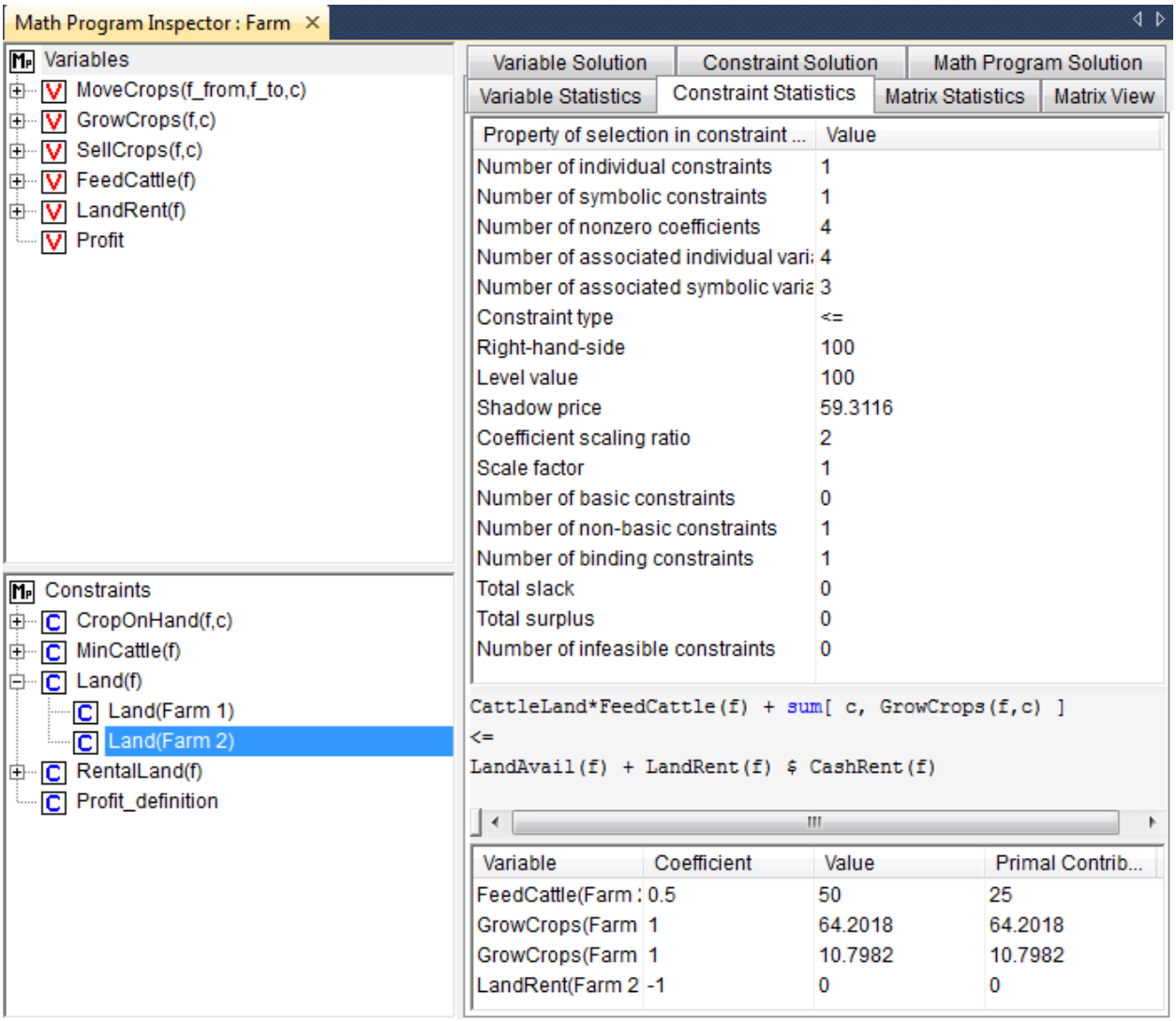


Fig. 2.40: The math program inspector window

Matrix View tab

The Matrix View tab contains a graphical representation of the generated matrix. This view is available in two modes that are accessible through the right-mouse popup menu. The symbolic block view displays at most one block for every combination of symbolic variables and symbolic constraints. The individual block view allows you to zoom in on the symbolic view and displays a block for every nonzero coefficient in the matrix. It is interesting to note that the order in which the symbolic and individual variables and constraints are displayed in the block view follows the order in which they appear in the trees.

Block coloring

The colors of the displayed blocks correspond to the value of the coefficient. The colors will vary between green and red indicating small and large values. Any number with absolute value equal to one will be colored green. Any number for which the absolute value of the logarithm of the absolute value exceeds the logarithm of some threshold value will be colored red. By default, the threshold is set to 1000, meaning that all nonzeros $x \in (-\infty, -1000] \cup [-\frac{1}{1000}, \frac{1}{1000}] \cup [1000, \infty)$ will be colored red. All numbers in between will be colored with a gradient color in the spectrum between green and red.

Block patterns

Any block that contains at least one nonlinear term will show a hatch pattern showing diagonal lines that run from the upper left to the lower right of the block.

AIMMS option

The value of the threshold mentioned in the previous paragraph is available as an AIMMS option with name `bad_scaling_threshold` and can be found in the **Project** - **Math program inspector** category in the **AIMMS Options** dialog box.

Block tooltips

While holding the mouse inside a block, a tooltip will appear displaying the corresponding variables and constraints. In the symbolic view the tooltip will also contain the number of nonzeros that appear in the selected block. In the individual view the actual value of the corresponding coefficient is displayed.

Block view features

Having selected a block in the block view you can use the right-mouse popup menu to synchronize the trees with the selected block. As a result, the current bookmarks will be erased and the corresponding selection in the trees will be bookmarked. Double-clicking on a block in symbolic mode will zoom in and display the selected block in individual mode. Double-clicking on a block in individual mode will center the display around the mouse.

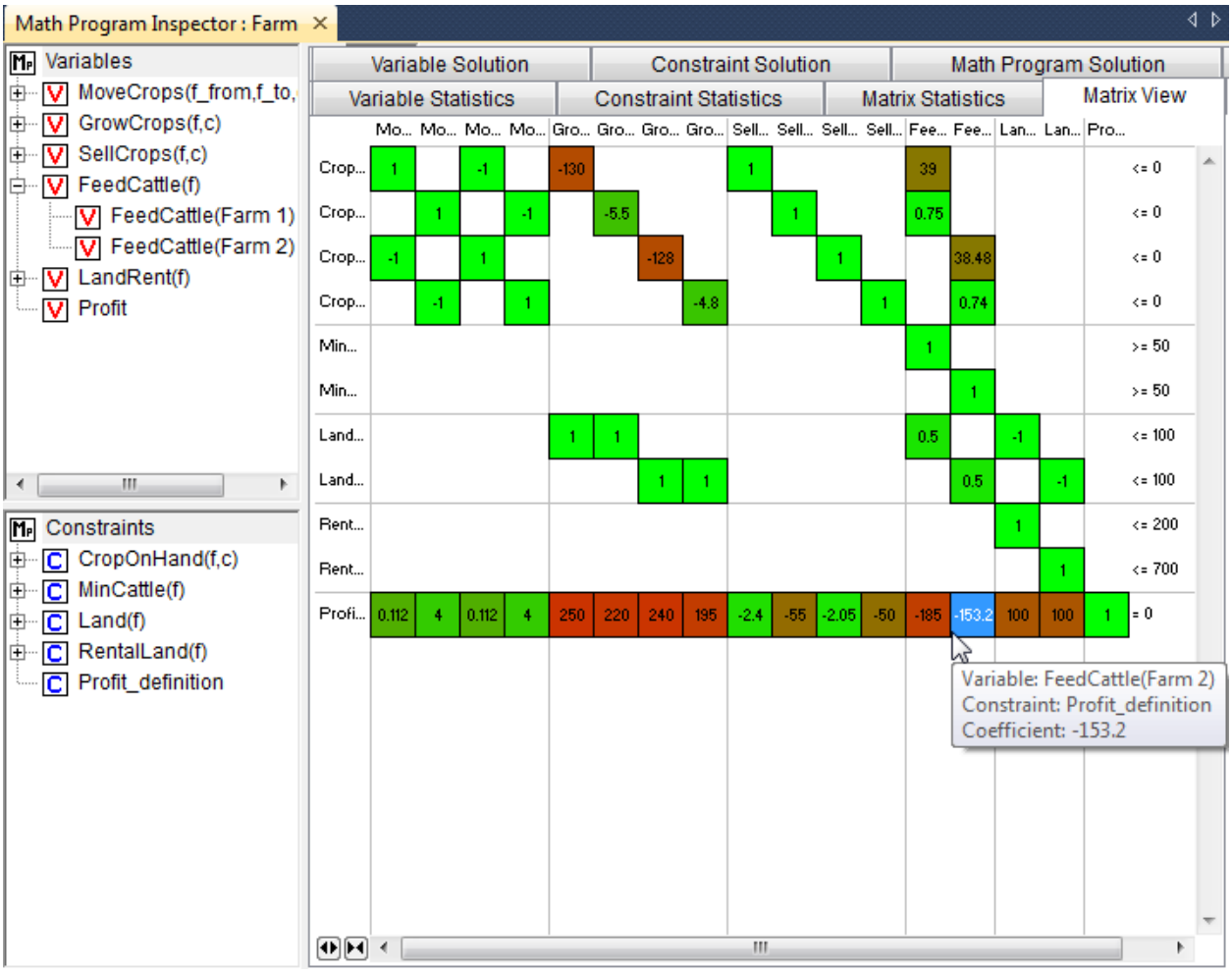


Fig. 2.41: The matrix view (individual mode)

Block coefficient editing

When viewing the matrix in individual mode, *linear* coefficient values can be changed by pressing the **F2** key, or single clicking on the block containing the coefficient to be changed.

Inspecting Solution Information

Solution tabs

The tabs discussed so far are available as long as the math program has been generated. As soon as a solution is available, the next three tabs reveal more details about this solution.

Variable Solution tab

The Variable Solution tab shows the following seven columns

- Variable Name,
- Lower Bound,
- Value (i.e. solution/level value),
- Upper Bound,
- Marginal (i.e. reduced cost),
- Basis (i.e. *Basic*, *Nonbasic* or *Superbasic*), and
- Bound (i.e. *At bound* or *In between bounds*).

By clicking in the header of a column you can sort the table according to that specific column.

Constraint Solution tab

A similar view is available for the constraints in your mathematical program. The Constraint Solution tab contains the following five columns

- Constraint Name,
- Value (i.e. solution),
- Marginal (i.e. shadow price),
- Basis (i.e. *Basic*, *Nonbasic* or *Superbasic*), and
- Bound (i.e. *Binding* or *Nonbinding*).

Solution related AIMMS options

By default AIMMS will only store marginal solution values if explicitly specified in the **Property** attribute (through the *ReducedCost* or *ShadowPrice* property). An more convenient way to ensure that all marginal solution information is available to the math program inspector is by setting the option `Store_complete_solver_solution_tree` to *yes*. When the nonlinear presolver has been activated (by setting the `Nonlinear_presolve` option (in the `Solvers General` category) to *on*), the option `Store_nonlinear_presolve_info` has to be set *yes* to make sure that the math program inspector is able to display information about the reductions that have been achieved by the nonlinear presolver.

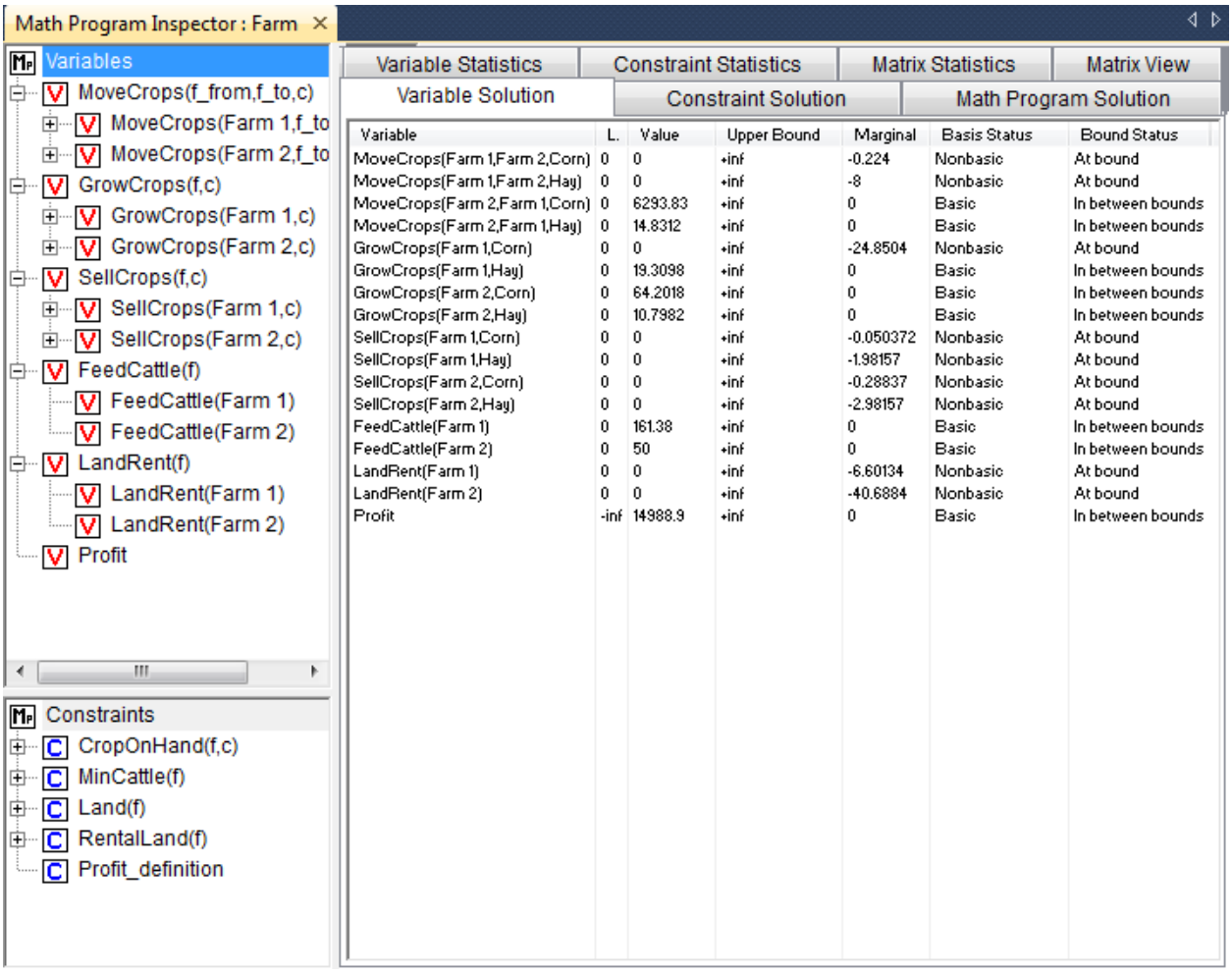


Fig. 2.42: The variable solution

Math Program Solution tab

The Math Program Solution tab retrieves solution information about the mathematical program that has been solved. This information is similar to that in the AIMMS **Progress** window.

Logging messages

The lower part of the information retrieved by this tab is used to display logging messages resulting from the **Bound Analysis** and **Unreferenced Identifiers** commands in the **Actions** menu.

Solving MIP models

Whenever your linear model is a mixed-integer model, the solver will most probably use a tree search algorithm to solve your problem. During the tree search the algorithm will encounter one or more solutions if the model is integer feasible. Once the search is completed, the optimal solution has been found.

MIP Search Tree tab

With the MIP Search Tree tab you can retrieve branching information about the search tree. Only CPLEX and GUROBI provide this information. In addition the option `Show_branch_and_bound_tree` has to be set to *on* (before the solve) to instruct AIMMS to store search tree information during the solve.

Improving the search process

The size and shape of the search tree might give you some indication that you could improve the performance of the solver by tuning one or more solver options. Consider the case in which the search tree algorithm spends a considerable amount of time in parts of the tree that do not seem interesting in retrospect. You might consider to use priorities or another branching rule, in an attempt to direct the search algorithm to a different part of the tree in an earlier stage of the algorithm.

Controlling search tree memory usage

Because all structural and statistical information is kept in memory, displaying the MIP search tree for large MIPs might not be a good idea. Therefore, you are able to control to the and size of the stored search tree through the option `Maximum_number_of_nodes_in_tree`.

Search tree display

For every node several solution statistics are available. They are the sequence number, the branch type, the branching variable, the value of the LP relaxation, and the value of the incumbent solution when the node was evaluated. To help you locate the integer solutions in the tree, integer nodes and their parent nodes are displayed in blue.

Incumbent progress

The lower part of the MIP Search Tree tab retrieves all incumbent solutions that have been found during the search algorithm. From this view you are able to conclude for example how much time the algorithm spend before finding the optimal solution, and how much time it took to proof optimality.

Performing Analysis to Find Causes of Problems

Unreferenced identifiers

One of the causes of a faulty model may be that you forgot to include one or more variables or constraints in the specification of your mathematical model. The math program inspector helps you in identifying some typical omissions. By choosing the **Unreferenced Identifiers** command (from the **Actions** menu) AIMMS helps you to identify

- constraints that are not included in the constraint set of your math program while they contain a reference to one of the variables in the variable set,
- variables that are not included in the variable set of your math program while a reference to these variables does exist in some of the constraints, and
- defined variables that are not included in the constraint set of your math program.

The results of this action are visible through the Math program solution tab.

A priori bound analysis

In some situations it is possible to determine that a math program is infeasible or that some of the constraints are redundant even before the math program is solved. The bound analysis below supports such investigation.

Implied constraint bounds

For each linear constraint with a left-hand side of the form

$$\sum_j a_{ij} x_j$$

the minimum level value $\underline{b_i}$ and maximum level value $\overline{b_i}$ can be computed by using the bounds on the variables as follows.

$$\begin{aligned} \underline{b_i} &= \sum_{j|a_{ij}>0} a_{ij}\underline{x_j} + \sum_{j|a_{ij}<0} a_{ij}\overline{x_j} \\ \overline{b_i} &= \sum_{j|a_{ij}>0} a_{ij}\overline{x_j} + \sum_{j|a_{ij}<0} a_{ij}\underline{x_j} \end{aligned}$$

Performing bound analysis

By choosing the **Bound Analysis** command (from the **Actions** menu) the above implied bounds are used not only to detect infeasibilities and redundancies, but also to tighten actual right-hand-sides of the constraints. The results of this analysis can be inspected through the Math Program Solution tab. This same command is also used to perform the variable bound analysis described below.

Implied variable bounds ...

Once one or more constraints can be tightened, it is worthwhile to check whether the variable bounds can be improved. An efficient approach to compute implied variable bounds has been proposed by Gondzio [Gon94], and is presented without derivation in the next two paragraphs.

... for $\leq$ constraints

For i in the set of constraints of the form $\sum_j a_{ij}x_j \leq b_i$, the variable bounds can be tightened as follows.

$$x_k \leq \underline{x_k} + \min_{i|a_{ik}>0} \frac{b_i - \underline{b_i}}{a_{ik}}$$
$$x_k \geq \overline{x_k} + \max_{i|a_{ik}<0} \frac{b_i - \underline{b_i}}{a_{ik}}$$

... and $\geq$ constraints

For i in the set of constraints of the form $\sum_j a_{ij}x_j \geq b_i$, the variable bounds can be tightened as follows.

$$x_k \leq \underline{x_k} + \min_{i|a_{ik}<0} \frac{b_i - \overline{b_i}}{a_{ik}}$$
$$x_k \geq \overline{x_k} + \max_{i|a_{ik}>0} \frac{b_i - \overline{b_i}}{a_{ik}}$$

Phase 1 analysis

In case infeasibility cannot be determined a priori (e.g. using the bound analysis described above), the solver will conclude infeasibility during the solution process and return a phase 1 solution. Inspecting the phase 1 solution might indicate some causes of the infeasibility.

Currently infeasible constraints

The collection of currently infeasible constraints are determined by evaluating all constraints in the model using the solution that has been returned by the solver. The currently infeasible constraints will be bookmarked in the constraint tree after choosing the **Infeasible Constraints** command from the **Actions** menu.

Substructure causing infeasibility

To find that part of the model that is responsible for the infeasibility, the use of slack variables is proposed. By default, the math program inspector will add slacks to all variable and constraint bounds with the exception of

- variables that have a definition,
- zero variable bounds, and
- bounds on binary variables.

Adapting the use of slack variables

The last two exceptions in the above list usually refer to bounds that cannot be relaxed with a meaningful interpretation. However these two exceptions can be overruled at the symbolic level through the Analysis Configuration tab of the **Properties** dialog. These properties can be specified for each node at the first level in the tree. Of course, by not allowing slack variables on all variable and constraint bounds in the model, it is still possible that the infeasibility will not be resolved.

Slack on variable bounds

Note that to add slacks to variable bounds, the original simple bounds are removed and (ranged) constraints are added to the problem definition.

$$\underline{x_j} \leq x_j + s_j^+ - s_j^- \leq \overline{x_j}$$

Elastic model

After adding slack variables as described above, this adapted version of the model is referred to as the elastic model.

Minimizing feasibility violations

When looking for the substructure that causes infeasibility, the sum of all slack variables is minimized. All variables and constraints that have positive slack in the optimal solution of this elastic model, form the substructure causing the infeasibility. This substructure will be bookmarked in the variable and constraint tree.

Irreducible Inconsistent System (IIS)

Another possibility to investigate infeasibility is to focus on a so-called *irreducible inconsistent system* (IIS). An IIS is a subset of all constraints and variable bounds that contains an infeasibility. As soon as at least one of the constraints or variable bounds in the IIS is removed, that particular infeasibility is resolved.

Finding an IIS

Several algorithms exist to find an *irreducible inconsistent system* (IIS) in an infeasible math program. The algorithm that is used by the AIMMS math program inspector, if the option `Use_IIS_from_solver` is disabled, is discussed in Chinneck ([Chi91]). Note that since this algorithm only applies to linear models, the menu action to find an IIS is not available for nonlinear models. While executing the algorithm, the math program inspector

1. solves an elastic model,
2. initializes the IIS to all variables and constraints, and then
3. applies a combination of *sensitivity* and *deletion* filters.

Deletion filtering

Deletion filtering loops over all constraints and checks for every constraint whether removing this constraint also solves the infeasibility. If so, the constraint contributes to the infeasibility and is part of the IIS. Otherwise, the constraint is not part of the IIS. The deletion filtering algorithm is quite expensive, because it requires a model to be solved for every constraint in the model.

Sensitivity filtering

The sensitivity filter provides a way to quickly eliminate several constraints and variables from the IIS by a simple scan of the solution of the elastic model. Any nonbasic constraint or variable with zero shadow price or reduced cost can be eliminated since they do not contribute to the objective, i.e. the infeasibility. However, the leftover set of variables and constraint is not guaranteed to be an IIS and deletion filtering is still required.

Combined filtering

The filter implemented in the math program inspector combines the deletion and sensitivity filter in the following way. During the application of a deletion filter, a sensitivity filter is applied in the case the model with one constraint removed is infeasible. By using the sensitivity filter, the number of iterations in the deletion filter is reduced.

Substructure causing unboundedness

When the underlying math program is not infeasible but unbounded instead, the math program inspector follows a straightforward procedure. First, all infinite variable bounds are replaced by a big constant M. Then the resulting model is solved, and all variables that are equal to this big M are bookmarked as being the *substructure causing unboundedness*. In addition, all variables that have an extremely large value (compared to the expected order of magnitude) are also bookmarked. Any constraint that contains at least two of the bookmarked variables will also be bookmarked.

Options

When trying to determine the cause of an infeasibility or unboundedness, you can tune the underlying algorithms through the following options.

- In case infeasibility is encountered in the presolve phase of the algorithm, you are advised to turn off the presolver. When the presolver is disabled, solution information for the phase 1 model is passed to the math program inspector.
- During determination of the substructure causing unboundedness or infeasibility and during determination of an IIS, the original problem is pertubated. After the substructure or IIS has been found, AIMMS will restore the original problem. By default, however, the solution that is displayed is the solution of the (last) pertubated problem. Using the option `Restore_original_solution_after_analysis` you can force a resolve after the analysis has been carried out.
- Solvers like CPLEX and GUROBI have their own algorithm to calculate an IIS. If the option `Use_IIS_from_solver` is switched on, its default setting, then AIMMS will retrieve an IIS calculated by the solver. If this option is switched off then AIMMS will use its own algorithm based on Chinneck ([Chi91]), as described above.

Scaling

A coefficient matrix is considered badly scaled if its nonzero coefficients are of different magnitudes. Scaling is an operation in which the variables and constraints in the model are multiplied by positive numbers resulting in a matrix containing nonzero coefficients of similar magnitude. Scaling is used prior to solving a model for several reasons, the most important being (1) to improve the numerical behavior of the solver and (2) to reduce the number of iterations required to solve the model.

Scale model

Solvers like CPLEX and GUROBI use their own algorithms to scale a model but in some cases it might be beneficial to use a different scaling algorithm that uses symbolic information. The scaling tool in the math program inspector can be used to find scaling factors for all symbolic variables and constraints in the model by selecting the **Scale Model** command from the **Actions** menu. The scaling factors will be displayed in the Scaling Factors tab. Once the scaling tool is finished you can select the **Resolve** command from the **Actions** menu to resolve the model which then automatically uses these scaling factors. However, to use the scaling factors in your AIMMS model you have to manually update the **Unit** attribute of the corresponding variables and constraints.

2.6.3 A worked example

This section

The example in this section is adapted from McCarl ([McC98]), and is meant to demonstrate the tools that were discussed in the previous sections. The example model is used to illustrate the detection of infeasibility and unboundedness. In addition, the example is used to find the cause of an unrealistic solution.

Model Formulation

A Farm planning model

The model considers a collection of farms. For each of these farms several decisions have to be made. These decisions are

- the amount of cattle to keep,
- the amount of land to grow a particular crop,
- the amount of additional land to rent, and
- the amount of inter-farm crop transport.

The objective of the farm model is to maximize a social welfare function, which is modeled as the total profit over all farms.

Notation

The following notation is used to describe the symbolic farm planning model.

Indices:

$f, \hat{f}$	farms
c	crops

Parameters:

C^g_{fc}	unit cost of growing crop c on farm f
C^c_f	unit cost of keeping cattle on farm f
C^m_c	unit transport cost for moving crop c
C^r_f	rental price for one unit of land on farm f
P^s_{fc}	unit profit of selling crop c grown on farm f
P^c_f	unit profit of selling cattle from farm f
L_f	amount of land available on farm f
Q	amount of land needed to keep one unit of cattle
Y_{fc}	crop yield per unit land for crop c on farm f
D_{fc}	consumption of crop c by one unit of cattle on farm f
M^c_f	minimum required amount of cattle on farm f
M^r_f	maximum amount of land to be rented on farm f

Variables:

p	total profit
c_f	amount of cattle on farm f
$m_{f\hat{f}c}$	amount of crop c moved from farm f to farm $\hat{f}$
g_{fc}	amount of land used to grow crop c on farm f
s_{fc}	amount of crop c sold by farm f
r_f	amount of extra land rented by farm f

Land requirement

The *land requirement* constraint makes sure that the total amount of land needed to keep cattle and to grow crops does not exceed the amount of available land (including rented land).

$$Qc_f + \sum_c g_{fc} \leq L_f + r_f, \qquad \forall f$$

Upper bound on rental

The total amount of rented land on a farm cannot exceed its maximum.

$$r_f \leq M^r_f, \qquad \forall f$$

Crop-on-hand

The *crop-on-hand* constraint is a crop balance. The total amount of crop exported, crop sold, and crop that has been used to feed the cattle cannot exceed the total amount of crop produced and crop imported.

$$\sum_{\hat{f}} m_{f\hat{f}c} + s_{fc} + D_{fc}c_f \leq Y_{fc}g_{fc} + \sum_{\hat{f}} m_{\hat{f}fc}, \qquad \forall (f,c)$$

Cattle requirement

The cattle requirement constraint ensures that every farm keeps at least a pre- specified amount of cattle.

$$c_f \geq M^c_f, \qquad \forall f$$

Profit definition

The total profit is defined as the net profit from selling crops, minus crop transport cost, minus rental fees, plus the net profit of selling cattle.

$$p = \sum_f \left(\sum_c \left(P^s_{fc}s_{fc} - \mathsf{C}^g g_{fc} - \sum_{\hat{f}} \mathsf{C}^m m_{f\hat{f}c} \right) - \mathsf{C}^r_f r_f + (P^c_f - \mathsf{C}^c_f)c_f \right)$$

The generated problem

Once the above farm model is solved, the math program inspector will display the variable and constraint tree plus the matrix block view as illustrated in Fig. 2.41. The solution of the particular farm model instance has already been presented in Fig. 2.42.

Investigating Infeasibility

Introducing an infeasibility

In this section the math program inspector will be used to investigate an artificial infeasibility that is introduced into the example model instance. This infeasibility is introduced by increasing the land requirement for cattle from 0.5 to 10.

Locating infeasible constraints

By selecting the **Infeasible Constraints** command from the **Actions** menu, all violated constraints as well as all variables that do not satisfy their bound conditions, are bookmarked. Note, that the solution values used to identify the infeasible constraints and variables are the values returned by the solver after infeasibility has been concluded. The exact results of this command may depend on the particular solver and the particular choice of solution method (e.g. primal simplex or dual simplex).

Substructure causing infeasibility

By selecting the **Substructure Causing Infeasibility** command from the **Actions** menu a single constraint is bookmarked. In this example, one artificial violation variable could not be reduced to zero by the solver used, which resulted in a single infeasibility. Fig. 2.43 indicates that this infeasibility can be resolved by changing the right-hand-side of the 'MinCattle' constraint for 'Farm 1'. A closer investigation shows that when the minimum requirement on cattle on 'Farm 1' is decreased from 50 to 30, the infeasibility is resolved. This makes sense, because one way to resolve the increased land requirement for cattle is to lower the requirements for cattle.

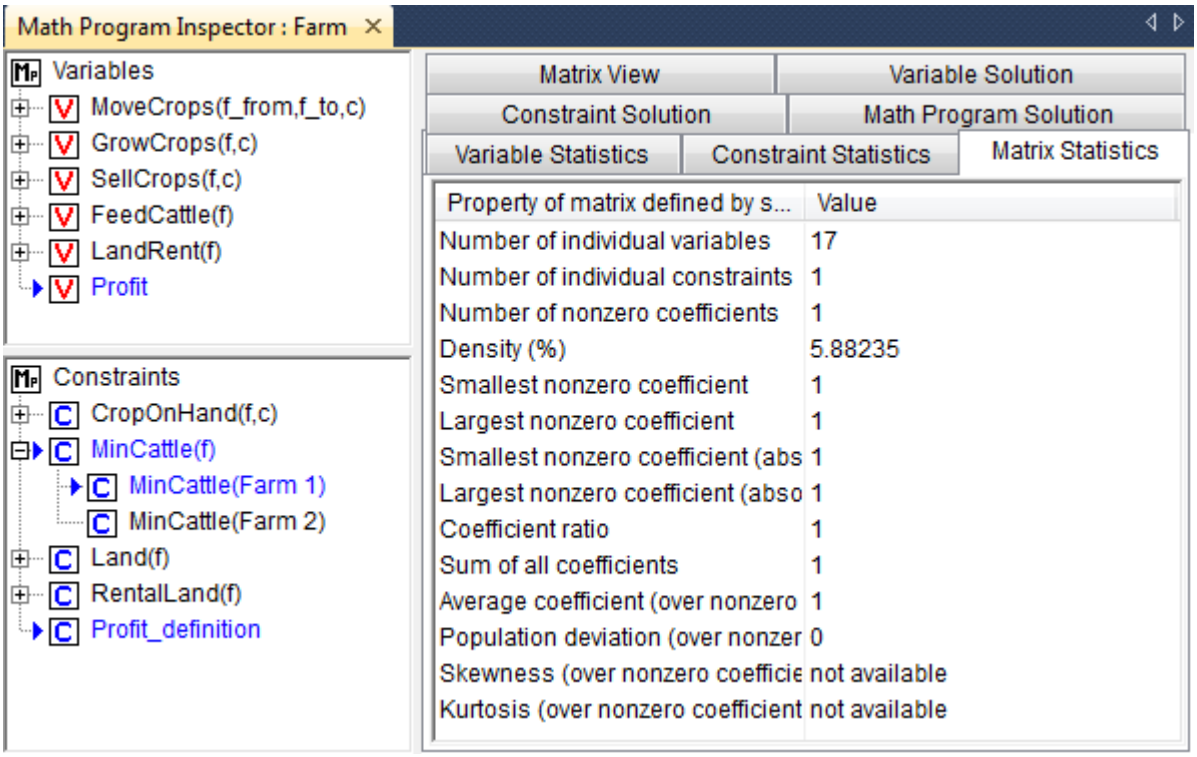


Fig. 2.43: An identified substructure causing infeasibility

Locating an IIS

By selecting the **Irreducible Inconsistent System** command from the **Actions** menu, an IIS is identified that consists of the three constraints 'RentalLand', 'Land' and 'MinCattle', all for 'Farm 1' (see Fig. 2.44).

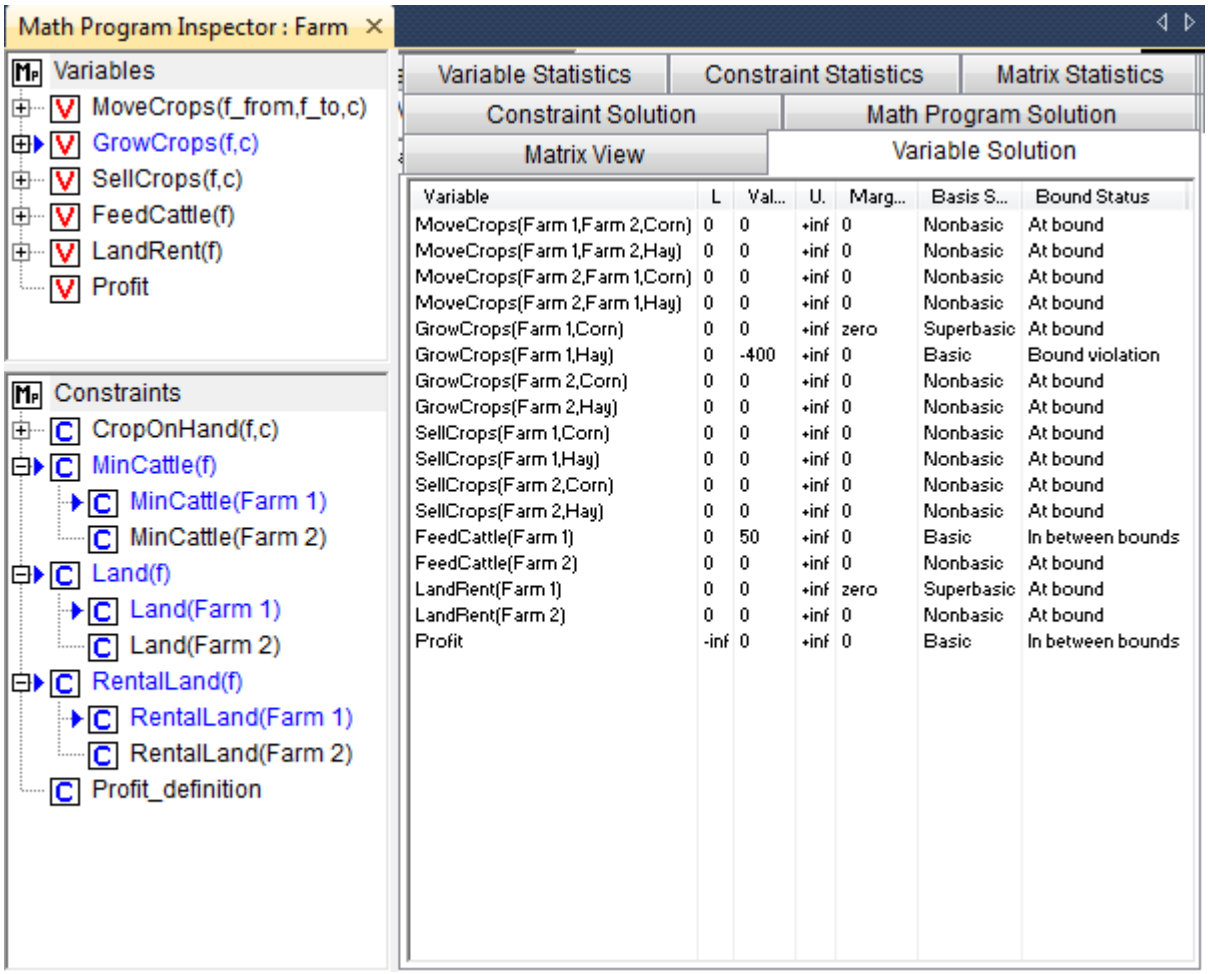


Fig. 2.44: An identified IIS

Resolving the infeasibility

The above IIS provides us with three possible model changes that together should resolve the infeasibility. These changes are

1. increase the availability of land for 'Farm 1',
2. change the land requirement for cattle on 'Farm 1', and/or
3. decrease the minimum requirement on cattle on 'Farm 1'.

It is up to the producer of the model instance, to judge which changes are appropriate.

Investigating Unboundedness

Introducing unboundedness

The example model is turned into an unbounded model by dropping the constraints on maximum rented land, and at the same time, by multiplying the price of cattle on 'Farm 1' by a factor 100 (representing a unit error). As a result, it will become infinitely profitable for 'Farm 1' to rent extra land to keep cattle.

Substructure causing unboundedness

By selecting the **Substructure Causing Unboundedness** command from the **Actions** menu four individual variables are bookmarked, and all of them are related to 'Farm 1'. Together with all constraints that contain two or more bookmarked variables these bookmarked variables form the problem structure that is subject to closer investigation. From the optimal solution of the auxiliary model it becomes clear that the 'FeedCattle' variable, the two 'GrowCrops' variables and the 'LandRent' variables tend to get very large, as illustrated in Fig. 2.45.

Resolving the unboundedness

Resolving the unboundedness requires you to determine whether any of the variables in the problem structure should be given a finite bounds. In this case, specifying an upper bound on the 'RentalLand' variable for 'Farm 1' seems a natural choice. This choice turns out to be sufficient. In addition, when inspecting the bookmarked variables and constraints on the Matrix View tab, the red color of the objective function coefficient for the 'FeedCattle' variable for 'Farm 1' indicates a badly scaled value.

Analyzing an Unrealistic Solution

Introducing an unrealistic solution

The example model is artificially turned into a model with an unrealistic solution by increasing the crop yield for corn on 'Farm 2' from 128 to 7168 (a mistake), and setting the minimum cattle requirement to zero. As a result, it will be unrealistically profitable to grow corn on 'Farm 2'.

Inspecting the unrealistic solution

Once the changes from the previous paragraph have been applied, the solution of the model is shown in Fig. 2.46. From the Variable Solution tab it can indeed be seen that the profit is unrealistically large, because a large amount of corn is grown on 'Farm 2', moved to 'Farm 1' and sold on 'Farm 1'. Other striking numbers are the large reduced cost values associated with the 'FeedCattle' variable on 'Farm 2' and the 'GrowCrops' variable for hay on 'Farm 2'.

Badly scaled matrix coefficients

When investigating an unrealistic solution, an easy first step is to look on the Matrix View tab to see whether there exist matrix coefficients with unrealistic values. For this purpose, first open the Matrix View tab in symbolic view. Blocks that are colored red indicate the existence of badly scaled values. By double clicking on such a block, you will zoom in to inspect the matrix coefficients at the individual level. In our example, the symbolic block associated with the 'GrowCrops' variable and the 'CropOnHand' constraint is the red block with the largest value. When you zoom in on this block, the data error can be quickly identified (see Fig. 2.47). You can also use the **Scale Model** command from the **Actions** menu to let AIMMS calculate scaling factors that can be used to reduce the amount of badly scaled values in the coefficient matrix.

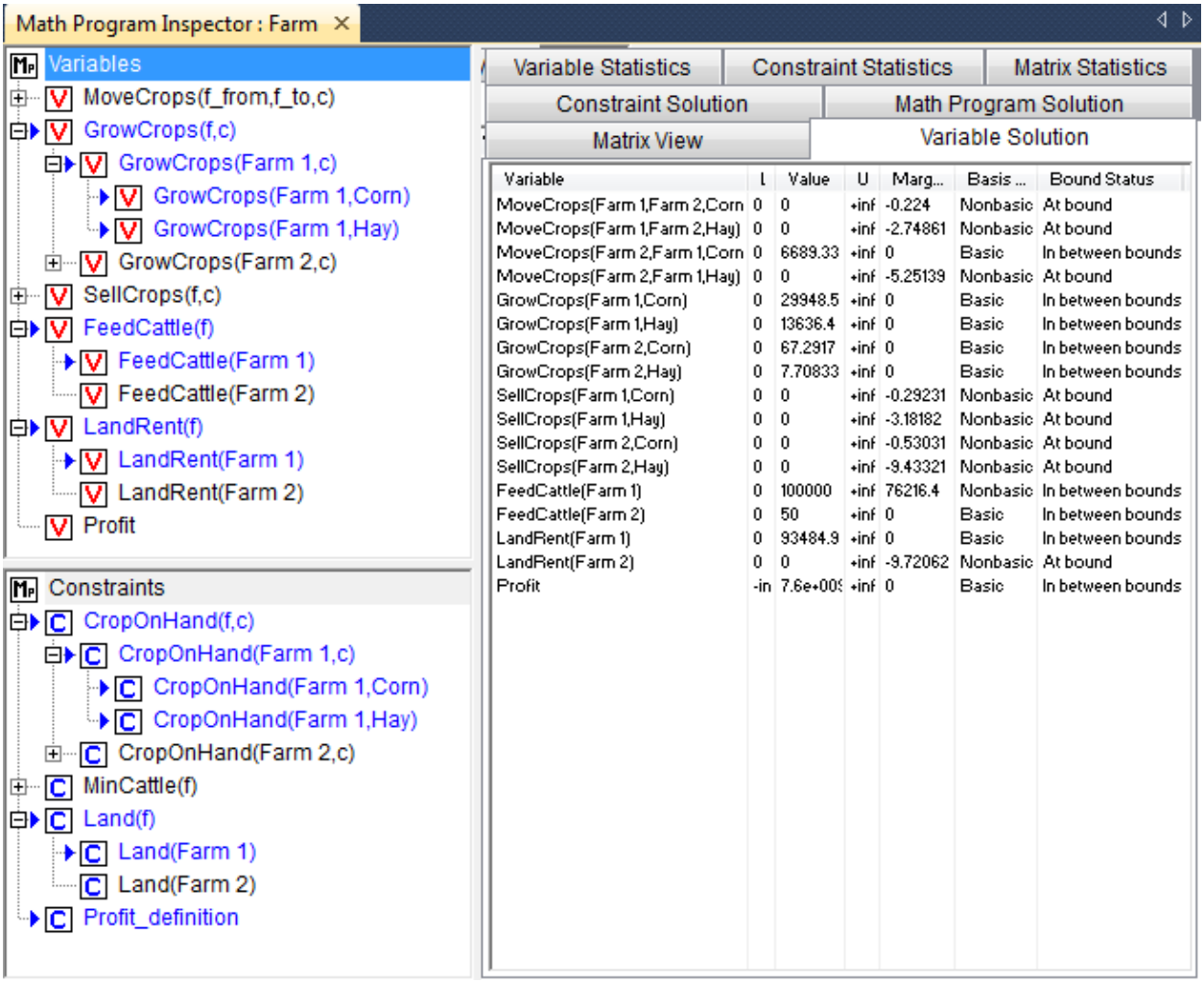


Fig. 2.45: An identified substructure causing unboundedness

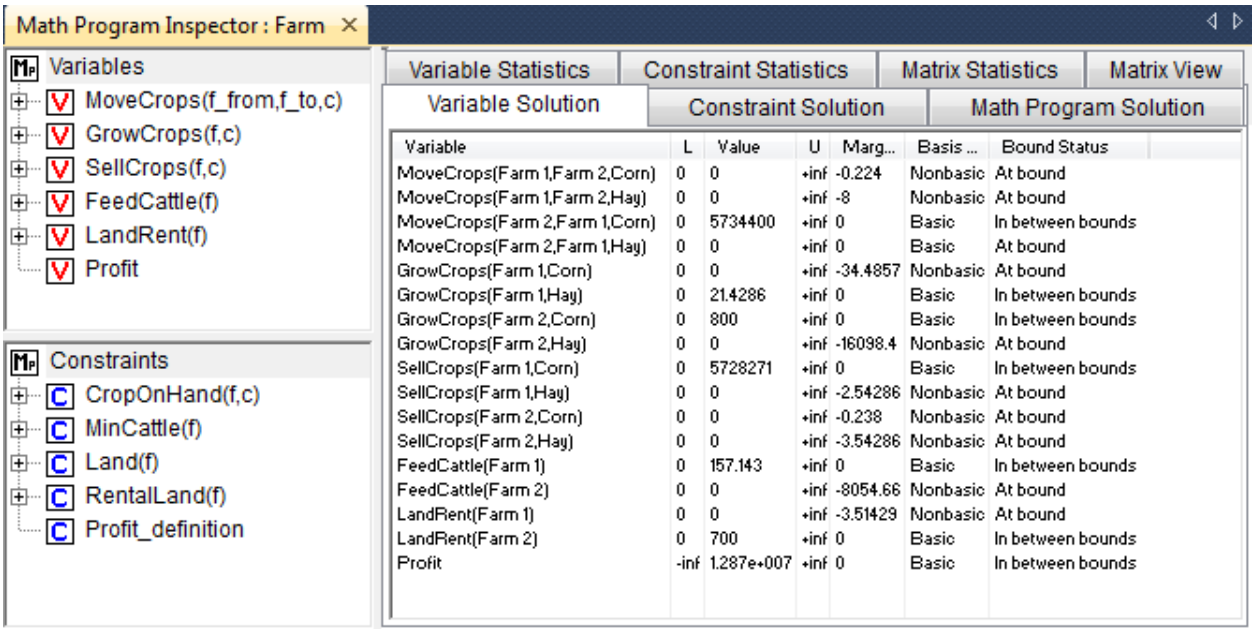


Fig. 2.46: An unrealistic solution

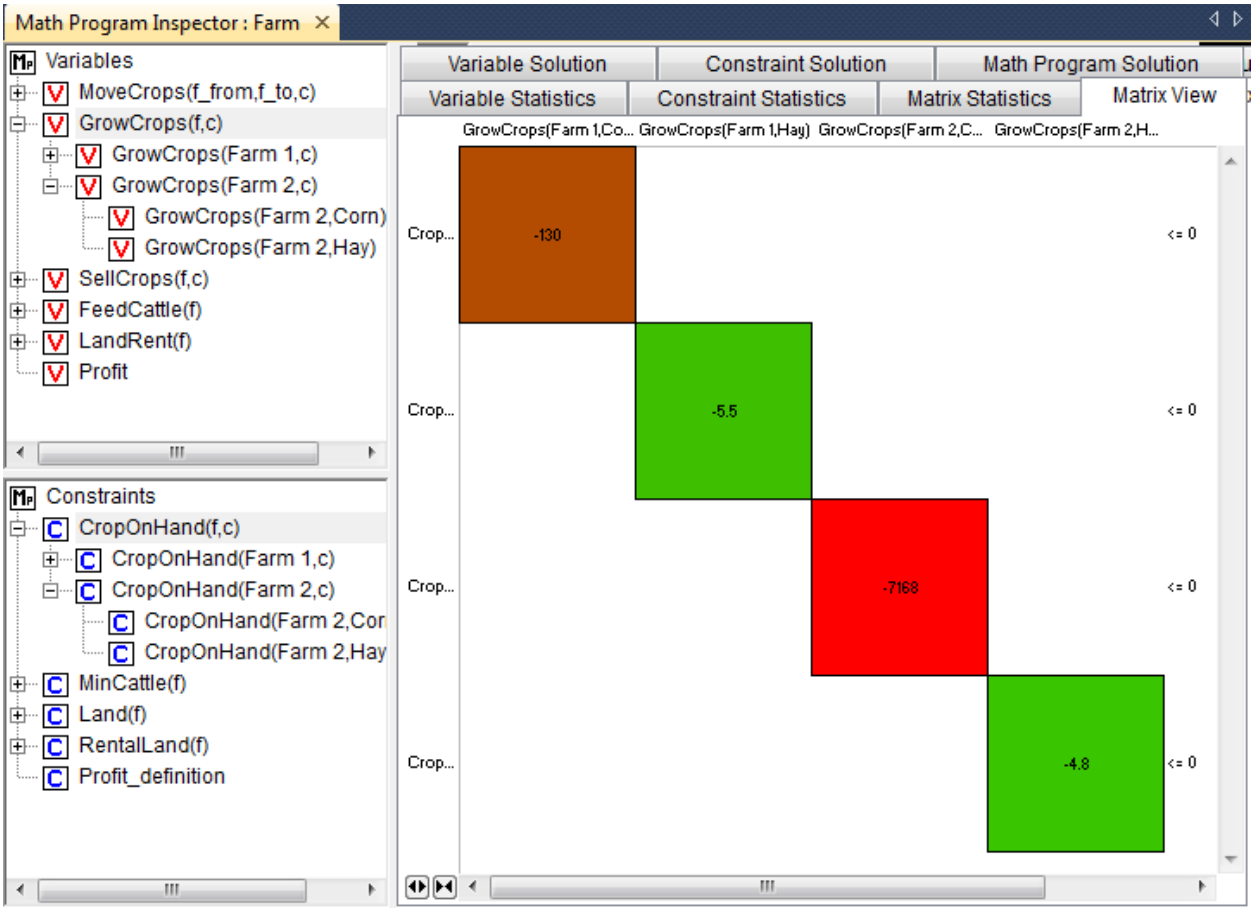


Fig. 2.47: The Matrix View tab for an unrealistic solution

Primal and dual contributions . . .

A second possible approach to look into the cause of an unrealistic solution is to focus on the individual terms of both the primal and dual constraints. In a primal constraint each term is the multiplication of a matrix coefficient by the value of the corresponding variable. In the Math Program Inspector such a term is referred to as the *primal contribution*. Similarly, in a dual constraint each term is the multiplication of a matrix coefficient by the value of the corresponding shadow price (i.e. the dual variable). In the Math Program Inspector such a term is referred to as the *dual contribution*.

. . . can be unrealistic

Whenever primal and/or dual contributions are large, they may indicate that either the corresponding coefficient or the corresponding variable value is unrealistic. You can discover such values by following an iterative process that switches between the Variable Solution tab and the Constraint Solution tab by using either the **Variable Statistics** or the **Constraint Statistics** command from the right-mouse popup menu.

Procedure to resolve unrealistic solutions

The following iterative procedure can be followed to resolve an unrealistic solution.

- Sort the variable values retrieved through the Variable Solution tab.
- Select any unrealistic value or reduced cost, and use the right-mouse popup menu to switch to the Variable Statistics tab.
- Find a constraint with an unrealistic dual contribution.
- If no unrealistic dual contribution is present, select one of the constraints that is likely to reveal some information about the construction of the current variable (i.e. most probably a binding constraint).
- Use the right-mouse popup menu to open the Constraint Statistics tab for the selected constraint.
- Again, focus on unrealistic primal contributions and if these are not present, continue the investigation with one of the variables that plays an important role in determining the level value of the constraint.
- Repeat this iterative process until an unrealistic matrix coefficient has been found.

You may then correct the error and re-solve the model.

Inspecting primal contributions

In the example, the 'Profit' definition constraint indicates that the profit is extremely high, mainly due to the amount of corn that is sold on 'Farm 1'. Only two constraints are using this variable, of which one is the 'Profit' definition itself. When inspecting the other constraint, the 'CropOnHand' balance, it shows that the corn that is sold on 'Farm 1' is transported from 'Farm 2' to 'Farm 1'. This provides us with a reason to look into the 'CropOnHand' balance for corn on 'Farm 2'. When inspecting the primal contributions for this constraint the data error becomes immediately clear (see Fig. 2.48).

Inspecting dual contributions

The same mistake can be found by starting from an unrealistic reduced cost. Based on the large reduced cost for the 'FeedCattle' variable on 'Farm 2', the dual contributions indicate that the unrealistic value is mainly caused by an unrealistic value of the shadow price associated with the 'Land' constraint on 'Farm 2'. While investigating this constraint you will notice that the shadow price is rather high, because the 'GrowCrops' variable for corn on 'Farm 2' is larger than expected. The dual contribution table for this variable shows a very large coefficient for the 'CropOnHand' constraint for corn on 'Farm 2', indicating the data error (see Fig. 2.49).

Combining primal and dual investigation

The above two paragraphs illustrate the use of just primal contributions or just dual contributions. In practice you may very well want to switch focus during the investigation of the cause of an unrealistic solution. In general, the Math Program Inspector has been designed to give you the utmost flexibility throughout the analysis of both the input and output of a mathematical program.

2.6.4 Bibliography

2.7 Distributing an AIMMS app

A complete AIMMS application consists of several files, all of which should also be distributed to the user of the application. To make the distribution of an AIMMS application easier, AIMMS offers the possibility to distribute your project as an end-user application, by packing all relevant files into a single file with the `.aimmspack` extension, and publishing your app to an AIMMS PRO Platform. We also provide a Cloud Platform where this AIMMS PRO platform is already installed, maintained and scalable.

Publishing

To prepare and publish your app, please refer to Deploy an Application on AIMMS PRO

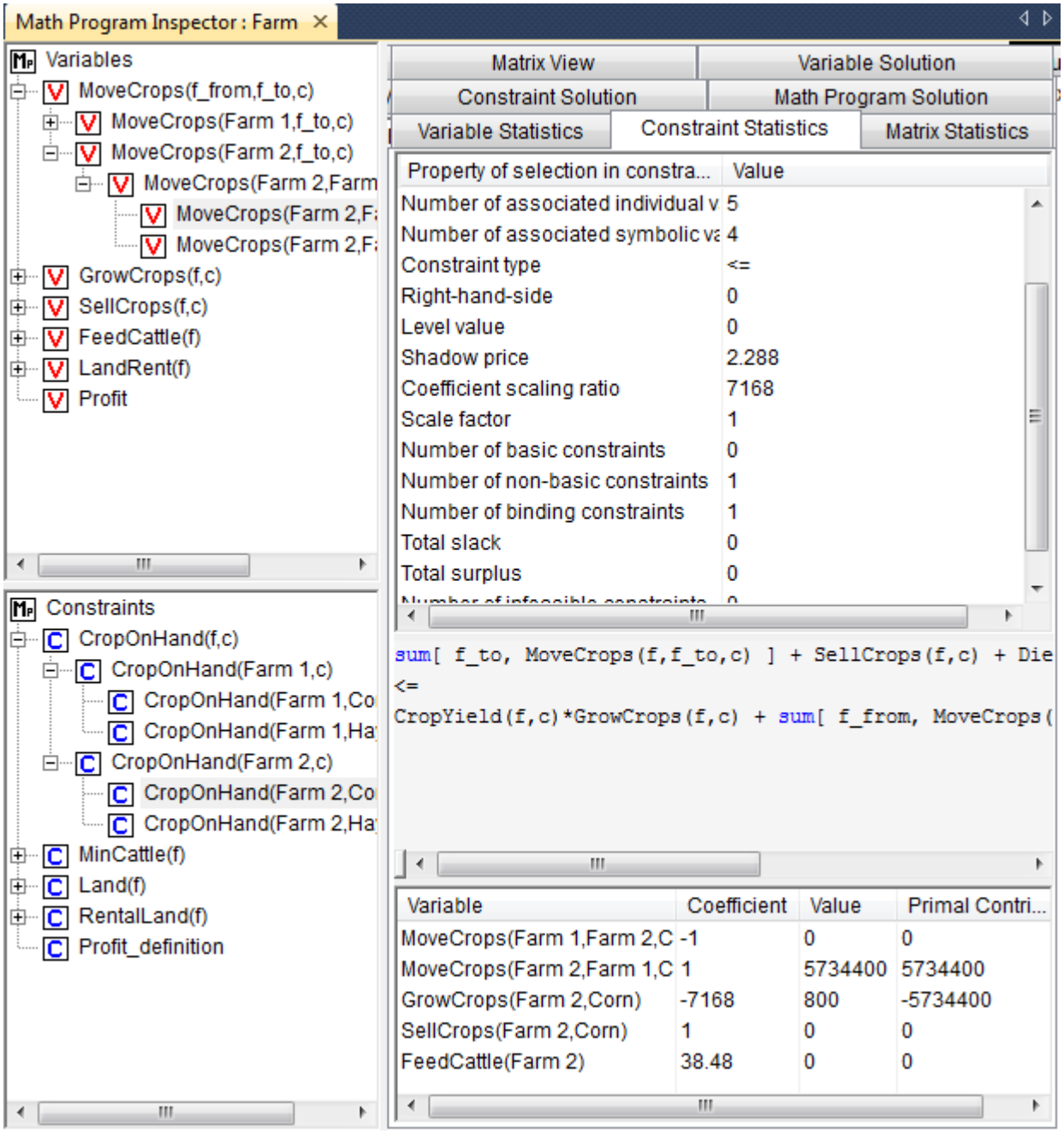


Fig. 2.48: Inspecting primal contributions

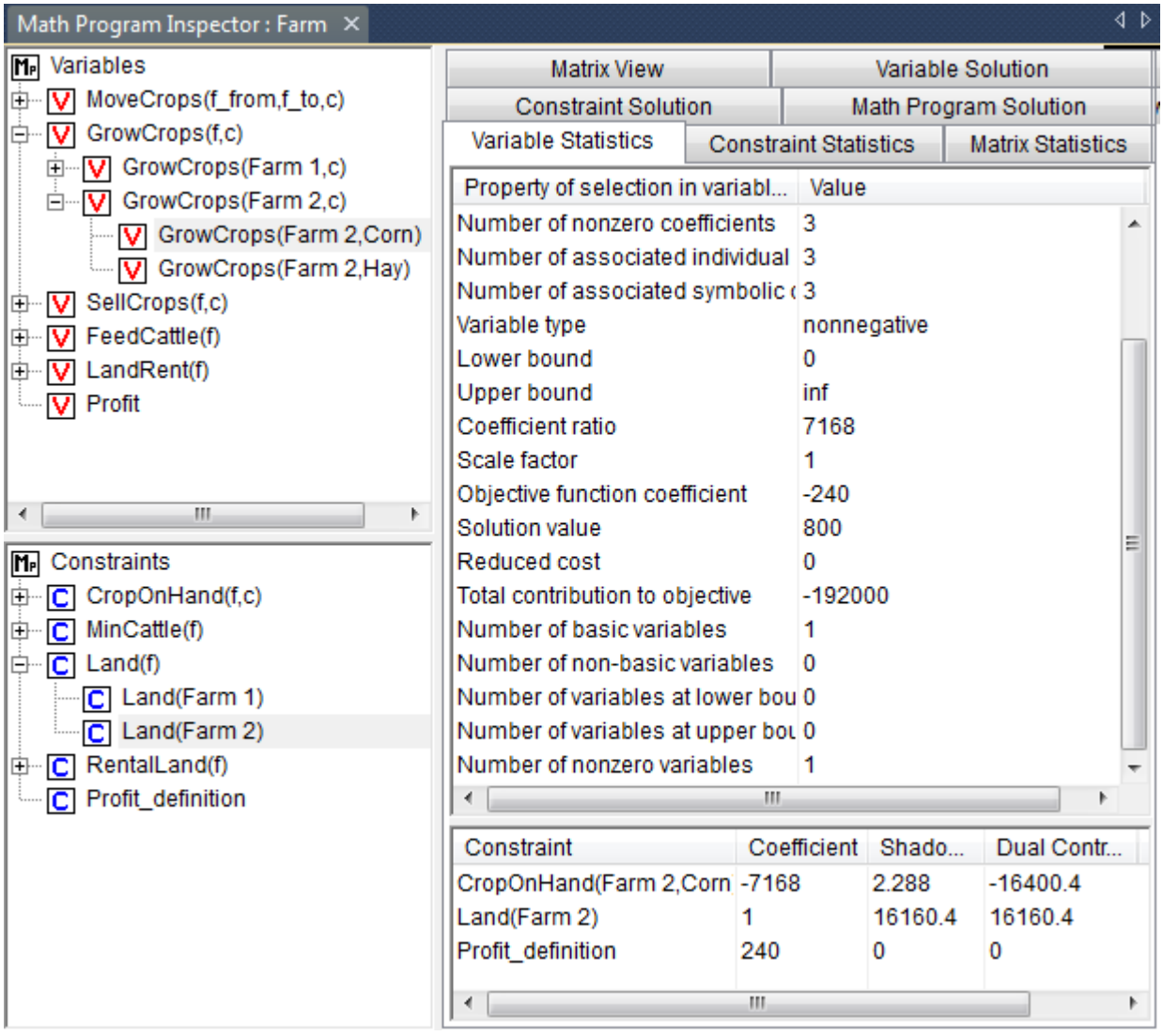


Fig. 2.49: Inspecting dual contributions

CHAPTER

THREE

DATA MANAGEMENT

3.1 Case Management

Working with data is a central part of any modeling application. Data can come from external sources or from AIMMS' proprietary case files. This chapter introduces AIMMS' capabilities with respect to creating and managing a collection of case files. Furthermore, AIMMS' capabilities of working with data from multiple case files, both from within the language and from within graphical data objects on end-user pages, are illustrated.

Not in this chapter

AIMMS uses a proprietary binary format for case files to store data compactly, quickly, and easily. This propietary format makes case files unsuitable to exchange data with other programs. AIMMS' capabilities to exchange data with other programs is documented in the Language Reference:

- Communicating With Databases
- Reading and Writing Spreadsheet Data
- Reading and Writing XML Data
- The AIMMS Programming Interface

Furthermore, the Data Exchange Library is the prefered way nowadays to exchange data between AIMMS and other softwares or applications, notably through a REST API.

3.1.1 Working with Cases

Case management tasks

A case file is a single file containing the data of some identifiers in an AIMMS model. The **Data** menu is the main tool through which you can accomplish tasks such as saving, loading, merging, and comparing case files. This menu item is part of the developer menu and is available by default on all end-user pages.

The active case

In AIMMS, all the data that you are currently working with, is referred to as the *active* case. If you have not yet loaded or saved a case file, the active case is *unnamed*, otherwise the active case is *named* after the name of the last loaded or saved case file on disk. If the active case is named, its name is displayed in the status bar at the bottom of the AIMMS window.

Saving a case file

When you save a named active case, AIMMS will save it to the associated case file on disk by default, thereby overwriting its previous contents. If the active case is unnamed, or when you try to save a case using the **Data-Save Case As** menu, AIMMS will open the **Save Case** dialog box illustrated in *The Save Case File dialog box* (page 108).

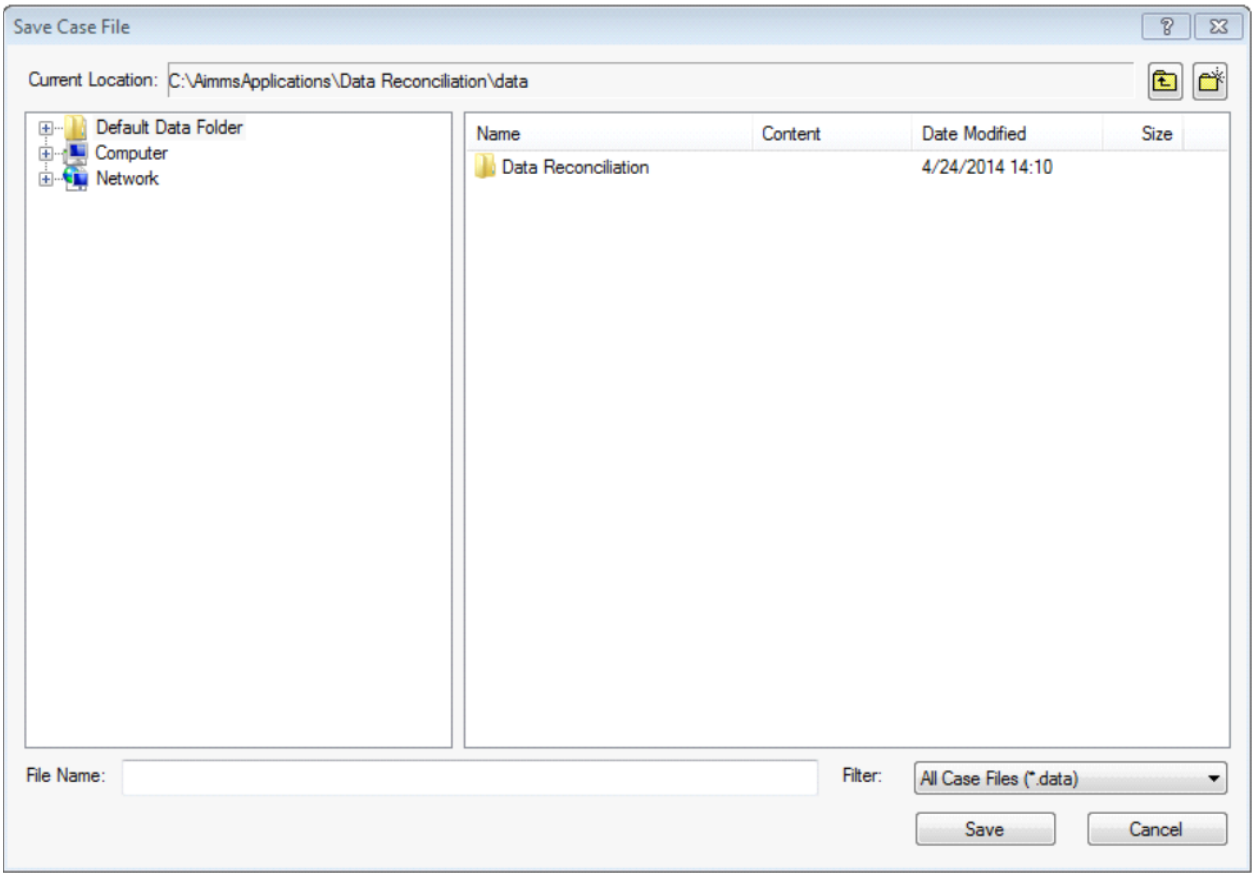


Fig. 3.1: The **Save Case File** dialog box

In the **Save Case File** dialog box you can enter the name of the case file, and, optionally, select the folder in which the case file is to be stored. After successfully saving a case file through the **Save Case File** dialog box, the active case will become named.

Loading a case file

AIMMS supports three modes for loading the data of a case file, as summarized in the following table:

mode	changes name of active case	replaces data	merges data
load as active	✓	✓	
load into active		✓	
merge into active			✓

The modes are explained in more detail below.

Load as active

The most frequently used mode for loading a case file is loading the case file *as active*, through the **Data-Load Case-As Active** menu. Loading a case file as active completely replaces the active data of all identifiers in the case file being loaded. Data of identifiers that are not stored in the case file, remain unchanged. In addition, the active case will be named after the loaded case file. Before loading a case file as active, AIMMS will ask you whether the current active case data needs to be saved whenever this is necessary.

Load into active

Loading a case file *into active*, through the **Data-Load Case-Into Active** menu, is completely identical to loading a case as active, with the exception that the name of the active case will not be changed. Thus, by loading data into the active case you can replace part, or all, of the contents of the active case with data obtained from another case file.

Merge into active

Merging a case file *into active*, through the **Data-Load Case-Merge Into Active** menu, does not change the name of the active case either. Merging a case file into the active case partially replaces the data in the active case with only the nondefault values stored in the loaded case file. Data in the active case, for which no associated nondefault values exist in the merged case file, remain unchanged.

Starting a new case

Using the **Data-New Case** menu item, you can instruct AIMMS to start a new, unnamed, active case. However, the data in the active case will remain *unchanged*. Before starting a new case, AIMMS will ask you whether the current active case data needs to be saved.

3.1.2 Managing multiple case selections

Viewing multiple case files

AIMMS allows you to simultaneously view the results of several case files within the graphical user interface. In addition, it is possible to reference data from multiple case files from within the modeling language, enabling you to perform advanced forms of case comparison.

Multiple case selections

AIMMS offers a tool to construct a selection of cases to which you want simultaneous access, either from within the graphical user interface or from within the model itself. You can add one or more selected cases from within the **Data** menu to the multiple case file selection through the **Data-Multiple Cases** menu. This will open the **Select Multiple Case Files** dialog box illustrated in *The Select Multiple Case Files dialog box* (page 110).

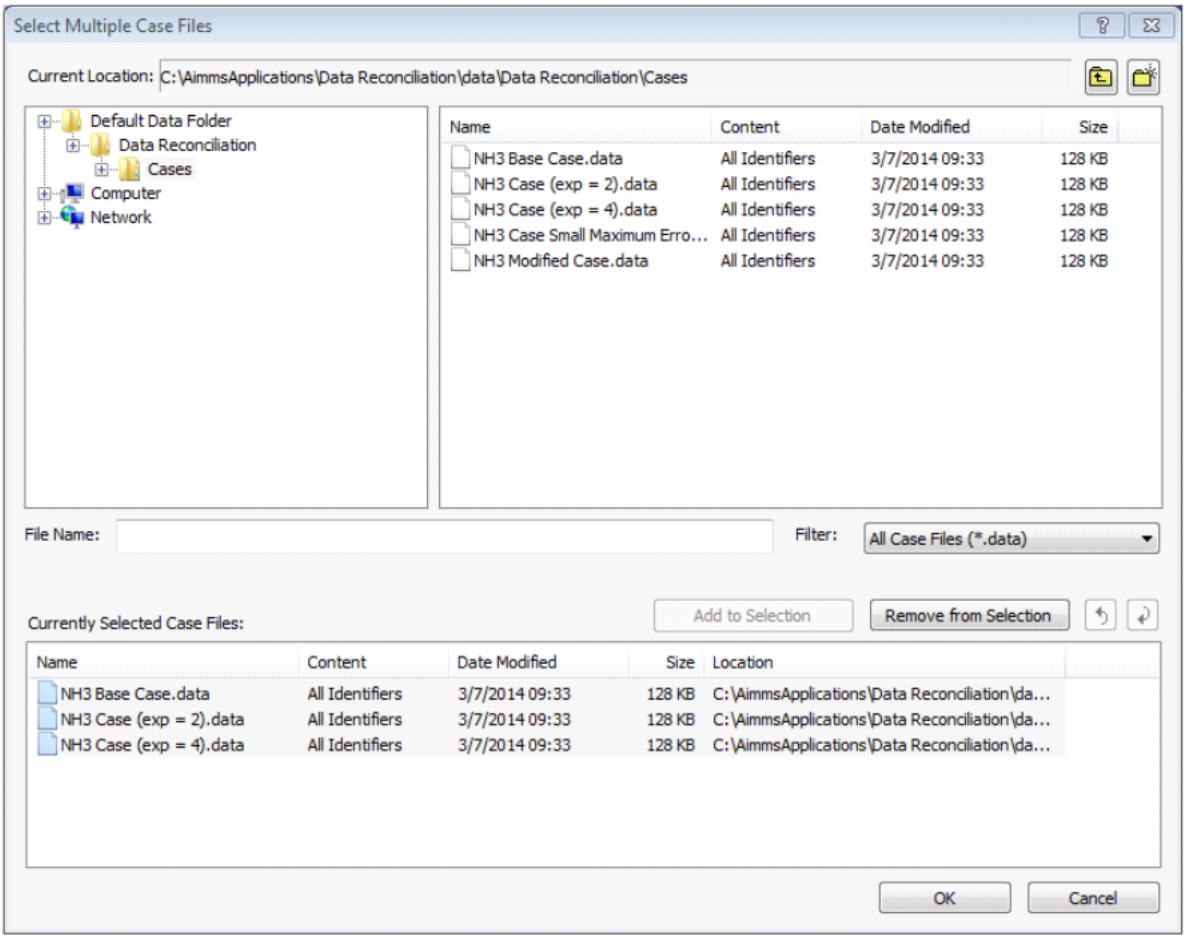


Fig. 3.2: The **Select Multiple Case Files** dialog box

It shows the current contents of the multiple case file selection. You can modify the order of the displayed cases, and add cases to or delete cases from the collection.

Viewing Multiple Case Data

Viewing multiple case data

The prime use of multiple case selection takes advantage of AIMMS' capability of displaying data from multiple cases within its graphical objects. *Example of a multiple case object* (page 111) is an illustration of a table which displays the contents of a single identifier for all the cases in the case selection shown in *The Select Multiple Case Files dialog box* (page 110).

Case comparison of flows and compositions:

	NH3 Base Case			NH3 Case (exp			NH3 Case (exp		
	Meas	Flow	Error	Meas	Flow	Error	Meas	Flow	Error
Inflow	111.98	117.03	4.51	111.98	113.58	1.42	111.98	113.55	1.40
Mix		475.03			470.88			468.88	
NH3-Mix		475.03			470.88			468.88	
NH3-Flow	105.59	105.59		105.59	104.26	1.26	105.59	104.13	1.38
Residu		369.44			366.62			364.75	
Ar-Flow		11.44			9.32			9.42	
Feedback	358.00	358.00		358.00	357.30	0.20	358.00	355.32	0.75

Fig. 3.3: Example of a multiple case object

Creating multiple case objects

A data object on a page in the graphical end-user interface can be turned into a multiple case object by checking the multiple case property in the object-specific options in the object **Properties** dialog box. *Table-specific Properties dialog box* (page 111). illustrates the object-specific **Properties** dialog box of a table object.

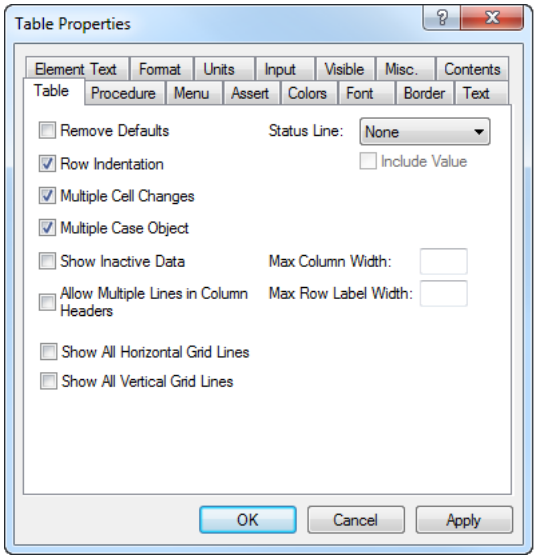


Fig. 3.4: Table-specific **Properties** dialog box

As a result of enabling multiple case display, the object will be extended with one additional virtual dimension, the case index, which will be displayed in a standard way.

Restrictions

AIMMS only supports the display of multiple case data in object types for which the added dimension can be made visible in a well-defined manner. The most important object types that support multiple case displays are tables, pivot tables, curves, bar charts and scalar objects. Because of the extra dimension, the bar chart object is only able to display multiple case data for scalar and 1-dimensional identifiers. During a single case display, a bar chart can also be used to view 2-dimensional identifiers.

Case Referencing from Within the Language

Using inactive case data

In addition to viewing data from multiple case files as graphical objects in the graphical user interface, AIMMS also allows you to reference the data of case files that are not currently active within the model. This allows you, for instance, to perform advanced forms of case file differencing by comparing the current values of particular identifiers in your model with the corresponding values stored in an inactive case.

The set `AllCases`

The collection of all case files referenced via the AIMMS data menu, or via the intrinsic functions such as `CaseFileLoad`, and `CaseFileMerge` is available in the AIMMS language through the predefined integer subset `AllCases`. Each case file is represented by an integer element in this set, and, as explained in *Case Management Functions* (page 124), AIMMS offers several built-in functions to obtain additional information about a case through its case number.

The set `CurrentCaseSelection`

AIMMS stores the case selection constructed in the **Select Multiple Case Files** dialog box presented above in the predefined set `CurrentCaseSelection`, which is a subset of the ever growing set `AllCases`. Through this set you get easy access within your model to the cases selected by your end-users in the **Select Multiple Case Files** window.

Referencing case data

You can reference the values of specific identifiers within a particular case by simply prefixing the identifier name with an index or element parameter in the set Allcase or any of its subsets. Thus, if `cs` is an index in the set`CurrentCaseSelection`, the following simple assignment will inspect every case in the user-selected multiple case selection, and store the values of the variable `Transport(i,j)` stored in that case in the parameter `CaseTransport`, which has one additional dimension over the set of `CurrentCaseSelection`.

```
CaseTransport(cs,i,j) := cs.Transport(i,j);
```

Advanced case comparison

The capability of referencing inactive case data, enables you to perform advanced forms of case comparison, which would be hard to accomplish without the AIMMS facilities for case referencing. As an example, consider the following statement.

```
RelativeDiff(cs,i,j) := (cs.Transport(i,j) - Transport(i,j)) /$ Transport(i,j);
```

It computes the relative difference between the current values of the variable `Transport(i,j)` and those values stored for each case referenced.

Inactive data

Please note that `cs.Transport(i,j)` above, may contain inactive data, when index `cs` refers to the active case. In order to remedy this, you may want to use the `CleanUp` statement, see Data Control, at the start of procedures containing case referencing.

3.1.3 Working with selections of identifiers

Case content type

Next to saving the contents *all* identifiers in a case file, it is also possible to save the data of *a selection of* identifiers in a case file. Such a selection of identifiers to be saved to a case file is called a *case content type*. A case content type is a subset of `AllIdentifiers`.

Collection

The set `AllCaseFileContentTypes` contains all case content types. It is a subset of the predeclared set `AllSubsetsOfAllIdentifiers`. The set `AllCaseFileContentTypes` is initialized to contain only the case content type `AllIdentifiers`. By adding additional subsets of `AllIdentifiers`, you are allowing your end user to decide which selection of identifiers is to be saved. The predeclared element parameter `CurrentCaseFileContentType` is used to indicate the case content type selected by the end-user of your application.

Example

You may add the predeclared set `CurrentInputs` to the set `AllCaseContentTypes`, which allows an end-user to decide whether to save the data of all identifiers in your model, or just of the collection of current input parameters. This is illustrated in *The Save Case File dialog box offering content types* (page 114) where the **Save Case File** dialog box allows you to select between the case content types `AllIdentifiers` and `CurrentInputs`.

Use during case load

When loading a case, all identifiers stored in the case file will be loaded; the current contents of the case content type by which the file is saved will be ignored.

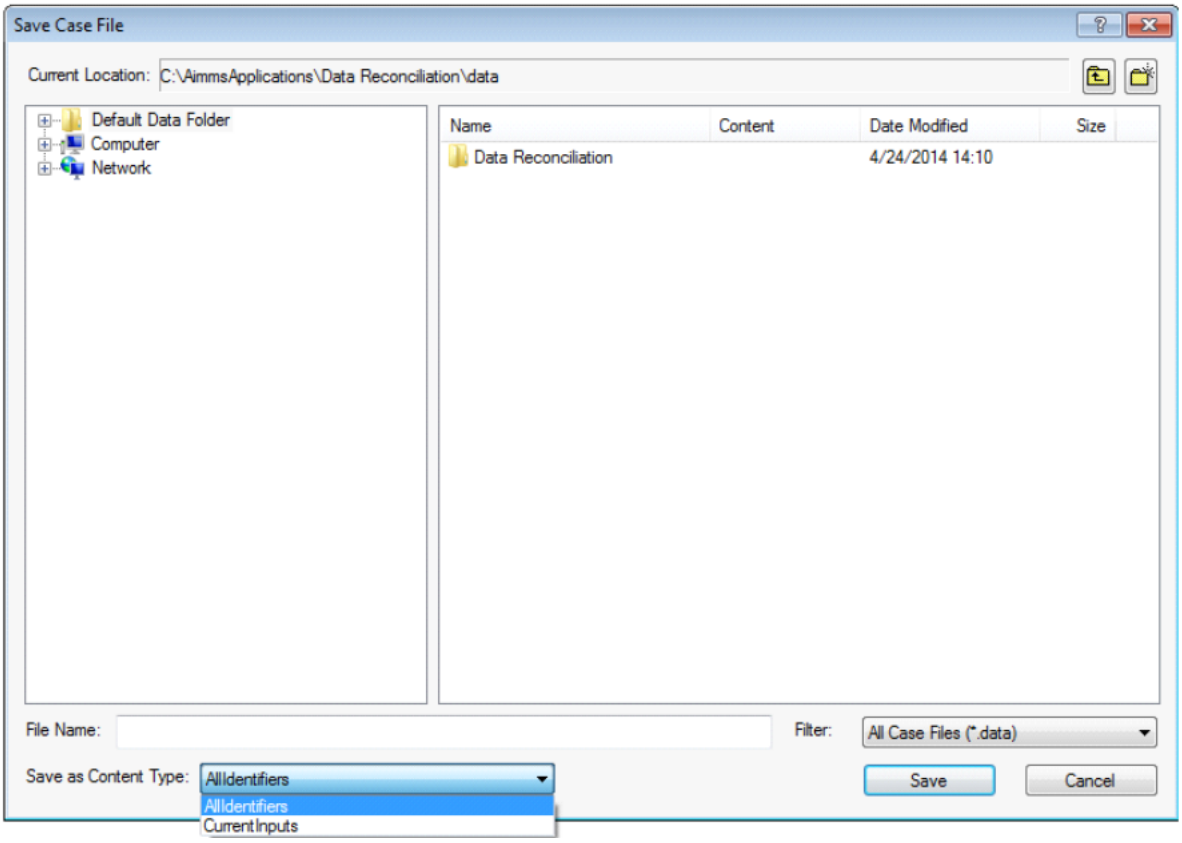


Fig. 3.5: The **Save Case File** dialog box offering content types

Data not stored

Identifiers in an AIMMS model can have the `NoSave` property. Identifiers with this property will not be saved in any case file regardless of the current case content type. This property can be set via the attribute forms of the identifiers that can contain data.

CHAPTER

FOUR

MISCELLANEOUS

4.1 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

- 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 User Interface Related Functions.

4.1.1 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 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 AllUpdatableIdentifiers

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.

4.1.2 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' ;
```

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. 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.

4.1.3 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.

Page Functions

Warning: The AIMMS WinUI is deprecated, and thus Page functions as well. Please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and the WebUI System Library.

Model page control

The possibility of opening pages from within a model provides flexibility compared to page tree-based navigation. 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*.

Print Functions

Warning: The AIMMS WinUI is deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead.

Printing facilities

AIMMS provides a printing capability in the form of *print pages*.

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*.

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.

Dialog Box Functions

Warning: The AIMMS WinUI is deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and use Dialog Pages, Status Bar, Download Widget and Upload Widget.

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, and open these using the PageOpen function discussed in *Page Functions* (page 121).

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*])

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 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*
  .8cm,*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.

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*])

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
- MemoryStatistics
- IdentifierMemoryStatistics

Profiler control

The following profiler control functions are available in AIMMS.

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

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*)

4.2 Calling Aimms

This chapter discusses the command line options of the AIMMS program, and explains the details for running AIMMS end-user applications. In addition, the chapter explains how you can link AIMMS to your own program as a DLL, and presents a short overview of the functionality available through the AIMMS- specific Application Programming Interface (API) provided by this DLL.

4.2.1 AIMMS command line options

Calling AIMMS

On the AIMMS command line, you can specify a number of options and arguments that will influence the manner in which AIMMS is started. The following line illustrates the general structure of a call to the AIMMS program.

aimms.exe [command-line-options] [project-file [session-arguments]]

Command line options

Table 4.1 provides an overview of the command line options that you can specify. AIMMS offers both long and short option names, and some options require a single argument. All short option names start with a single minus (-) sign, followed by a single character. By convention, short options that require an argument use capital characters. The long option names are always preceded by a double minus sign (--), followed by a descriptive text. In general, the long option names are easier to remember, while the short names permit a more compact command line. Short option names without an argument may be appended one after another with only a single minus sign at the beginning.

Table 4.1: AIMMS command line options

Long name	Short name	Argument
--user	*-U*	user[:password]
--backup-dir	*-B*	backup directory
--log-dir	*-L*	log directory
--config-dir	*-C*	configuration directory
--license		license name
--license-wait-seconds		seconds to wait
--run-only	*-R*	procedure name
--user-database		user database file maximum number of threads
--minimized	*-m*	/
--maximized	*-x*	/
--hidden		/
--as-server		/
--end-user	*-e*	/
--no-solve		/
--help	*-h*	/
--unpack-folder		unpack folder
--export-to		export aimmspack/folder

Specifying a user

When an AIMMS project is linked to an end-user database (see *Project Security* (page 135)), you must log on to the project before being able to run it. Through the *--user* command line option, you can specify a user name and optionally a password with which you want to log on to the system. When you specify just a user name, a log on screen will appear with the provided user name already filled in. If you specify a password as well, AIMMS will verify its correctness and skip the log on screen altogether if the user name- password combination is acceptable. Providing both the user name and the password is not recommended for interactive use, but may be convenient when you want the model to run unattended.

Backup and log directories

With the *--backup-dir* and *--log-dir* options you can override the default directories where AIMMS will store temporary information such as case and model backups, the AIMMS and solver listings, and the message log. You can modify the defaults for these directories using the project options dialog box (see *AIMMS execution options* (page 140)).

AIMMS configuration

By default, AIMMS stores a number of global configuration files, such as the AIMMS license file and the solver configuration file, in the common application area of your computer. If you want to store configuration files in a different location, you can indicate this through the *--config-dir* option. You can use this option, for instance, to indicate where the configuration files for your particular machine can be found when the AIMMS system that you use is stored on a network disk, and when you do not use a license server.

License name

Through the *--license* option you can select any AIMMS license that you installed in the AIMMS **License Configuration** dialog box. The value that you specify for the *--license* option should match an entry in the **License** column in the left pane of the **License Configuration** dialog box. In case you are using a network license with different profiles, you should make a different entry in the AIMMS **License Configuration** for each profile you want to use and you can use the *--license* option to open AIMMS with a license with a specific profile.

Network logon timeout

When you are using a network license, the license server may not have a license available for you right away. Through the *--license-wait-seconds* option you can specify the number of seconds you want AIMMS to wait for a network license to become available. If you do not specify this option AIMMS will use a default timeout of 0 seconds. When reaching the given timeout, AIMMS will try the next license in your license configuration, or will return with a license error if no other licenses are available.

User database location

When your application has been set up for use by multiple users, all user and group information associated with the application is stored in a separate (encrypted) user database (see *User authentication and authorization* (page 139) for more details on this topic). Through the *--user-database* option you can move the location of this user database file (to for example a single location that is shared among all users on the network) even though you might not have developer rights to the application.

Limiting the number of parallel threads

Through the option *--max-threads* you can specify the maximum number of (virtual) cores that AIMMS is allowed to use during the execution of statements, the evaluation of definitions or during a parallel solve. By default AIMMS uses the number of cores that are available on the machine, but during a heavy load of multiple processes it might be beneficial to limit the number of cores that AIMMS will use. This option is ignored if you set it to a value that is larger than the actual number of cores.

Running minimized, maximized, hidden, or as server

Through the *--minimized*, *--hidden* and *--maximized* options you can indicate whether you want AIMMS to start in a minimized or hidden state (i.e. just as a button on the task bar, or not visible at all), or to fill up the entire screen. Running AIMMS minimized or hidden may be convenient when AIMMS is called non-interactively from within another program through the AIMMS API (see The AIMMS Programming Interface of the Language Reference). In this way, your program can use AIMMS to solve an optimization model after which it resumes its own execution. The `--as-server` option extends the `--hidden` option, and should be used when AIMMS is started with limited privileges by a system service (e.g. through the Internet Information Server). It suppresses all dialog boxes that may appear during startup of AIMMS, as well as during the execution of your model.

Developer versus end-user mode

With the *--end-user* option you can force AIMMS to start up a project in end-user mode using a developer license, allowing you to preview your application as if you were an end-user without the need to explicitly export an end-user project. Please note that the option to emulate end-user model using an AIMMS developer license will not work, unless it has been enabled in your AIMMS developer license.

Exporting an end-user project

Through the *--export-to* option you can instruct AIMMS to create an encrypted end-user project either packed to the AIMMSfile file specified, or unpacked into a specified folder. When using this commandline option, AIMMS will use the export settings as saved by the previous call to the **File-Export End-User Project** menu. You can use this commandline option, for instance, within the context of a continuous integration server, to automate the deployment of your AIMMS application after new commits have been pushed to the version control repository managing the project.

Specifying the unpack folder

When running an AIMMSfile file, AIMMS will ask for the folder where you want the AIMMSfile file to be unpacked. Alteratively, you can already specify the unpack folder through the *--unpack-folder* commandline option.

Solverless AIMMS sessions

AIMMS strictly enforces that the number of AIMMS sessions with full solving capabilities running on your computer simultaneously is in accordance with your AIMMS license. Typically, for a single-user license, this means that you can only start up a single AIMMS session that is capable of solving optimization programs at a time. However, for every fully capable AIMMS session, AIMMS also allows you to start up an additional AIMMS session without solving capabilities. You can use such a session, for instance, to make modifications to your model, while a first session is executing an optimization run. In that case, AIMMS will present a dialog box during start up to indicate that the session has no solving capabilities. You can suppress this dialog box, by specifying the *--no-solve* command line option.

Executing a procedure and terminating AIMMS

When you want to run an AIMMS project unattended, you can call AIMMS with the *--run-only* option. This option requires the name of a procedure in the model, which will be executed after the project is opened. When you use the *--run-only* option, all other initial project settings, such as the initial case, procedure and page settings, will be ignored. AIMMS will, however, call the procedures `MainInitialization`, `PostMainInitialization`, `PreMainTermination`, `MainTermination`, and all library initialization and termination procedures as usual. Once the procedure has finished, the AIMMS session will be terminated. You can only specify the *--run-only* option if you also specify a project file on the command line.

Opening a project to run

AIMMS will interpret the first non-option argument on the command line as the name of the project file with which you want to open AIMMS. If you specify a project file, the settings of the project may initiate model-related execution or automatically open a page within the project.

Opening a project to edit

If you want to open a project for editing purposes only, you should hold down the **Shift** key when opening the project. The initial actions will also not be performed if the command line contains the *--run-only* option. In this case execution takes place from within the specified procedure only.

Passing session arguments

Directly after the name of the project file, AIMMS allows you to specify an arbitrary number of string arguments which are not interpreted by AIMMS, but can be used to pass command line information to the project. In the model, you can obtain the values of these string arguments one at a time through the predefined function `SessionArgument`, which is explained in more detail in *Execution Control Functions* (page 126).

Example

The following call to AIMMS, will cause AIMMS to start the project called `transport.aimms` in a minimized state using the user name `batchuser` with password `batchpw`, run the procedure `ComputeTransport`, and subsequently end the session. A single argument `"Transport Data"` is provided as a session argument for the model itself.

```
aimms --minimized --user batchuser:batchpw --run-only ComputeTransport \
      transport.aimms "Transport Data"
```

Note that the \ character at the end of the first line serves as the continuation character to form a single command line. Using the short option names, you can specify the same command line more compactly as

```
aimms -mUbatchuser:batchpw -RComputeTransport transport.aimms "Transport Data"
```

In this command line, the `-m` and `-U` options are combined. No space is required between a short option name and its argument.

Using session arguments

Given the above AIMMS call, you can use the function SessionArgument to fetch the first session argument and assign it to the string parameter ODBCDataSource as follows.

```
if ( SessionArgument(1, ODBCDataSource) ) then
    /*
     *  Execute a number of READ statements from ODBCDataSource
     */
endif;
```

Following this statement, the string parameter ODBCDataSource will hold the string "Transport Data". In this example, the string parameter ODBCDataSource is intended to serve as the data source name in one or more DATABASE TABLE identifiers, from which the input data of the model must be read.

4.2.2 Calling AIMMS from External Applications

Use AIMMS as a component

In addition to starting the AIMMS program itself, you can also link AIMMS, as a component, to your own application. Using AIMMS as a component has the advantage that, from within your program, you can easily access data with AIMMS and run procedures in the associated AIMMS project. Thus, for instance, when your program requires optimization, and you do not want to bother writing the interface to a linear or nonlinear solver yourself, you can

- specify the optimization model algebraically in AIMMS,
- feed it with data from your application, and
- retrieve the solution after the model has been solved successfully.

Several options

When linking AIMMS as a component to your own application, you have several options:

- link through a REST API using Data Exchange Library and HTTP Client Library
- link directly against the AIMMS API (see The AIMMS Programming Interface of the Language Reference).

Programming required

Through the AIMMS component technologies described above you have varying degrees of control over the data inside your model. Use of these technologies requires, however, that you set up the interface to your model in a programming language such as C, C++, Java or .NET. While the control offered by these technologies may be relevant for advanced or real-time applications where efficiency in data communication is of the utmost importance, these technologies come with a certain learning curve, and if you only want to perform simple tasks such as communicating data in a blockwise manner and running procedures inside the model, you might consider setting up the communication using either text data files or databases.

Using the AIMMS API

Please note that using the AIMMS API to start up a new AIMMS session from within an external application that also performs other significant tasks than starting up that AIMMS session, is *not recommended*. Opening an AIMMS project from within another application may, especially under Windows, lead to unwanted interactions between the AIMMS and the original application. The AIMMS API is also not particularly suited to start up an AIMMS session from within the same process multiple times. In such cases we advise to use a technology that starts up an AIMMS session in a separate process.

4.2.3 The AIMMS Command Line Tool

AIMMS command line tool

Next to accessing AIMMS from within your own programs through the AIMMS component technologies, AIMMS also supports a command line tool through which you can control an AIMMS project externally. You can start the AIMMS command line tool by running

`AimmsCmd` project-path

The `AimmsCmd` program is located in the `Bin` directory of your AIMMS installation.

Commands

The AIMMS command line tool offers commands to

- assign values to sets, and to scalar and multidimensional identifier slices,
- display the contents of sets, and the values of scalar and multidimensional identifier slices,
- empty sets or multidimensional identifier slices,
- retrieve the cardinality of sets or multidimensional identifier slices,
- run procedures,
- execute system commands, and
- close the AIMMS project and quit the program.

Each command is terminated by a semicolon.

Assignments

You can assign a value to sets and multidimensional identifiers and slices thereof through one of the commands

`Let` reference `:=` data-expression `;`
`Let` reference += data-expression `;`

where the `:=` operator refers to completely replacing the contents of *reference* and the += operator refers to a merge operation.

References

A *reference* in an assignment is either

- an identifier name such as "Transport", or
- a reference to an identifier slice such as

  ```
  Transport('Amsterdam',j)
  ```

  where each sliced dimension must refer to a quoted set element.

Data expressions

The *data expressions* allowed in an assignment are

- a set expression preceded by the keyword Set as in

  ```
  Set {'Amsterdam', 'Rotterdam'}
  ```

  where all set elements must be quoted,
- a ranged integer set preceded by the keyword Set as in

  ```
  Set {1 .. 10}
  ```
- a scalar numeric, element or string value as in

  ```
  10
  11.7
  'an element'
  "a string"
  ```
- a tuple list of numeric, element or string values preceded by the keyword List as in

  ```
  List {('Amsterdam','Paris') :  10, ('Paris','London') :  20}
  ```

  List keyword may be optionally preceded by the keyword Strict. In this case using an element name not present in the domain set will trigger an error (it will be added automatically to the domain set otherwise),
- a dense multidimensional array of numeric, element or string values preceded by the keyword Array as in

  ```
  Array [[1,2],[3,4],[5,6]]
  ```

Value display

You can request AIMMS to display the contents of sets and multidimensional identifier slices in your model through the command

Display reference [:precision] [as Array] ;

For multidimensional identifier data AIMMS will, by default, use the List format described above. Through the optional "as Array" clause you can instruct AIMMS to display the identifier data as a dense array.

Empty identifiers

To empty the data of sets and multidimensional identifier slices in your model you can use the command

```
Empty reference ;
```

Identifier cardinality

You can request AIMMS to retrieve the cardinality of sets and multidimensional identifier slices in your model through the command

```
Card reference ;
```

Run procedures

With the command

```
Run procedure-name ;
```

you can request AIMMS to run a procedure (without arguments). When finished, AIMMS will display the return value of the procedure.

Executing system commands

You can let AIMMS execute a system command through the command

```
System system-command ;
```

where *system-command* is a string to be executed by command shell.

Help

Through the `Help` command, a list with a brief description all available commands will be displayed.

Closing the project

You can close the AIMMS project and quit the command line tool through the command

```
Quit ;
```

4.3 Project Security

Project security

When you are creating a model-based end-user application there are a number of security aspects that play an important role.

- How can you protect the proprietary knowledge used in your model?
- How can you prevent the end-users of your application from modifying the project (thereby creating a potential maintenance nightmare)?
- How can you distinguish between the various end-users and their level of authorization within your application?

AIMMS offers several security-related features that address the security issues listed above. These features allow you to

- encrypt the source code of your model,
- introduce authorization levels into your model, and
- set up an authentication environment for your application.

This chapter describes these mechanisms in full detail, together with the steps that are necessary to introduce them into your application.

4.3.1 Encryption

Encryption ...

If you want to protect your investment in model development, the easiest way to accomplish this protection is to use the encryption scheme discussed in this section. Note that project access to the project and model is unconditionally prohibited in an encrypted project, even by the developer of the model himself.

Several ways of encryption

AIMMS supports several manners of encryption of project and model source files, including your model source. Please note that AIMMS will only encrypt `.aimms`, `.libprj`, and `.ams` files. All other files that are exported (including user files that reside in your project file) are not encrypted. It is up to you to choose the encryption scheme that works for you.

Standard encryption results in an end-user version of your application that can be run by everybody.

Password protected encryption results in an end-user version of your application that can be run by anyone who knows the password. Upon starting of the application the user is prompted for the password.

Key-based encryption result is an end-user version of your application that can only be run by users whose public key was present in the key folder that was specified during encryption. The users need to store their private key in the `ApplicationKeys` folder on their local system or, in case a license server is being used, on the system on which the license server is running.

Exporting your project

To ship your application for end-user deployment you should export your application as a single AIMMSfile file (see also *Distributing an AIMMS app* (page 103)). By combining the export with one of the available encryption schemes you simply produce an ready-to-ship version of your application in which the source of your project and model files is securely protected.

Export ...

You can create such a single AIMMSfile file version of your application through the **File-Export End User Project** menu, which will open a **Select Destination .aimmspack file** dialog box. This dialog box requires you to specify the location and name for the AIMMSfile file.

. . . and encrypt

Having specified the name for the AIMMSfile file, the **Encryption of Exported End-User Project** dialog box (as illustrated in Fig. 4.1) opens and allows you to add encryption to the exported version of your application.

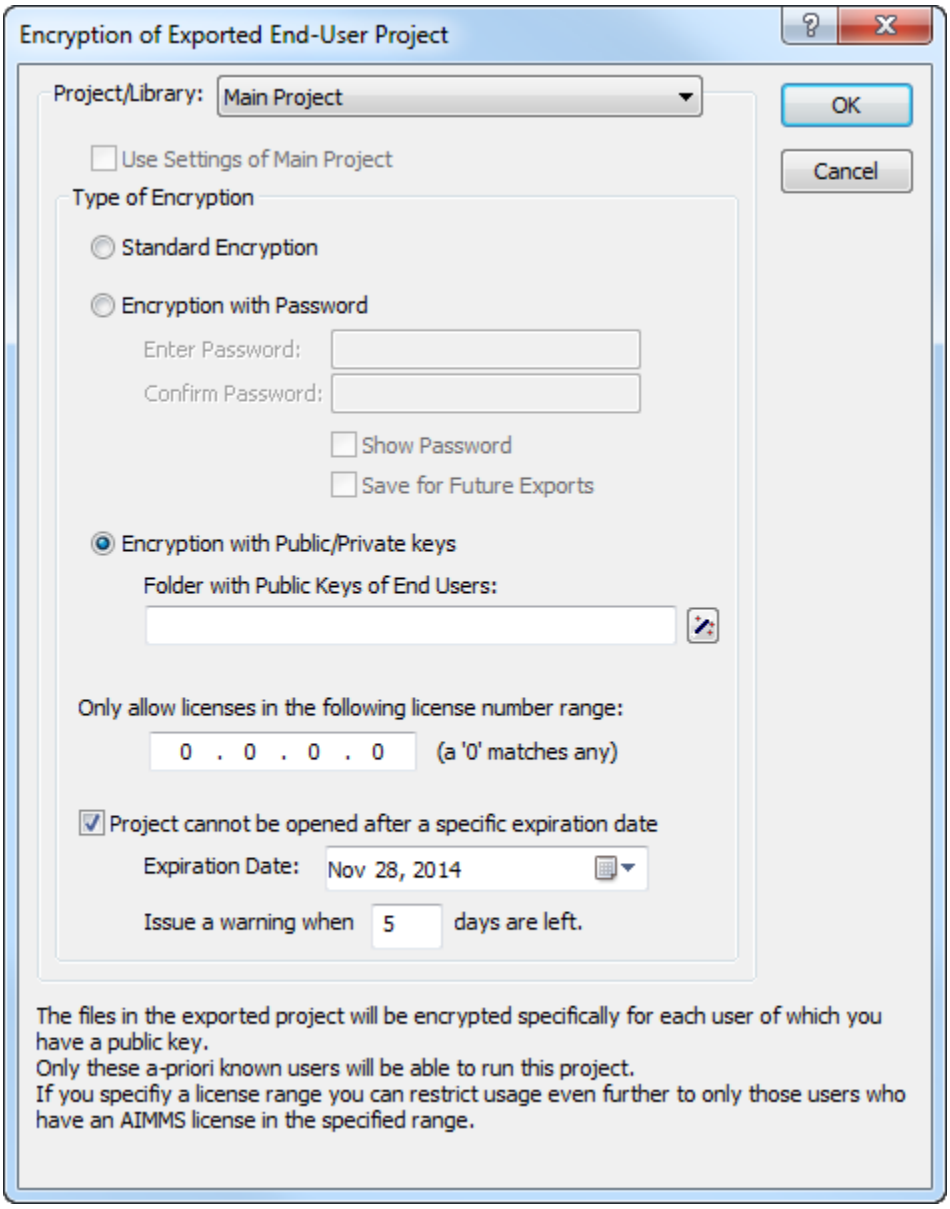


Fig. 4.1: The **Encryption of Exported End-User Project** dialog box

Select one of the available encryption schemes and specify all relevant missing information (e.g. passwords, a folder containing the public keys of your users).

Restrict access to a specific license number

In addition to encrypting your application, you can restrict access to your application such that only users whose AIMMS' license number lies within a specified range can run the application. This prevent the application from being run by other AIMMS users, even in case a password or private key has been compromised.

Add an expiration date

If you add an expiration date to the encrypted application, AIMMS will not allow your end-user to run the application after that specific date. In addition, you can have AIMMS warn your end-user about the expiration date if the application is started within a specified number of days of the expiration date.

Public Key Encryption

Public vs. private keys

AIMMS' key encryption uses a common public key algorithm which assumes the presence of two associated keys, a *public key* and a *private key*. Anyone who has access to a certain public key can encrypt data, but only the owner of the corresponding private key can decrypt the data. So, if you want someone to send you encrypted data, you should share your public key. At all means, a private key should be kept private.

Creating a key pair

Through the **Tools-License-Generate Public/Private Key Pair** menu, you can generate two associated key files.

Encryption using multiple public keys

An application can be encrypted using a collection of public keys. The resulted encrypted application can then only be run by any private key, matching one of the public keys in the collection that was used during decryption.

Private key folder

When attempting to decrypt an application, AIMMS will look for matching private keys in the `AIMMS\ApplicationKeys` folder. The folder is located as a subfolder of the folder described by the `ProgramData` Windows environment variable. On a typical Windows 7 or Windows 8 system, this private key folder is `C:\ProgramData\AIMMS\ApplicationKeys`. In case a license is provided over the network by an AIMMS network license server, the private key to decrypt the application may also be present on the system that runs the license server. In case the private key is provided by the license server, only users that are granted access to a network license on the server, may use the private key from the server.

Encrypting Your Application: Some Use Cases

Use an existing public key

To encrypt an application for a specific user that has already created his own key pair, just request the user for a copy of his public key and use the public key to encrypt your application.

Use as application license

In case you generate a new public/private key pair yourself and use the newly generated public key to encrypt your application, the corresponding private key serves as an *application license*: As soon as you provide an AIMMS user with this private key (and access to the encrypted version of your application), he will be able to run the application. In this scenario, it is even possible to generate a collection of key pairs in advance and distribute a new *application license* anytime you get a new user for your application.

Use in an AIMMS PRO environment

When publishing an application on a AIMMS PRO server, you are advised to encrypt your application using the public key of the AIMMS network license server that is used in the PRO configuration. After that, any user who has been granted access to the PRO server (and the specific application), is able to run the the encrypted application, without the need to have a public/private key pair of his own.

4.3.2 User authentication and authorization

User authentication

When an application is set up for use by multiple users through AIMMS PRO, it is usually considered desirable to limit access to the application to particular (groups of) users, make sure that users have access to only those parts of the application that are of interest to them, and can be given or denied the right of access to each others data.

Authorization via AIMMS PRO portal

When publishing an application on a AIMMS PRO server, you can manage access to your application through the AIMMS PRO portal. The AIMMS PRO documentation User Management decribes more details about the setup of users and groups for your application.

Authorization via model

Next to arranging access to your application application-wide through the PRO portal, the PRO system library extends your model with functionality to access user- and group-related information from within your AIMMS application. More, specifically, through the PRO library function

```
pro::GetCurrentUserInfo
```

you can retrieve the currently connected PRO user name and the PRO group membership of the currently connected user.

Role-based security

Using the PRO user name and groups discussed above, you can set up your own customized role-based security scheme within your application. You can accomplish this by associating roles within your application with group membership of particular groups defined through the user management facilities in the AIMMS PRO portal. If PRO user management is linked to your Active Directory environmoment, role-based authorization to your application can also be arranged directly through your company's Active Directory environment.

Example

Assume that ExecutionAllowed is a two-dimensional parameter defined over a set AllApplicationRoles declared in your model, of which the actual set of PRO groups, retrieved via pro::GetCurrentUserInfo, is a subset, and a user-defined set of application-specific ActionTypes. Then the following code illustrates the to allow or forbid a certain statement to be executed in a role-based manner.

```
if ( exists(appRole | ExecutionAllowed(appRole, 'Solve') ) then
    solve OptimizationModel;
else
    DialogError( "None of your application roles does allow you\n" +
                 "to solve the optimization model" );
endif;
```

Use in the interface

You can also use parameters defined over AllApplicationRoles to influence the appearance and behavior of the end-user interface. More specifically, the following aspects of an AIMMS end-user interface can be influenced through the nonzero status of (indexed) parameters:

- the access to a page through the page tree-based navigational controls,
- the visibility of graphical (data) objects on a page,
- the read-only status of data in a data object, and
- the visibility and enabled/disabled status of menu items and buttons.

Note that both the Windows and browser-based AIMMS UIs support such dynamic model-based access controls.

4.4 Project Settings and Options

Several aspects of AIMMS, including its startup behavior, its appearance, the inner workings of the AIMMS execution engine or the solvers used in a session, can be customized to meet the requirements of your project. This chapter describes the various tools available in AIMMS for making such customizations.

4.4.1 AIMMS execution options

Options

Many aspects of the way in which AIMMS behaves during a session can be customized through the AIMMS execution *options*. Such options can be set either globally through the options dialog box, or from within the model using the OPTION statement. As every project has its own requirements regarding AIMMS' behavior, option settings are stored per project in the project file.

See also:

The OPTION and PROPERTY Statements

Option types

AIMMS offers options for several aspects of its behavior. Globally, the AIMMS execution options can be categorized as follows.

- **Project options:** how does AIMMS behave during startup, and how does AIMMS appear during a project.
- **Execution options:** how does the AIMMS execution engine with respect to numeric tolerances, reporting, case management and various other execution aspects.
- **General solver options:** how does AIMMS behave during the matrix generation process, and which information is listed.
- **Specific solver options:** how are the specific solvers configured that are used in the project.

Option dialog box

Through the **Settings-Project Options** menu you can open the global AIMMS **Options** dialog box illustrated in Fig. 4.2.

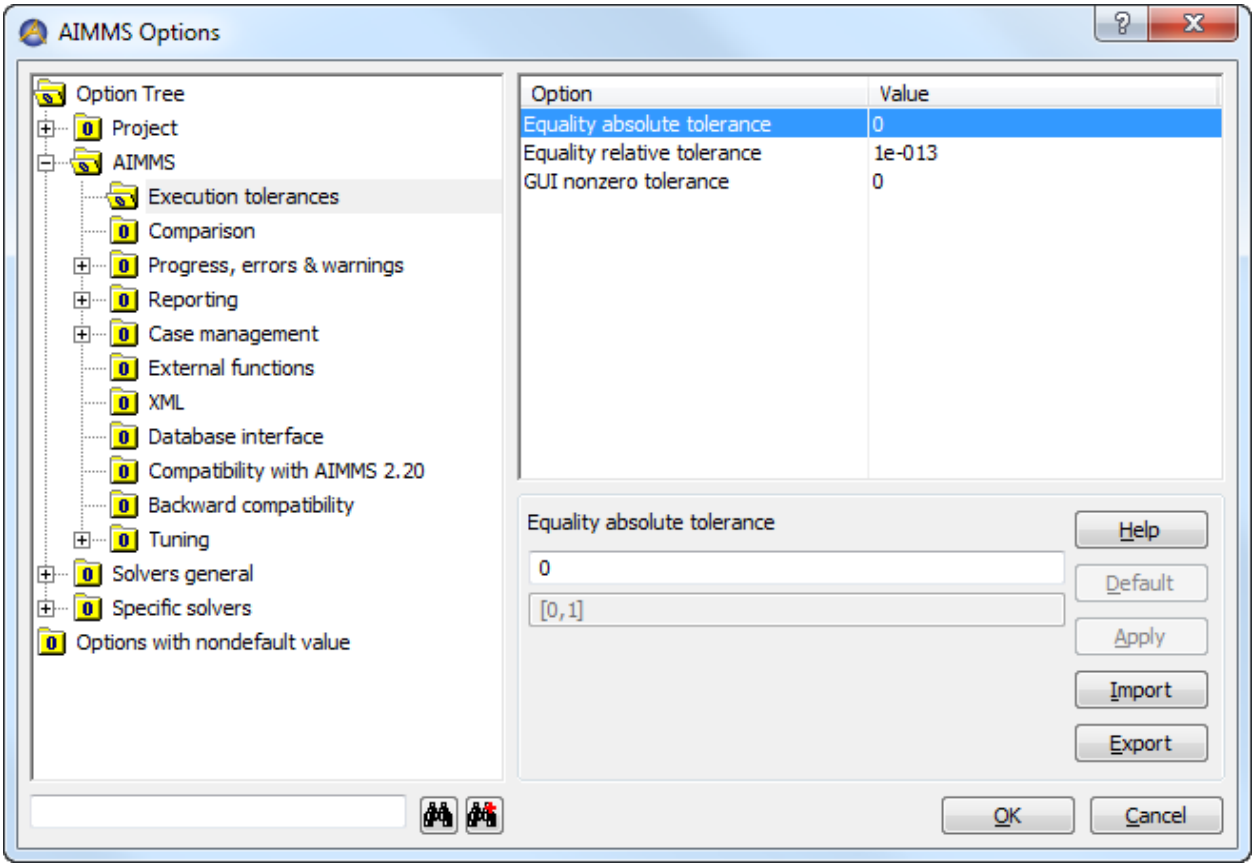


Fig. 4.2: The AIMMS **Options** dialog box

In this dialog box, an *option tree* lists all available AIMMS execution and solver options in a hierarchical fashion.

Modifying options

After selecting an option category from the left-hand side of the **Options** dialog box, you can modify the values of the options in that category on the right-hand side of the dialog box. As illustrated in Fig. 4.2, AIMMS lists the currently selected value for every option (in the first edit field) along with the allowable range of all possible option values (in the second field). Option values can be either integer numbers, floating point numbers or strings, and, depending on the option, you can modify its value through

- a simple edit field,
- radio buttons,
- a drop-down list, or
- a wizard in the case where the value of an option is model-related.

Committing options

With the **Apply** button, you can commit the changes you have made to the value of a particular option and continue changing other options; the **OK** button will commit the changes and close the option dialog box. With the **Default** button at the right-hand side of the dialog box, you can always reset the option to its default value. It is only active when the option has a nondefault value.

Option description

When you have selected an option, and need to know more about its precise meaning before changing its value, you can press the **Help** button at the right-hand side of the options dialog box. As illustrated in Fig. 4.3,

this will open a help window containing a more detailed description of the selected option.

Options with nondefault value

To help you quickly identify all the options which you have modified for a particular project, all modified options are summarized at the end of the options tree in a special section, **Options with nondefault value**. You can modify these options either in this section, or in their original locations. If you set a modified option back to its default value, it will be removed from the nondefault section. When you select an option from the **Options with nondefault value** section, the  **Location in Tree** button will become available. Pressing this button will select the originating option category in the option tree.

Copying solver options

When you add a new version of some solver to the solver configuration (see *Solver configuration* (page 145) for a description of how to add a new solver), the options of this new solver will appear in the **Specific Solvers** category. To copy solver options from the old solver version (e.g. Cplex 11.1 to CPLEX 12.6), select the source solver in the option tree and select the **Copy Option** command from the right-mouse popup menu. This will open the **Copy Options** dialog box as shown in Fig. 4.4.

By default this dialog will only show options that differ between both solvers plus options that are only available in one of the two solvers. Once you press the **Ok** button, all options that remain in this list (and are available in both solvers) are copied from the source to the destination solver.

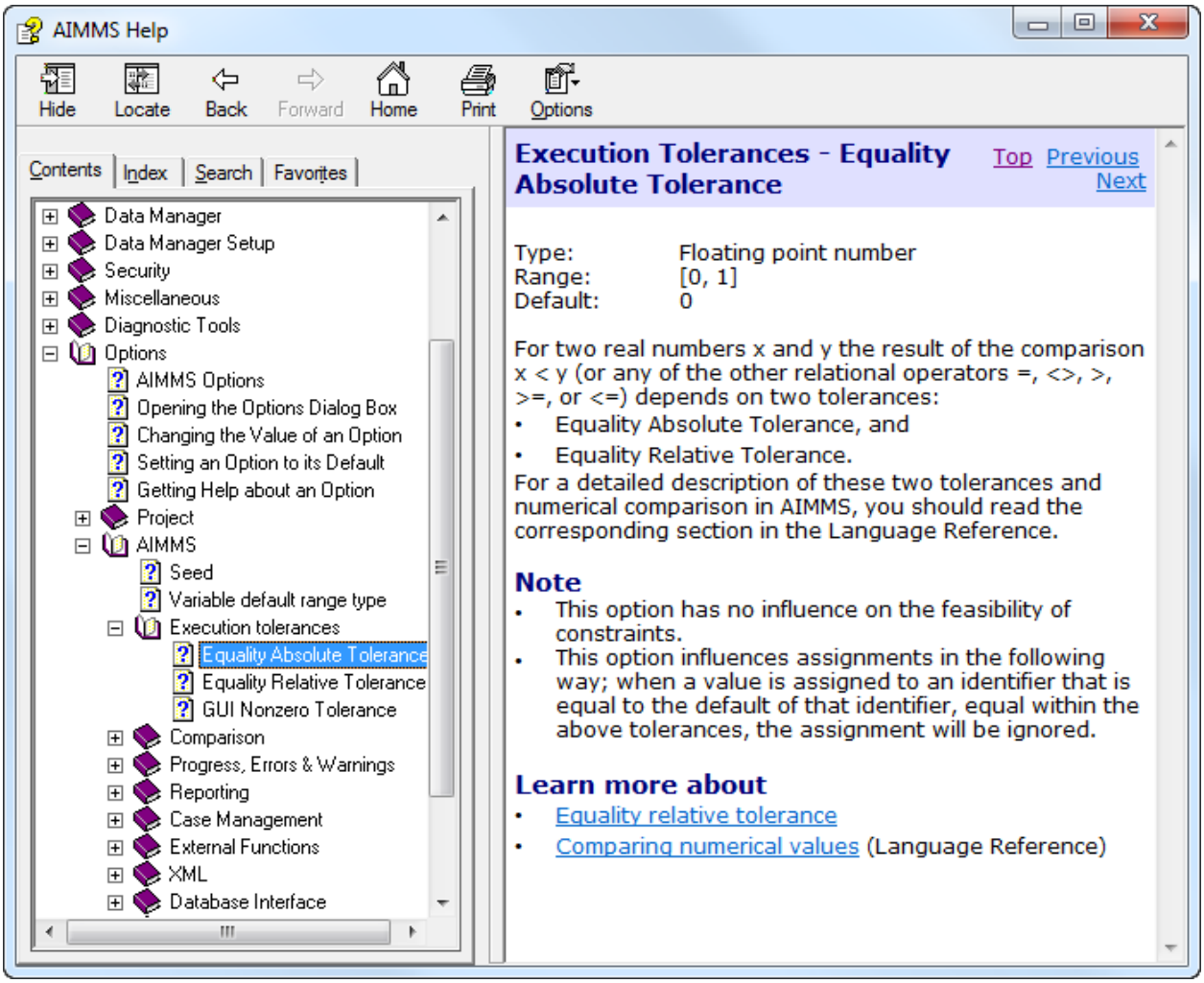


Fig. 4.3: Option help

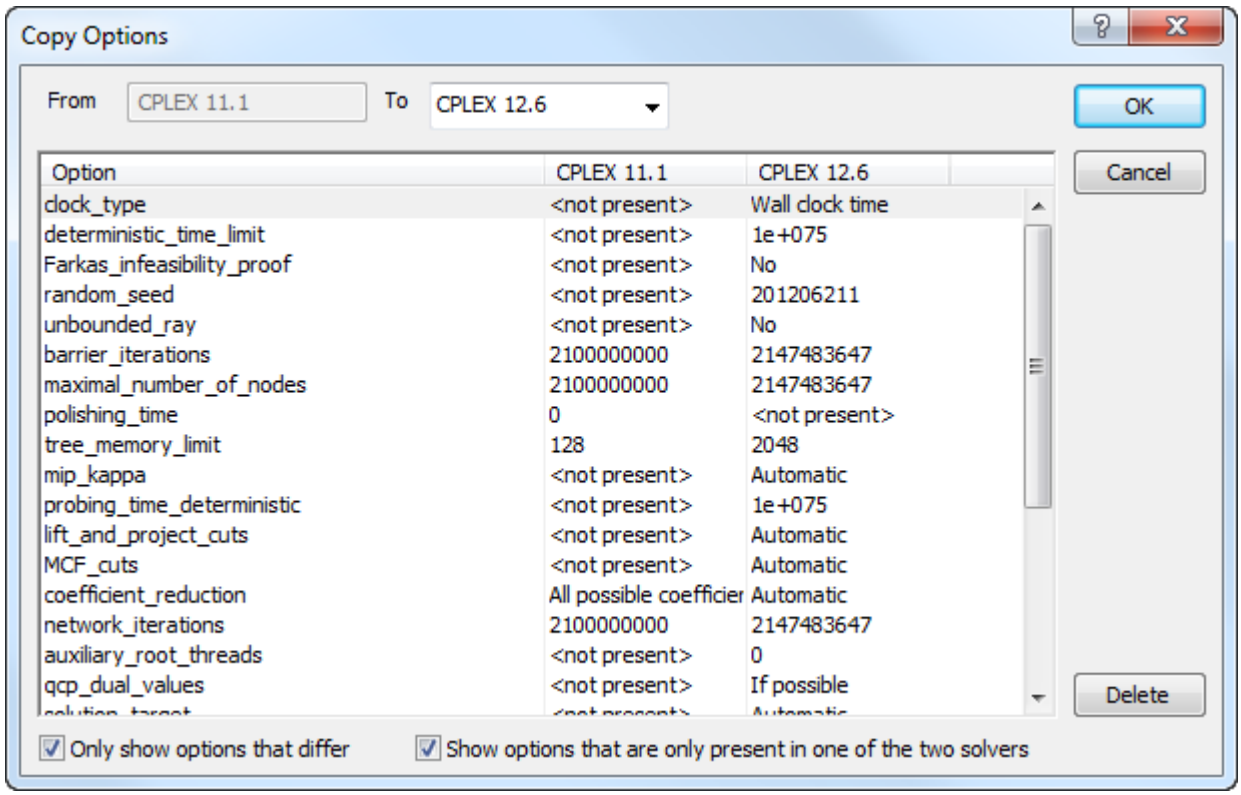


Fig. 4.4: The **Copy Options** dialog box

Searching for options

When you know (part of) the name of an option, but do not know where it is located in the option tree, you can use the search facility in the lower left- hand part of the option dialog box to help you find it. When you enter (part of) an option name, AIMMS will jump to the first option in the tree whose name contains the entered string.

Setting options within the model

In addition to modifying option values in the options dialog box, you can also set options from within your model using the `OPTION` statement. The `OPTION` statement is discussed in The OPTION and PROPERTY Statements. While changes to option values in the options dialog box are stored in the project file and reused at the beginning of the next project session, run time option settings are lost when you close the project. Setting options during run time can be convenient, however, if different parts of your model need different option settings.

4.4.2 End-user project setup

Setting up an end-user project

A number of options and settings are of particular importance when you want to set up a project in such a manner that it is ready to be used by end-users. You can find these options in the **Project-Startup & authorization** and the **Project-Appearance** sections of the **Options** dialog box. This section discusses the most important options.

Startup procedure

With the *startup procedure* option you can select a procedure within your model which you want to be executed during the start up of your project. Such a procedure can perform, for instance, all the necessary data initialization for a proper initial display of the end-user GUI automatically, thus preventing your end-users from having to perform such an initialization step themselves.

Startup page

With the *startup page* option, you can indicate the page which AIMMS will display at start up. It is important to specify a startup page for end-user projects, as all data communication with the model must take place through end- user pages designed by you. Therefore, you should also ensure that every relevant part of your application can be reached through the startup page.

Startup by-pass

In a developer project you can by-pass the startup sequence by holding down the Shift key when you select the project to be opened.

Project title

By default, AIMMS will display the name of the currently loaded project in the title bar of the AIMMS window. Using the *project title* option you can modify this title, for instance to provide a longer description of your project.

4.4.3 Solver configuration

Configuring solvers

With every AIMMS system you can obtain a license to use particular solvers to solve mathematical programs of a specific type. As AIMMS provides a standardized interface to its solvers, it is even possible for you to link your own solver to AIMMS. This section provides an overview of how to add solvers to your system or modify the existing solver configuration.

Solver configuration dialog box

You can obtain a list of solvers currently known to your AIMMS system through the **Settings-Solver Configuration** menu. This will open the **Solver Configuration** dialog box illustrated in Fig. 4.5.

The dialog box shows an incidence matrix between all available solver and types of mathematical programs. An 'x' indicates the capability of a specific solver to solve mathematical programs of a particular type. A bold '**X**' indicates that the specific solver is used as the default solver for mathematical problems of a particular type.

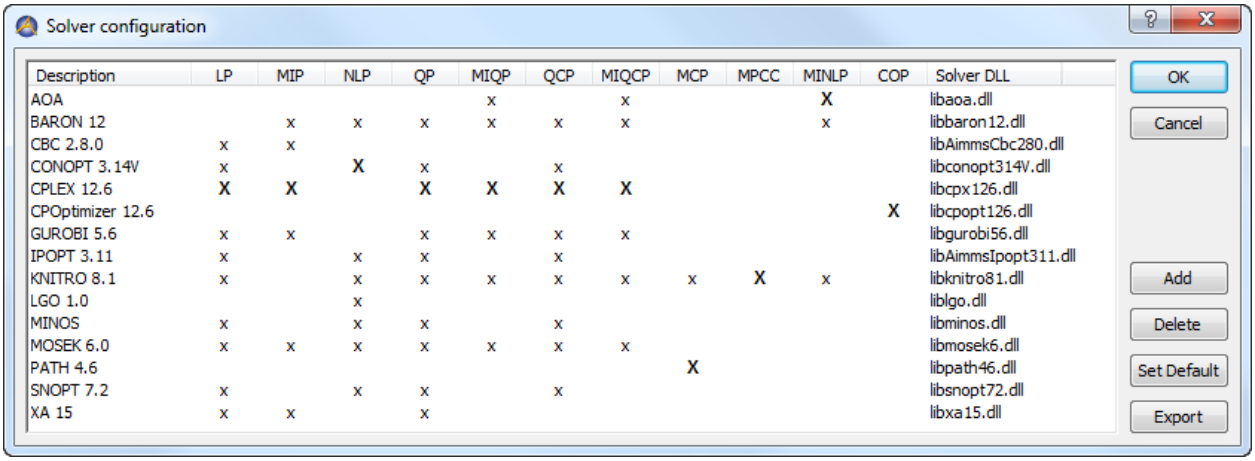


Fig. 4.5: The **Solver Configuration** dialog box

Modifying solver settings

The buttons on the right-hand side of the dialog box let you globally modify the solver configuration of your AIMMS system. Through these buttons you can perform tasks such as:

- modify the default solver for a particular model type, and
- add or delete solvers.

Selecting default solver

With the **Set Default** button you can set the default solver for a particular type of mathematical program. AIMMS always uses the default solver when solving a mathematical program of a particular type. A run time error will occur, if you have not specified an appropriate solver.

Adding a solver

When you want to add an additional solver to your system, you can select the **Add** button from the **Solver Configuration** dialog box, respectively. This will open a **Solver Configuration Data** dialog box as shown in Fig. 4.6.

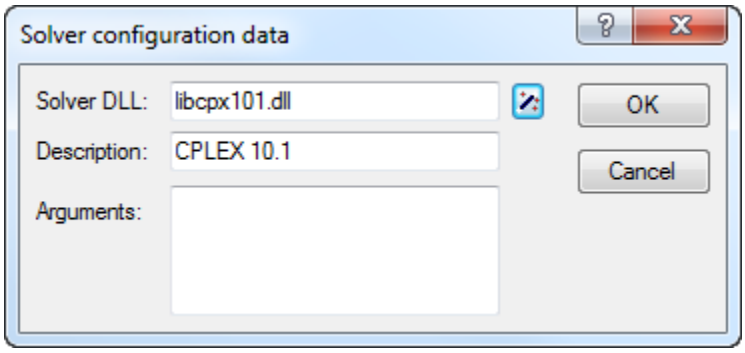


Fig. 4.6: The **Solver Configuration Data** dialog box

In this dialog box you have an overview of the interface DLL, the name by which the solver is known to AIMMS and any appropriate arguments that may be needed by the solver.

Select solver DLL

In the **Solver DLL** area of the **Solver Configuration Data** dialog box you can select the DLL which provides the interface to the solver that you want to link to AIMMS. AIMMS determines whether the DLL you selected is a valid solver DLL, and, if so, automatically adds the solver name stored in the DLL to the **Description** field.

Solver arguments

In the **Arguments** area of the **Solver Configuration Data** dialog box you can enter a string containing solver-specific arguments. You may need such arguments, for instance, when you have a special licensing arrangement with the supplier of the solver. For information about which arguments are accepted by specific solvers, please refer to the help file accompanying each solver.

Installation automatically adds

After you install a new AIMMS version, AIMMS will automatically add the solvers available in that installation to the **Solver Configuration** dialog box. If the newly installed solver is the first solver of a particular type, AIMMS will also automatically make the solver the default solver for that type. Thus, after installing a new AIMMS system, you do not have to worry about configuring the solvers in most cases, provided of course that your AIMMS license permits the use of the solvers you have installed.

Using a nondefault solver

By modifying the value of the predefined element parameter `CurrentSolver` in the predefined `AllSolvers` during run time you can, at any time during the execution of your model, select a nondefault solver for a given mathematical programming type that you want AIMMS to use during the next `solve` statement for a mathematical program of that type. At startup, AIMMS will set `CurrentLPSolver` to the default LP solver as selected in the solver configuration dialog box.

4.4.4 Print configuration

Print configuration

AIMMS offers two distinct facilities to create printed reports associated with your model, namely printouts of graphical end-user pages and print pages, and printouts of text files such as a text representation of a part of the model tree or the listing, log and `put` files. This section explains how you can configure the printing properties for both types of reports.

Printing end-user pages

> **Warning:** The AIMMS WinUI is deprecated, please refer to AIMMS Product Lifecycle. You may use the WebUI instead.

End-user pages and print pages are printed according to the settings that you have selected for these pages. These settings include:

- the selection of the paper type on which pages are printed, and
- the selection of object fonts and colors through the AIMMS font and color selection dialog boxes.

These settings must be fixed by you as the application developer, and cannot be changed by an end-user of your application. An end-user can, however, still select the printer to which the output must be sent, as explained below.

Text printing

Text files can be printed from within AIMMS, either from the **File-Print** menu inside an AIMMS text editor window, or through a call to the `FilePrint` procedure from within a procedure in your model. The print properties of all text files that you want to print, in either manner, can be modified through the **Settings-Text Printing** menu. This will invoke the dialog box illustrated in Fig. 4.7.

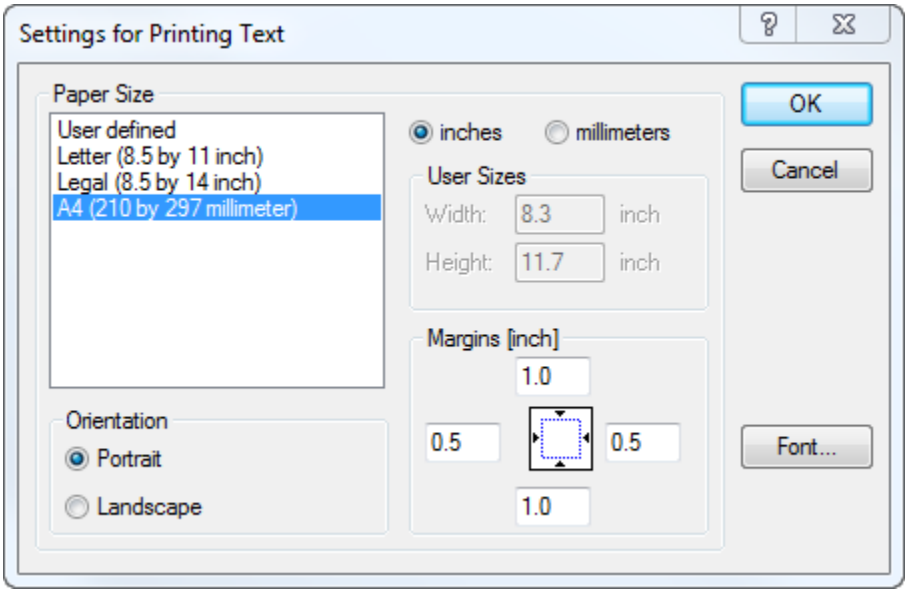


Fig. 4.7: The **Text Printing** dialog box

Text printing properties

In the **Text Printing** dialog box you can select the paper type and font with which you want all text files to be printed. For the paper type you can select one of the predefined paper types, or specify a user defined paper type by providing the page height and width, as well as the margins on each side of the page. By pressing the **Font** button on the right-hand side of the dialog box, you can select the font with which you want your text files to be printed. The text printing properties are stored globally on your machine.

Printer setup

With the **File-Print Setup** menu you can select the printer on which print pages and text files associated with your project are printed, and modify the properties of that printer. This command will invoke the standard Windows **Print Setup** dialog box illustrated in Fig. 4.8.

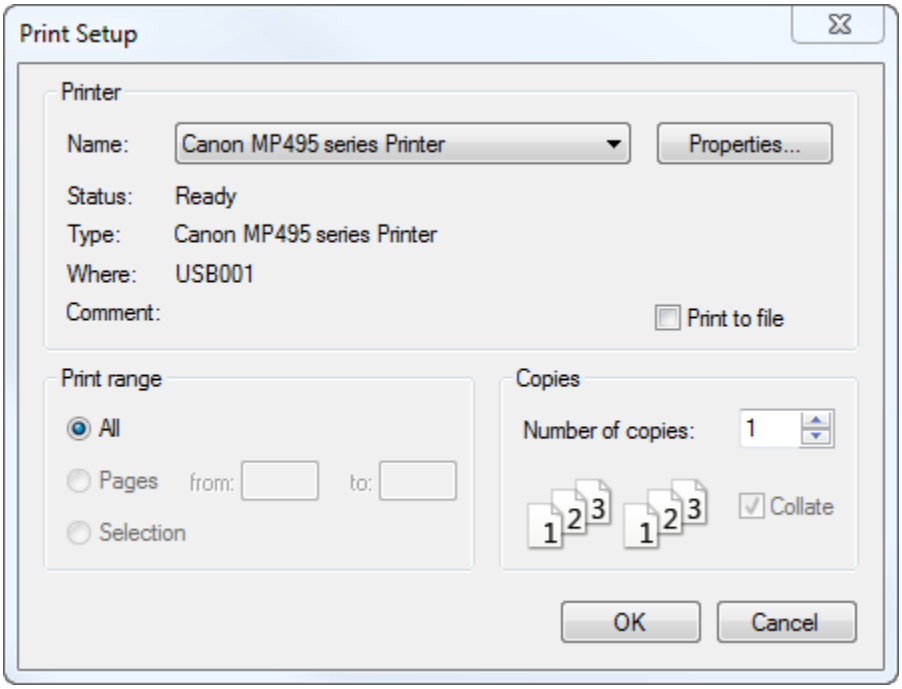


Fig. 4.8: The **Print Setup** dialog box

Default settings

The settings selected in this dialog box will only be valid during the current session of AIMMS. If you want to modify the default print setup globally, you can do this through the **Printer** section in the Windows **Control Panel**. There you can

- select a **Default** printer from the list of all printers available on your system, and
- modify the **Document Defaults** (i.e. the printer settings with which each print job is printed by default) for every individual printer on your system.

Without a call to the **File-Print Setup** dialog box, AIMMS will use the default printer selected here, and print according to the document defaults of that printer.

4.5 Localization Support

Warning: Some items (not all of them) presented in this chapter are related to AIMMS WinUI, which is deprecated. Please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and the Multi-Language Support.

Interface localization

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.

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.

4.5.1 Localization of end-user interfaces

Warning: Some items (not all of them) presented in this chapter are related to AIMMS WinUI, which is deprecated. Please refer to AIMMS Product Lifecycle. You may use the WebUI instead, and the Multi-Language Support.

Basic concepts

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

- 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.

Localization and libraries

If your application consist of multiple library projects (see also *Collaborative Project Development* (page 13)), 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.

Setting up localization support

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

- 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 Fig. 4.9,

AIMMS will request the names of the 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.

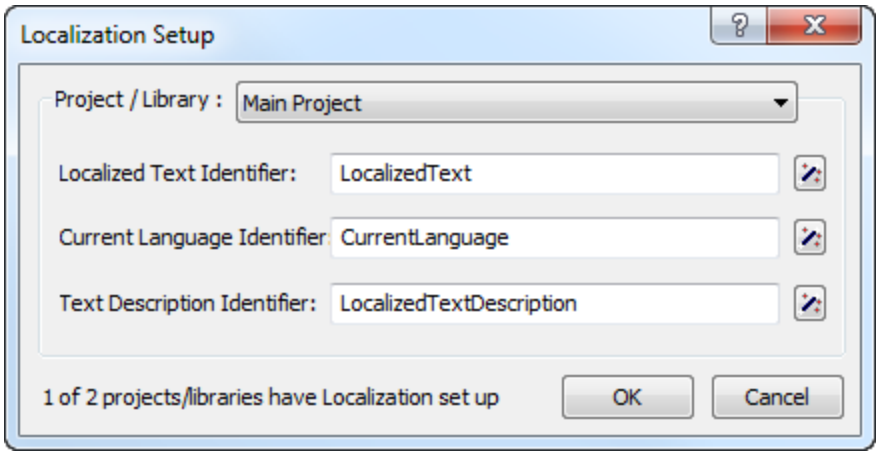


Fig. 4.9: Setting up localization support

Selecting the language

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.

Localization section

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 Fig. 4.10.

The declaration 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.

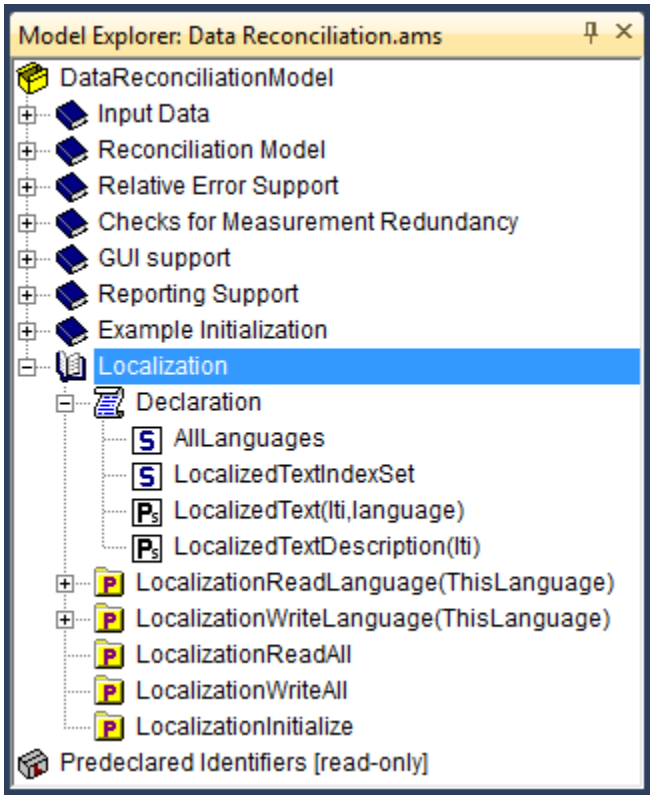


Fig. 4.10: Localization section in the model tree

Using other localization identifiers

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:

- 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.

Localization procedures

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:

- 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 initialization procedure

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).

Added to `MainInitialization`

If your model contains the (default) procedure `MainInitialization` (see also *Creating and managing models* (page 27)), 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.

Performing the localization conversion

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

- 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**.

String description

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.

Duplicate occurrences

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.

Editing localized strings

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 Fig. 4.11.

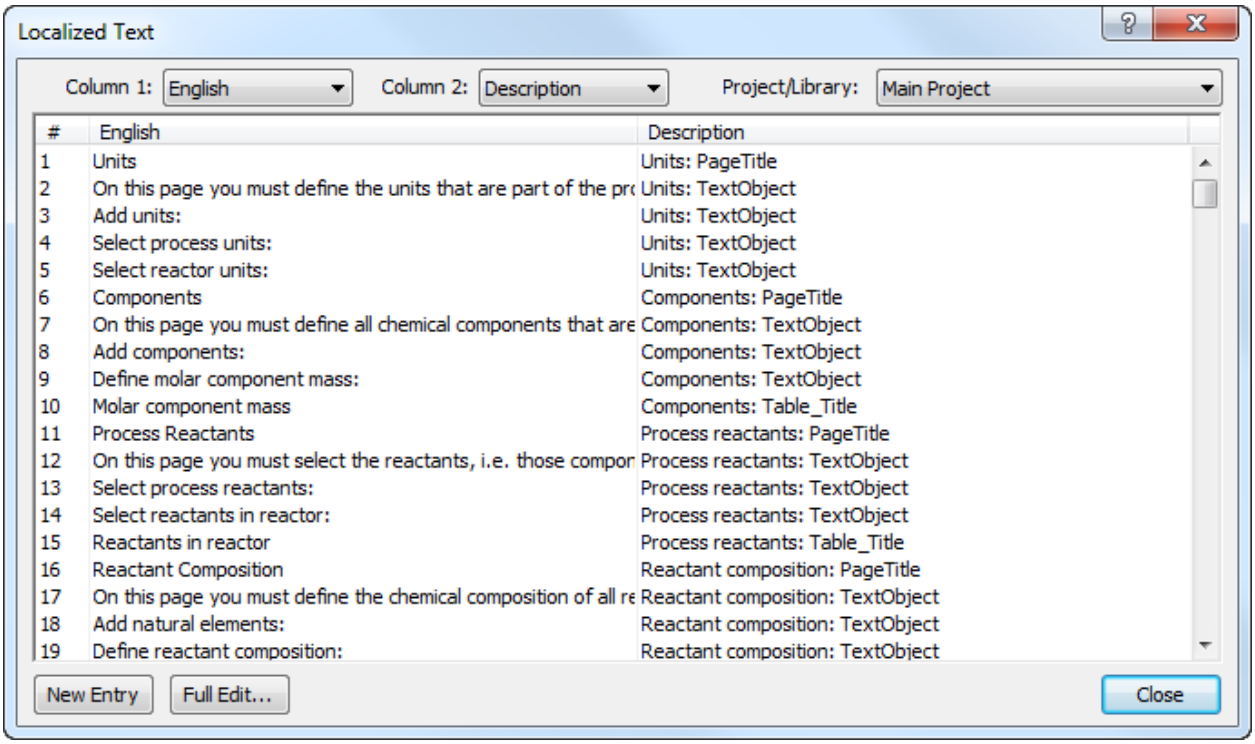


Fig. 4.11: The **Localized Text** dialog box

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.

Modifying dialog box contents

Through the drop down lists at the top of the **Localized Text** dialog box of Fig. 4.11, 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 multiline strings

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 Fig. 4.12.

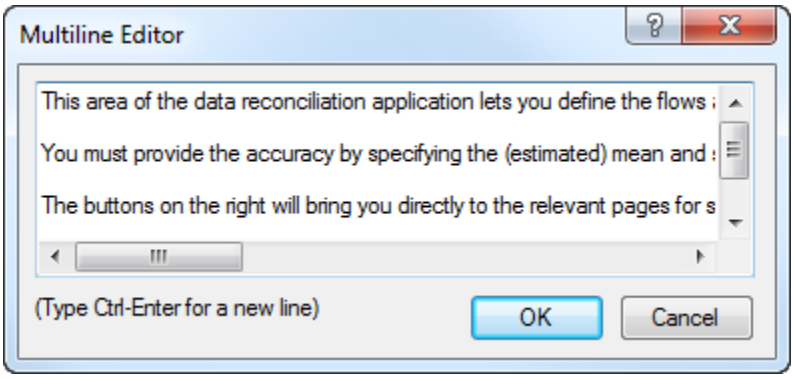


Fig. 4.12: The **Multineline Editor** dialog box

To invoke this multiline 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.

Localizing new texts

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

- 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.

The Localized Text wizard

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 Fig. 4.13

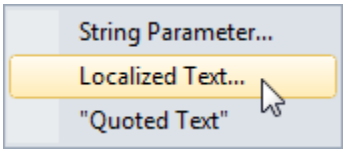


Fig. 4.13: The Localized Text wizard

Invoking this wizard will open the **Localized Text** dialog box illustrated in Fig. 4.11, 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.

Performing the conversion procedure again

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.

Localized text storage

By default, AIMMS stores the localization data as *project user files* containing standard AIMMS data statements within the project file (see also *Project User Files* (page 11)). The localized strings for every language, as well as the string descriptions are stored in separate user project files, as illustrated in Fig. 4.14.

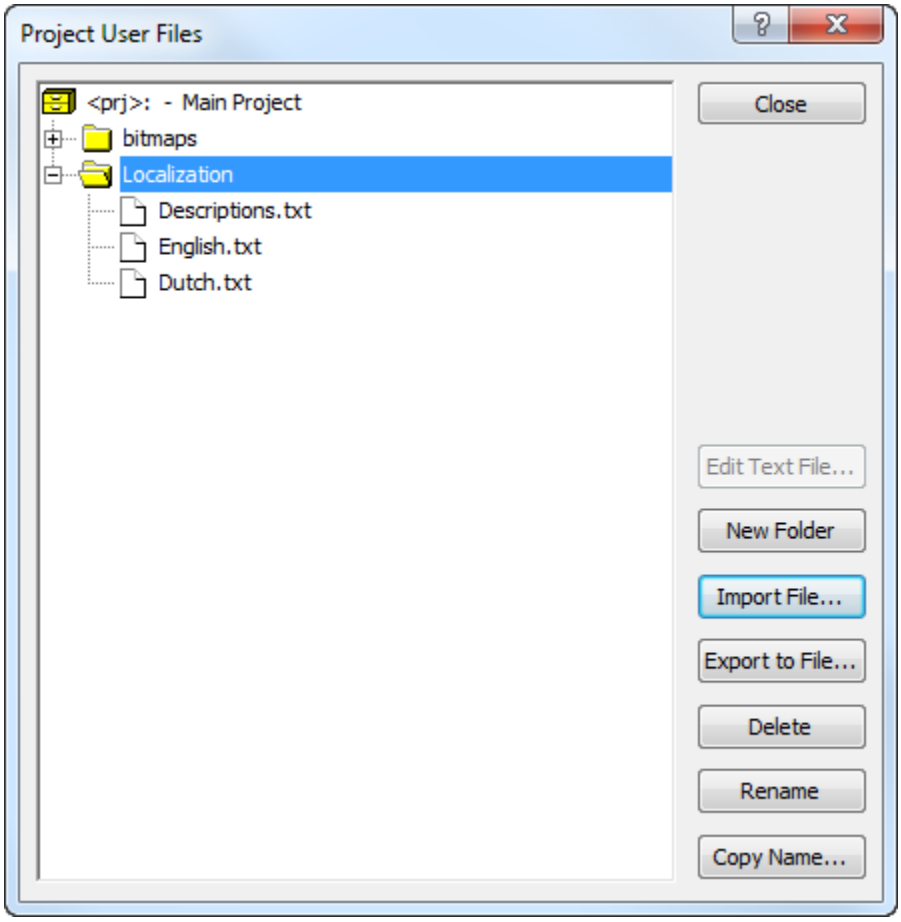


Fig. 4.14: Default of localization data as user project files

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.

Automatically updated

Whenever you use the **Localized Text** dialog box of Fig. 4.11, 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.

Manual edits

By using the import and export facilities for project user files (see also *Project User Files* (page 11)), 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.

Static strings in the model

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.

BIBLIOGRAPHY

[Chi91] J.W. Chinneck. Locating minimal infeasible constraint sets in linear programs. *RSA Journal on Computing*, 3(1):157–168, 1991.

[Gon94] J. Gondzio. *Presolve analysis of linear programs prior to applying an interior point method*. Volume 3. Logilab, HEC Geneva, Section of Management Studies, University of Geneva, Geneva, 1994.

[McC98] B.A. McCarl. A note on fixing misbehaving mathematical programs: post-optimality procedures and gams-related software. *Journal of Agricultural & Applied Economics*, 1998.